Diligent

A DIgital Library Infrastructure on Grid ENabled Technology

Deliverable No D1.1.1:

"Test-bed Functional Specification"

February 2005

Document Information

Project	
Project Title:	DILIGENT, A **DI**gital **L**ibrary **I**nfrastructure on **G**rid **EN**abled Technology
Project Start:	1st Sep 2004
Call/Instrument:	FP6-2003-IST-2/IP
Contract Number:	004260
Document	
Deliverable number:	D1.1.1
Deliverable title:	Test-bed Functional Specification
Contractual Date of Delivery:	28th February 2005
Actual Date of Delivery:	28th February 2005
Editor(s):	CNR – ISTI
Author(s):	Henri Avancini, Leonardo Candela, Pasquale Pagano, Manuele Simi
Reviewer(s):	Paolo Fabriani, Paolo Roccetti (ENG)
Participant(s):	CNR – ISTI, UoA, ETH Zurich, FhG/IPSI, UMIT, ENG, USG, FAST
Workpackage:	WP1.1
Workpackage title:	Test-bed functional and architectural design
Workpackage leader:	CNR - ISTI
Workpackage participants:	CNR – ISTI, UoA, ETH Zurich, FhG/IPSI, UMIT, CERN, ENG, USG, FAST, 4D Soft Ltd.
Est. Person-months:	12
Distribution:	Public
Nature:	Report
Version/Revision:	1.3
Draft/Final	Final
Total number of pages: (including cover)	273
File name:	D1.1.1.doc
Key words:	*Digital libraries, etc.*

Disclaimer

This document contains description of the DILIGENT project findings, work and products. Certain parts of it might be under partner Intellectual Property Right (IPR) rules so, prior to using its content please contact the consortium head for approval.

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of DILIGENT consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 25 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors. (http://europa.eu.int/)

DILIGENT is a project partially funded by the European Union

Table of Contents

Table of Figures

Summary

This report presents the result of the activity conducted within the "WP1.Test-bed functional and architectural design", task "T1.1.1 Test-bed functional specification" of the DILIGENT project in the period November 1st 2004 - February 28th 2005. By exploiting the analysis of the user-communities requirements, collected in WP2.1 and WP2.2, the *Test-bed functional specification* describes and specifies the functions and features of the DILIGENT system that will be perceived by the users and it formalizes how the users will interact with the system.

The notation adopted here is that recommended by the Unified Process software engineering methodology; this formal notation is accompanied by texts as expressed by the DILIGENT technological partners.

Executive Summary

The objective of the task "T1.1.1 Test-bed functional specification" of the "WP1.Test-bed functional and architectural design" is both to identify generic Digital Libraries functionalities that are useful for all the potential DILIGENT user communities and to specific functionalities that must be provided by the Application Layer in order to support the two validation scenarios.

The functional specification produced by this task has to be interpreted as a "functional solution", specifying which system functions are needed to meet the user communities' requirements. In other words, the functional specification describes which is the "visible" or external behaviour of the system and its components.

Firstly, this activity has analyzed the user requirements expressed by the "WP2.1 ARTE scenario" and "WP2.2 ImpECt scenario" in order to identify common basic functionality matching the requirements of the user scenarios; then these functions have been associated with the main functional areas of the DILIGENT project represented by WP1.2-WP1.6; after that, each partner has focused its analysis on the specific aspects in which it is mainly involved and in which it has more competences, contributing in this way to the specification of the functional view of the entire system; finally all contributions has been collected, analyzed, and integrated by CNR, leader of Task 1.1.1, with the support of all the other partners.

The entire specification activity has been conducted according to the guidelines of the Unified Process methodology and using the Unified Modeling Language to formalize functionality dependencies and relations.

This report describes the DILIGENT system functional specification by first describing the concepts that are common to the whole system and then presenting the system specification as the composition of the specifications of six main areas. Each area illustrates how the corresponding set of functionalities relates to the actors and to the other areas. The identified areas are:

- Digital Library Creation and Management. The major functions and features needed to create and maintain virtual digital libraries - i.e. transient DLs based on shared computational, multimedia and multi-type content and application resources - are described in this area. In particular the following group of functionalities is presented: DLs Management, Resources Management, Virtual Organizations Management, Users Management, and Notification Management.
- Content and Metadata Management. This area presents the functions and features needed to support the seamless access and storage of content and content related metadata in the DILIGENT system. In particular the following group of functionalities is described: Objects Management, Collections Management, Workspace Management, URI Management, Storage, Access and Content Management, Metadata Management, and Annotation Management.
- Index and Search Management. This area is related to the functions and features that allow DL users to locate information in DLs in a cost-effective manner, satisfying the level of quality to be met by the overall data retrieval and delivery operation. In particular the following group of functionalities is presented: Index Management, Content Source Description and Selection, Data Fusion, Feature Extraction, Personalization, and Search.
- Process Management. This area presents the main functions and features needed to combine and integrate services into a coherent whole through workflow and process technology. In particular, the following functionalities are described: Design, Create,

and Validate compound services, Update and Optimize compound services, and Run, Monitor, Abort, and Remove compound services.

- Application Specific Management. This area presents the main functions and features needed to address specific user communities' requirements. Moreover it includes the description of the user-system interactions and the composite document generator. In particular the following group of functionalities are presented: Basic Portal Functionalities (e.g. login, object visualization), Portal Management, Courses Management, Workshops Management, Exhibition Catalogues Management, and Reports Management.

The present functional specification will be used by WP1.2-WP1.6 in order to design all the DILIGENT services and define a strategy for their rapid prototyping and testing.

1 INTRODUCTION

This report describes the result of the activity carried out in Task 1.1.1 "Test-bed functional specification". It has been produced by analysing the content of the two reports "D2.1.1 ARTE Scenario Requirements Reports" and "D2.2.1 ImpECt Scenario Requirements Reports", produced respectively by the ARTE and ImpECt communities in the context of "WP2.1 ARTE scenario" and "WP2.2 ImpECt scenario". This analysis has been the foundation for the functional specification phase. This phase has identified both generic DL functionalities that are useful for all the potential DILIGENT user communities and specific functionalities that must be provided by the Application Layer in order to support the two validation scenarios expressed by the ARTE and ImpECt user communities.

The first step in constructing the functional specifications has been to understand the requirements that are common to more than one use case, reviewing each use case, and taking notes of any commonality. Using the result of this step, the analysts have minimized redundancy by creating included, extended, and generalized use cases. Requirements have been made more understandable and easier to maintain, and at the same time a functional decomposition that can be forwarded into the design phase has been defined.

The initial analysis has been conducted by each DILIGENT partner that has focused its understanding on the specific aspects in which it is mainly involved and has more expertise. Then, all contributions have been collected, analyzed, and integrated by CNR, the Task 1.1.1 leader, with the support of all other partners. This activity has been preceded by the identification of a set of key domain specific concepts whose definition has been agreed among all the partners before starting the specification phase.

The entire specification activity has been conducted according to the guidelines of the Unified Process methodology and it has used the Unified Modelling Language to formalize functionalities dependencies and relations.

This report is organised as follows: Section 2 reports the rationale of the functional specification illustrating the rules followed in the in-depth analysis of the use cases and in the use cases management; Section 3 briefly introduces the set of concepts of general interest that are used in all sections of this document; Section 4 reports the Digital Library Creation and Management area illustrating the major functions and features needed to create and maintain virtual digital libraries; Section 5 describes the Content and Metadata Management area presenting the functions and features needed to support the seamless access and storage of content and content related metadata in the DILIGENT system; Section 6 illustrates the Index and Search Management showing the functions and features that allow DL users to locate information in DLs in a cost-efficient manner; Section 7 presents the Process Management area illustrating the main functions and features needed to combine and integrate services into a coherent whole through workflow and process technology; and finally, Section 8 presents the Application Specific Management area inserted to accommodate the main functions and features needed to address the specific ARTE and ImpECt user communities requirements.

2 RATIONALE OF FUNCTIONAL SPECIFICATION

As we reported in the Description of Work document, the Unified Process (UP, in short) methodology has been adopted to drive the DILIGENT system engineering.

According to this methodology, the functional specification begins when the collection of the user requirements ends, and it fills the important role of describing how the new system will meet the items listed in the requirements specifications.

The DILIGENT functional specification has been written with two objectives: i) to complete the established contract with the user communities by means of a clear and formal functional representation of the system, and ii) to provide the design team with all the information they need to begin the system design. It aims at outlining the entire experience of the application, without really getting into the details of implementation, thereby providing the designers (and then, developers) with a comprehensive knowledge base and reference for any question concerning the project. The idea is that kinks in the design be worked out at a conceptual level, and (more importantly) that users get a clear idea of what the system is and how it will work. In the UP methodology, the functional specification defines the Use-Case Model. This is a model of the system's intended functionalities and its environment. Once finalized and approved by the user communities, the functional specification can be used by designers to create a detailed software design document.

The following picture illustrates the role of the Use-Case Model in the whole development process.

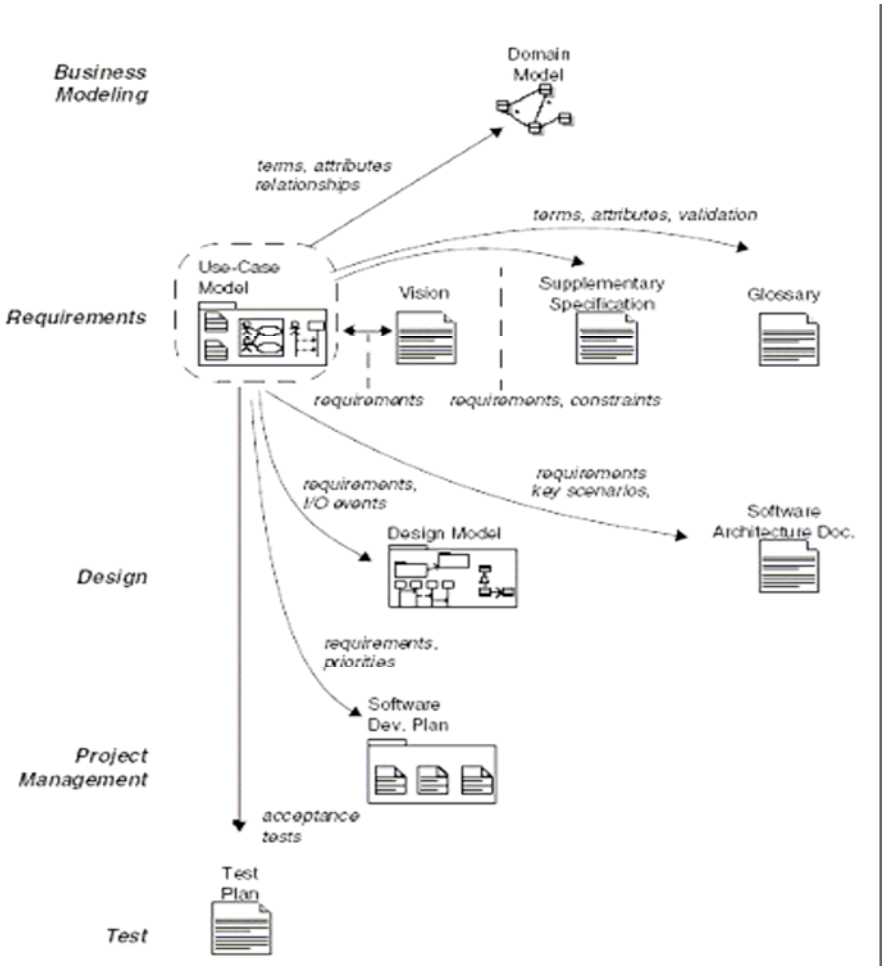

Figure 1: Use-Case Model role in the UP process

"System analysts", i.e. technicians that lead and coordinate requirements elicitation and Use-Case Modelling by outlining the system's functionality and delimiting the system, have performed the functional specification activity. Analysts have identified the user requirements from the use case specification documents. Starting from these documents, briefly summarised in Section 2.2, the produced Use-Case Model defines the system behaviour rather than users requirements. This goal has been achieved by performing a two-steps process, consisting in:

- An in-depth analysis of the use cases
- A use cases management

Both steps are discussed in detail in Section 2.1.

As a result of this activity, the present functional specification document:

- Describes any functionality included in the use cases
- Specifies functionalities not handled by the use case specifications
- Points out system delimitations – things that the system is not supposed to do
- Summarizes important technical facts about the system
- Gives a consistent, correct and realistic functional view of the system

2.1 Methodology

This section explains the steps that have been followed in order to obtain a useful Use-Case Model. It starts with the reporting of the guidelines used to analyze the use case specifications; then it presents how these use cases have been managed; finally it reports the methodology followed to structure the Use-Case Model.

2.1.1 Use Case Analysis

The first step in the analysis has been an in-depth study of the use cases specified by the two DILIGENT user communities in order to fully understand their needs and select, merge, and integrate useful use cases.

The following questions have be addressed in this phase:

- Is each requirement consistent with the overall objective of the system?
- Is the requirement specified at the proper level of abstraction?
- Is the requirement really necessary?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution?
- Is each requirement testable?
- Is the set of requirement consistent?

In the case of a negative response to one or more questions, an assessment phase was performed with the user communities, involved via T2.1.2 and T2.2.2 tasks, in order to refine the user stories.

As the system being built must conform to requirements, and conformance to them decides the success of the project, requirements have also been discarded, modified, and integrated by system analysts, or given low priority when it has been judged that:

- Satisfying them is unfeasible with the current technology
- They are too effort-demanding and not in line with the objectives of the project

Moreover, the user requirements have also been integrated with functionality not included in the use cases, when such functionality has been considered necessary for providing the

expected behaviour of the DILIGENT system also in other contexts not covered by the two selected application scenarios.

The functionalities included in the Use-Case Model thus rise as:

- a direct consequence of a user request: the functionality is in a one-to-one relationship with a user requirement;
- a indirect consequence of a user request: the functionality is needed to reach another functionality requested by users;
- a internal starting point: the functionality has no relation with any use case but it is considered mandatory by the analyst.

The following general specification rules have been established in order to use more easily the results of the distributed analysis carried out by the different partners:

- *A use case contains a segment of behaviour whose result - not the method for getting the result - is of importance to the rest of the use case.*

  The analyst has factored out this behaviour to a new inclusion use case. The include-relationship has thus allowed to clarify a use case by isolating and encapsulating complex details so they do not obscure the real meaning of the use case. Moreover, consistency is improved if the included behaviour appears in several base use cases.
- *A use case has segments of behaviour considered optional or exceptional in character, and not relevant or not necessary to the understanding of the primary purpose of the use case.*

  The analyst has factored those out to a new extension use case. Complex sub-flows and optional behaviour have been the first candidates for being partitioned out into extension use cases. Often this behaviour can be quite complex and hard to describe: including it in the flow of events of a use case can make the "normal" behaviour harder to see. Extracting it allows to improve the comprehensibility of the use-case model.
- *Two or more use cases have similarities in structure and behaviour.*

  The analyst has factored out the common behaviour to create a new parent use case. The original use cases expressed by the user community thus become the child use cases in generalization-relationships with their own parent. Clearly, each child use case inherits all the behaviour described for the parent use case. A generalization-relationship between two use cases means that when a use-case instance follows the description of a child use case, it also needs to follow the description of the parent use case in order to be considered complete.

  Also actors that had common characteristics have been modelled using actor-generalizations.

2.1.2 Use Case Management

After identifying functionalities, the analysts have enriched use cases with a number of information useful to make the system behaviour clear to the system designers.

As UML has been employed as modelling language to support the UP methodology, the following types of diagrams have been used to build the Use-Case Model.

Use Case Diagrams

The Use-Case Model includes use case diagrams capable to model the user requirements. However user cases have been directly translated into use case diagrams only when expressed in a form suitable for the functional specification scope; if not, they have been rearranged, divided, integrated, detailed, and expanded to better describe functionalities.

Moreover, a common vision has also been provided of the two different use case specifications from which the DILIGENT functional specification starts. This vision has been obtained by:

- Starting from one of the two user communities specifications;
- Designing the integration of parts extracted from one of the two user communities specifications;
- Designing a new use case capable of both meeting the user requests and being more appropriate for the project.

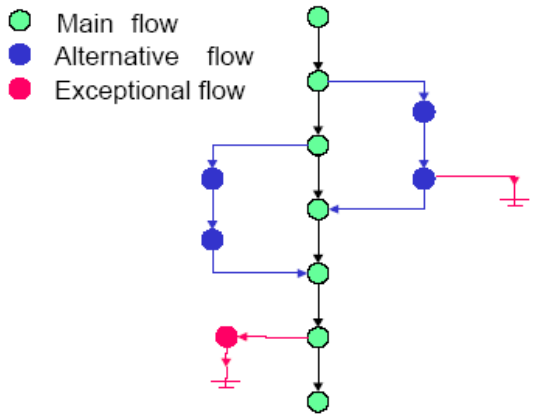

Figure 2: Flows description

Activity Diagrams

A use case usually describes only what it is called the "Main flow", i.e. a set of sequential actions that drive the system from a state to another. On the other hand, designers need to know also any alternative or exceptional flow of activities in order to design a system with a correct behaviour. When this occurs, designers enrich the main flow with the prevision of all the "flows" that a use case may follow. To draw out the primary and alternative paths in the system behaviour, an Activity Diagram has been used.

Sequence Diagrams

Sequence Diagrams are to document usage scenarios and both capture the required objects in the early stage of the analysis and verify the object usage later in the design stage. The DILIGENT analysts have used a Sequence Diagram anchored with a Use Case to further illustrate the sequence of steps taken to realize a use case. Sequence Diagrams have also been used to model the interactions among different use cases.

The following example shows a very basic example about how a Sequence Diagram can be used to describe the behaviour of a use case diagram (which is static by definition) we have derived a Sequence Diagram from a use case diagram.

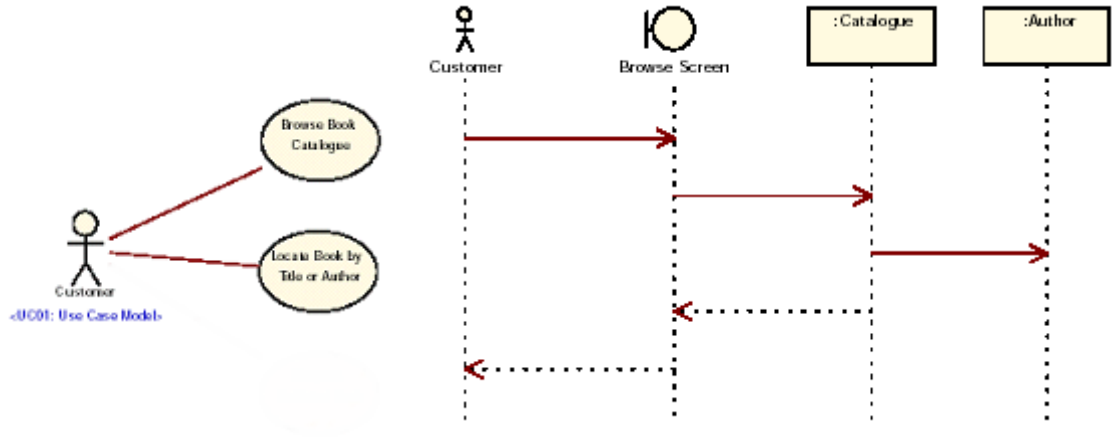

Figure 3: Sequence Diagrams use in the Use-Case Model

Of course, reaching the Sequence Diagrams we are beginning to move into the "how" of system behaviour, switching into a/the detailed analysis, and ultimately in a partial design. In fact, these diagrams provide a preliminary and partial description of the system behaviour modelled in the design phase.

2.1.3 Structuring the Use-Case Model

This section provides the set of guidelines used to write the functional specification. It illustrates the organization of the Use-Case Model and will help designers in using the model in the service specification and design phases.

Writing style

All statements related to functionality have been written as much clearly as possible using consistent terminology. This should let the design be naturally derived - and no interpretation be necessary - and a test plan be easily written, thus ensuring that the final system performs as described.

The functionalities have been grouped, where possible, under sub-headings to make an easily readable and understandable system description. Each group of functionalities has been described using a use case diagram that gives an immediate and formal representation of its behaviour. Then, any functionality represented by one use case has been described textually while including the pertinent subset of the sections listed in the paragraph below. Moreover, a sequence diagram and an activity diagram have been added to specific functionality to improve their readability and usability.

Sections

The following sections have been used as a standard for describing a single functionality of the system.

- Description and Priority

  Contains few sentences that provide a short description of the feature and indicate whether, and motivate why it is of high, medium, or low priority.
- User Requirements Fulfilled

  States the requirement (or requirements) that the functionality is attempting to fulfil. This may be an understanding of a user's requirements or a statement given as an internal starting point. In the first case, the section refers the Use Case specification document section from which the requirement is extracted or derived.
- Functional requirements

Illustrates functional requirements in the form of constraints or pre-requisites in order to better explain the functionality and simplify its use or integration in the flows.

- Numbers

  Details the number of users expected to use the system, the expected number of transactions (per minute/hour/day), peak usage times, etc.

- Constraints and assumptions

  Identifies the constraints and assumptions to be satisfied in order to achieve the functionality.

- UML diagrams

  Is the core section where the use case specifications are really enriched with a lot of information that forms the Use-Case Model. It contains any of the appropriate diagrams described in Section 2.1.2, including the use cases or part of them from which the functionality is extracted.

- Grid exploitation

  Explains the Grid aspects related with the functionality. It states if the functionality is suitable for an exploitation that will be based on batch system or if it requires human-driven interactive behaviour, or both; it also indicates motivations. Among the others, we have considered the following requirements as candidates for batch usage: data intensive or computing intensive tasks, potential tasks parallelization where each piece is independent from the execution of the others (DAG dependencies could be defined), real-time performances.

- Mapping between functionalities and DILIGENT services (system integration)

  Contains some considerations on design issues – even if the functional specification should not describe how the system has to be realized – in order to ensure that a realistic system is specified. Design considerations are facilitated by the fact that this document is already partitioned in logical areas that are directly related to the technical work-packages. Moreover this section allows to indicate which DILIGENT service will offer the described functionality any time this information is necessary but not immediately derivable from the functionality description.

- Use Stories

  Includes meaningful example stories to clarify the functionality.

- Testing issues

  Provides some information about the testing of the functionality: how this functionality could be tested? How to specify test-cases in a very simple fashion? This section allows to include functional and acceptance testing as well as any other kind of testing that can help the activity scheduled in the WP1.7 (e.g. compatibility testing, performance testing, stress testing, integration testing, regression testing, structural testing etc.).

- Related non functional requirements

  Lists a number of non-functional requirements. Typical examples are constraints on various attributes of the functionality such as usability, reliability, interoperability, scalability, and security. In particular, security issues (regarding both users and content management) have been reported any time this information allows the usability of this document be improved.

2.2 Requirements alignment

As already explained, the Technical Annex I - "Description of Work" and the two deliverables "D2.1.1 ARTE Scenario Requirements Report" and "D2.2.1 ImpECt Scenario Requirements Report" have been the main sources of information used to derive the DILIGENT functional specification. In particular, each of them presents the requirements as expressed by the corresponding community. Given the different background and objectives of the two communities, these requirements were originally not aligned. A preliminary task of the analysis phase was therefore aimed at classifying these requirements and appropriately aligning them.

Figure 4 and Figure 5 show, respectively, a logical decomposition of the ImpECt and of the ARTE requirements. Each box in the figure corresponds to a use-case described in the deliverable of the community.

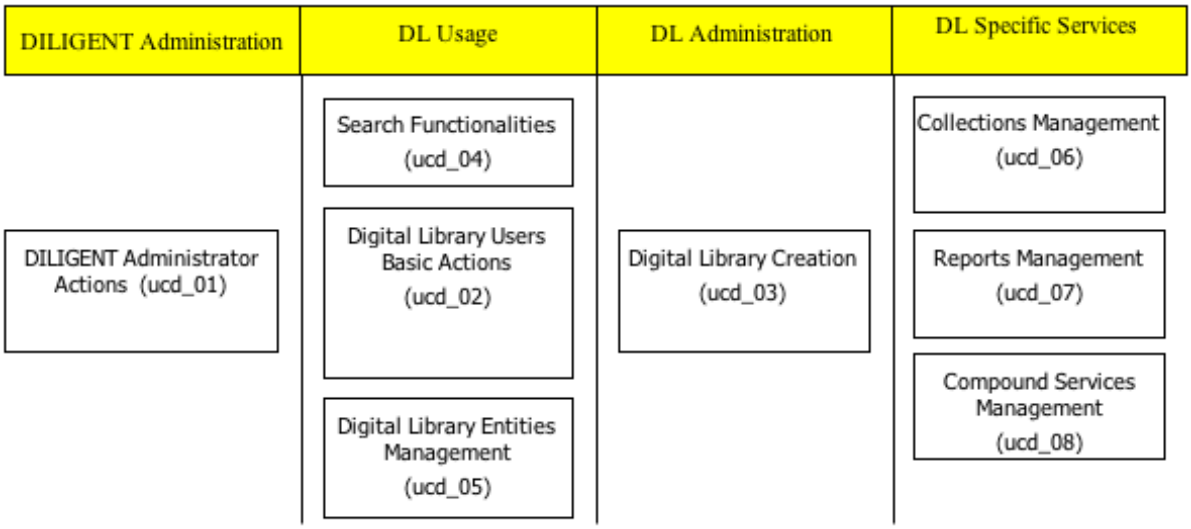

Figure 4: Decomposition of the ImpECt requirements

In particular, the "DL Specific Services" area of the ImpECt user community includes the automatically reports generation and the management of compound services in the context of a user story that sounds as follow: "The creation of a digital library produces the activation of its Web portal (that represents the workspace for each community user) and the initial set of resources is identified. Through their workspaces the users can access and use the DL resources. To be able to access a DL the users have to be invited while the resources are imported or registered in it: every Diligent-compliant resource (i.e. described by Diligent-compliant metadata, recognisable by Diligent) is 'imported' in the DL, while any external information source (e.g. retrieved on the Web) can be 'registered' in the DL, becoming DILIGENT compliant. The objects that populate a DL can be further categorised in simple objects (e.g. html, xls, doc, pdf, jpg files) or complex objects (e.g. reports, collections, compound services (CS), news)"[1].

[1] Information extracted from the D2.2.1 report

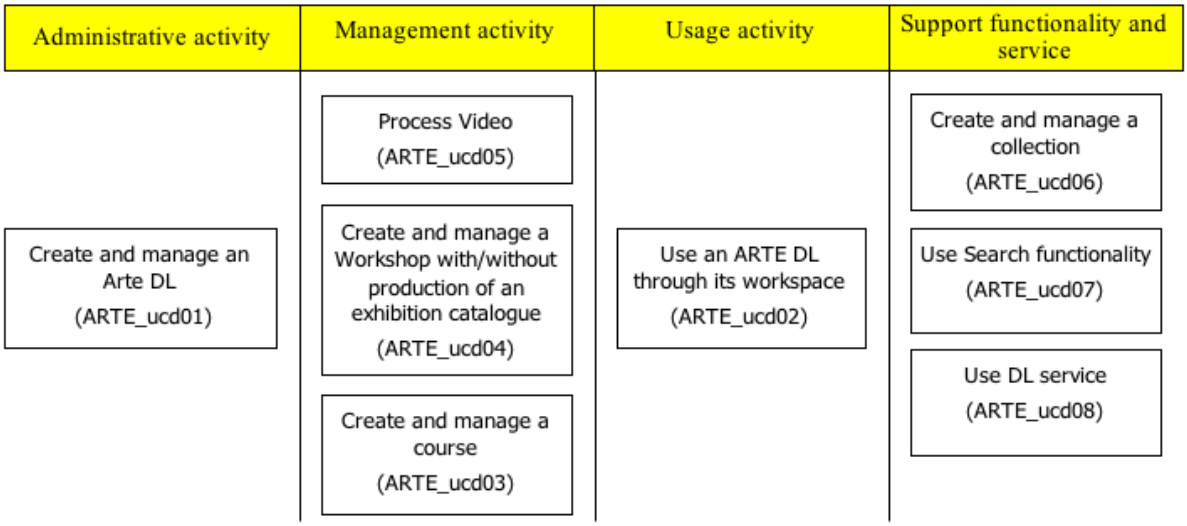

Figure 5: Decomposition of the ARTE requirements

For the ARTE user community, instead, it was important to stress specific functionalities capable to improve the collaboration among researchers as can be observed by their management activity: "The creation of an ARTE DL produces the activation of its DL workspace. Using this workspace, as reported in the second diagram, users can access and use the DL. The third, forth, and fifth diagrams describe the activities that authorized users, the ARTE members, can perform using the DL workspace. With their activities, the DL becomes live and supports courses, workshops, exhibition catalogues, and video process management. All the above mentioned diagrams use the packages included in the remaining three diagrams in order to perform their activity"[2].

Starting from these decompositions and by analysing the full list of use cases produced by the two scenarios, similar requirements have been aligned. Then, the aligned requirements have been partitioned in logical areas directly related to the technical work-packages.

The following tables report the results of this work.

DL Creation and Management	
ARTE	**ImpECt**
	DL Management
Create an ARTE DL	Create a DL
Update an ARTE DL	Update DL
Remove an ARTE DL	Dispose DL
Define an ARTE DL	
Redefine an ARTE DL	
Dispose an ARTE DL	
	Access Configuration
DILIGENT Resource Management	Resources Management
ARTE DILIGENT Management	
Add a Service	

[2] Information extracted from the D2.1.1 report.

Remove a Service	
Add an Archive	
Remove an Archive	
	Import
	Remove
	Delete
Select Services	
	Services Configuration
	Monitor Diligent Resources
User Management	Users Management
Add user	Create User
Remove user	Remove User
Edit user properties	Change User Details
Browse users	
Search Users	Search Users
Search by User properties	
Search by User Category	
Propose Services to be Added to / Removed from DILIGENT	
Propose Archives to be Added to / Removed from DILIGENT	
Browse Archives and Services proposals	
	Register

Content and Metadata Management	
ARTE	**ImpECt**
Collection Management	Collections Management
Update a Collection	Update Collection
Create a Collection	Create Collection
Remove a Collection	Remove Collection
Define Membership Criteria	
	Automatic Update
	Manual Update
Access Objects	
Save Object	
Remove Object	
	Select Object
	Inner Browsing
Personalize Object Views	

Navigate objects	Navigate
Play Video / Audio	
Manage Workspace	Use Workspace
Manage an ARTE DL workspace	
Manage Student Workspace	
Annotation Management	Annotation Management
Create an Annotation	Create Annotation
Read an Annotation	
Remove an Annotation	Remove Annotation
Update an Annotation	Update Annotation
Metadata Generation	Metadata Generation

Index and Search Management	
ARTE	**ImpECt**
Search and Retrieve Objects	Search
Search objects by Image	
Search part-of objects by Image	
Search objects by Video	
Search objects by Full Text	
Search video scenes by Keywords	
Search objects by Tone	
Search objects by Metadata	
Relevance Feedback	Interactive Information Discovery
	Define Context
Browse objects	
	Search/Browse Diligent
	Browse
	Explore
	Retrieve Related Resources
Process Video	
Index Management	Index Management
Search archives	
Search archives by Image	
Browse archives	
Search archives by ID	
Search archives by Metadata	
Resolve Image into parts	
Process Image	
Process Text	

Speech to text	
Thesaurus Generation	
Translation Service	

Process Management	
ARTE	**ImpECt**
	CSs Management
	Create CS
	Run CS
	Design CS
	Monitor CS
	Abort CS
	Remove CS
	Update CS

User Community Specific Application	
ARTE	**ImpECt**
Workshop Management	
Course Management	
Make an Exhibition Catalogue	
	LogIn
	LogOut
Edit an ARTE DL web portal properties	Personalise Portal
	Load Configuration
	Visualise Statistics
	Reports Management
	Create Report
	Document Model Definition
	Select Document Model
	Update Report
	Build Report
	Visualise Versions
	Compare Versions
	Remove Report

The technological partners have used the tables above as common starting point for the functional analysis.

3 BASIC CONCEPTS

This section briefly describes some basic concepts underlying the entire specification. All the partners have agreed the semantics of these concepts and the description given here represents a first contribution to the construction of a common glossary that will be shared among all the projects participants during all the phases of the project.

3.1 Resources

A resource is the basic component of a DL handled by the DILIGENT system. Examples of resources are: services, pieces of software with self contained procedures, persistent archives accessible via service public interfaces, Computing and Storage Element, Compound Service specifications, collections of objects. Handling of resources comprises gathering, storage, monitoring, and dissemination.

In particular, a resource is anything whose related information must be gathered, stored, monitored, and disseminated in order to provide the valuable amount of knowledge needed during the creation and management of a DL.

When the resources owner registers a resource, he/she will also provide a description of it. The services that will implement the Resource Management functionality will complete this description by automatically gathering additional information, if possible. The full description will be stored in a XML format and will be maintained both as information handled by the services of the Collective Layer and as a special materialized collection managed by the services of Content Management. Storing these resource descriptions in a special collection will permit to extend the basic discovery capabilities provided by the Resource Management (mainly the browse and search functionality based on exact matching procedures) with the advanced search functionalities that will be supported through the standard Search capabilities.

The relations among resources are depicted in the Figure 6.

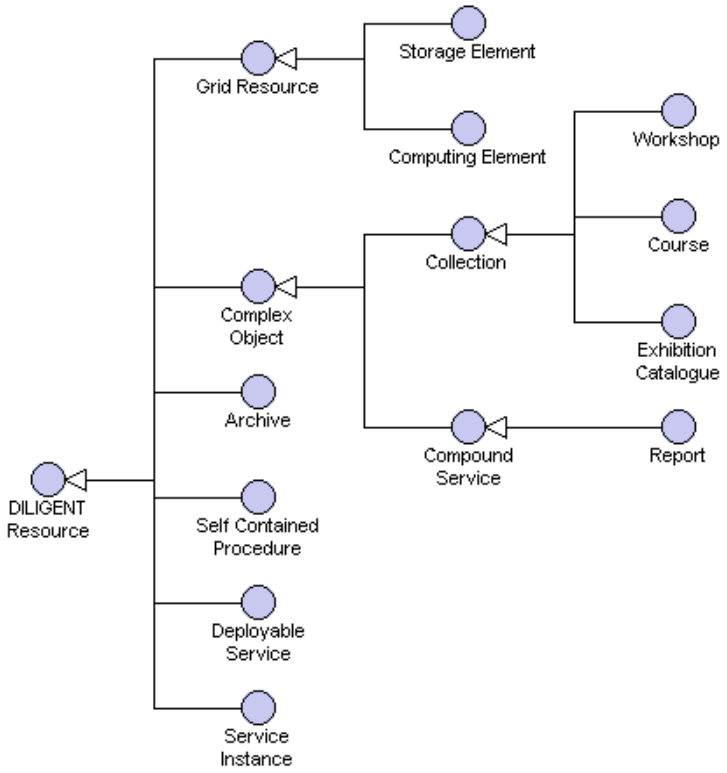

Figure 6: DILIGENT Resources

3.2 Actors Hierarchy

An actor is anything (human or not) that is able to interact with the system in some way.

The actors that access the DILIGENT system may have one or more roles. A role grants the actor with some specific rights (those assigned by VO Managers to that role). Roles are hierarchically related, where the specialization relationship means that the specialised role has all the rights of the parent and a number of other specific ones. The actors in the DILIGENT system are depicted in Figure 7. The role hierarchy reflects the actor hierarchy; during the DILIGENT lifetime it could be possible to have more roles in order to better model different authorizations granularities.

A detailed description of the rights of each role is provided when an associated functionality is presented.

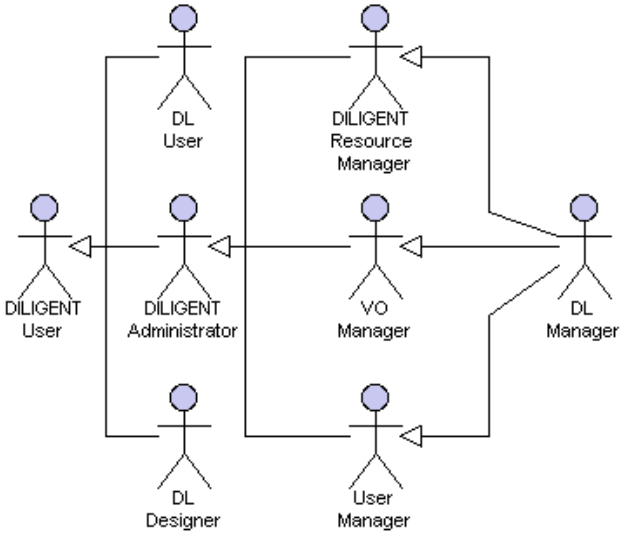

Figure 7: DILIGENT actors

Here, let us only emphasize that any user that accesses to the DILIGENT infrastructure must be a registered user, i.e. a user that owns a valid DILIGENT identity.

3.3 Metadata

Metadata are data about data. As a result of the analysis phase the following types of metadata have been identified:

- Descriptive Metadata: are associated with users, information objects and resources. They describe the corresponding element from an objective point of view. They can be given explicitly or they can be derived (automatically calculated).
- Annotation: descriptive and subjective (manually offered), focus on the semantic of the data. They are associated with any kind of information object in order to enrich its descriptive information capturing human remark on it.
- Features: derived and computing intensive. They are associated with any kind of information object in order to highlight and capture various characteristics of it. These data are usually used to populate an Index.
- Content Source Description: derived and computing intensive. They are associated with any kind of Content Source, e.g. Archives, Collections, and describe them from the content point of view.
- Index: derived, focus on efficient search. They are data structure associated with any kind of Content Source to speed up the discovery process.

All metadata are expressed in a XML format and will be maintained through the functionalities of the Metadata Management even if indexes could also be based on autonomous and ad-hoc storing mechanisms in order to improve the efficiency of the query processing.

3.4 Collection

A Collection is a set of digital objects. The DILIGENT system internally distinguishes two types of collections: v*irtual* and *materialized* collections.

A *virtual collection* is a collection built by specifying a membership condition, i.e. a condition that is verified by all and only the objects that belong to the collection. The membership to a virtual collection is established at access time, i.e. the objects belonging to the collection are identified at the time a user have access to the collection. The main consequence is that the collection content is highly dynamic, i.e. it is capable to follow the changes of the DILIGENT content. From an operational point of view, these collections will be automatically maintained making use of an appropriate workflow managed by the Process Management.

A *materialized collection* is a collection built by enumerating a pool of objects. As a consequence, materialized collections will be managed similarly to archives of digital objects, and therefore they will exploit the functionalities envisaged for the archive management (e.g. content source description, search, etc).

3.5 Workspace

A Workspace is particular kind of service that offers functionalities for supporting specific working activities. It allows downloaded or linked digital objects to be maintained, organized, annotated, removed, etc. A workspace can be personal or public. In the former case all the operations activated on the objects maintained in the workspace produce effects that are only visible to the owner of the workspace, while in the latter case these effects are visible to all the digital library users. The information space of a workspace is represented as a particular materialized collection. As such, it is managed by exploiting all the functions available for this type of collection.

3.6 Virtual Digital Library

A Virtual Digital Library (VDL) is a particular kind of Digital Library (DL), thus we need to explain this latter concept before to introduce the former.

A classical definition[3] states that "A digital library is an institution which performs and/or supports (at least) the functions of a library in the context of a distributed, networked collection of information objects in digital form. The functions meant are: selecting, collecting, preserving, organizing, representing, providing access to, ensuring knowledge of, and disseminating information objects, mediating and supporting interaction between information users and information objects".

From an operational point of view a DL is an entity that comprises a pool of integrated *Resources*, mainly *Archives* and *Services*, where authorized users are entitled to perform certain activities (those the *DL* have been built for) operating on the DL information space (the pool of archives or a subset of them) via DL services.

A VDL is an entity that, making use of the VO mechanism to glue together users and resources in a trusted environment, aggregate in a virtual way a pool of resources to provide digital library functionalities to a well fixed set of users.

---

[3] N. J. Belkin. *Understanding and Supporting Multiple Information Seeking Behaviors in a Single Interface Framework*. In Proceeding of the Eighth DELOS Workshop: User Interfaces in Digital Libraries, pages 11-18. European Research Consortium for Informatics and Mathematics, 1999.

However it is important to notice that throughout this document, and in general in DILIGENT, we use the term "Digital Library" as a synonym of "Virtual Digital Library" because all the DLs the system will create are VDLs.

3.7 Virtual Organization

A Virtual Organizations (VO) is a dynamic pool of distributed resources shared by dynamic sets of users from one or more organizations.

Providers usually make resources available to other parties under certain usage rules. Users are allowed to use Resources under Resource Provider (RP) conditions and with the respect of a set of inter-VO access rules. It is worth noticing that Users are not only human beings, but also resources and services willing to use other resources.

The VO model is defined in terms of users being assigned to roles and permissions being assigned to roles. In this model, the definitions of "Role" and "Permission" are borrowed from the *Role Based Access Control* standard[4]:

- "A *Role* is a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role".
- "A *Permission* is an approval to perform an operation on one or more objects". Usage rights are modelled in DILIGENT as associations between roles, actions and resources.

Furthermore, roles are organized in hierarchies allowing a natural way to capture organizational lines of authority and responsibility. Role hierarchies are not constrained to be trees; each role can have several ancestors with the only constraint that cycles are not allowed in the structure.

The concepts of User, Role and Permission and the relations between them, along with other VO entities that will be introduced later in this section are presented in Figure 8.

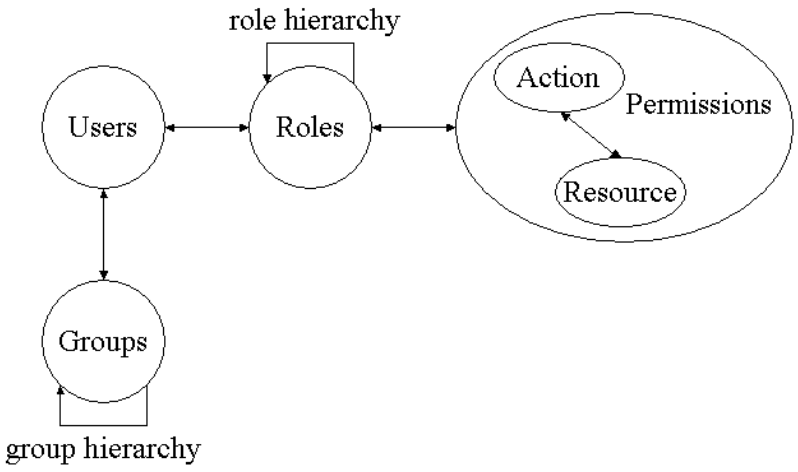

Figure 8: Relations between entities within a Virtual Organization

In order to put similar users together or maintain information related to many users (e.g. a contact person), the concept of *group* is also introduced. Figure 8 shows no relation

[4] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R.Chandramouli, "Proposed NIST standard for role-based access control", ACM Transaction on Information and Systems Security, vol. 4, no. 3, pp. 224-274, August 2001. http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf

between groups and either permissions or roles, meaning that a group is not meant to grant any right to users. The concept of *group* and that of *role* have to be considered orthogonal.

An important aspect in the operation of Virtual Communities is the definition of resource access policies by the Resource Provider and the fulfilment of those policies by members of the virtual community. Clearly we must address this feature taking into account the scalability of the system: resource and VO administration costs shouldn't increase with the number of users in the VO and the number of registered resources, respectively.

As a solution, the policy agreement between the Resource Provider and the Virtual Organization is first enforced by a course-grained access policy on the resource with respect to the Virtual Organization; then, a fine-grained access control is maintained within Virtual Organization which delegates access rights to the users. Although the Resource Provider is the ultimate authority for granting access to its resource, the trust contract between parties imposes a fair use of the resource. For this reason, it's highly important that the VO enforces the resource access policies on its own. This need is taken into account by modelling Resource Sharing Rules within the VO itself.

The need for decoupling sharing-rules and permissions is twofold:

- Sharing rules reflect, inside the VO, the trust contract between the Resource Provider and the Virtual Organization in order that the contract between them be respected. Usually, such a contract is expressed in terms of the whole community, without any explicit reference to community members.
- Registration of a resource should not rely on the existence of a specific actor authorized to use it. Modelling sharing rules with role-permission association would force the VO Manager to create unneeded roles and give them unneeded permissions (e.g. it might be unknown, at the registration time, what roles will have access to the resource). Furthermore who's registering the resource might not have the right to grant permissions to roles.

In the following picture, our first VO model is enriched with Sharing Rules. It can be noticed that there is a relation between Permissions and Sharing Rules modelling the fact that the former must be set in a consistent way with the latter.

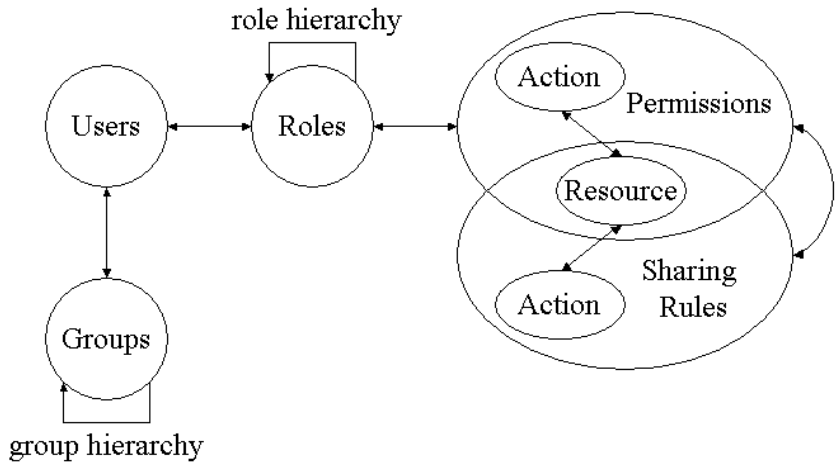

Figure 9: Virtual Organization Structure enriched with Sharing Rules

The need for sub-Virtual Organizations

Both ARTE and ImpECt scenario requirements state the need for a mechanism enabling a restricted set of the user community to focus its work on a particular set of resources using

specific services in order to fulfil a well-defined goal. Further, it's often the case that the lifetime of these very specific VO is limited to the accomplishment of the VO task.

In the ARTE scenario, "the organization of the material to be used by course students or to be exhibited/discussed in a workshop" is an example of goals of short-lived VOs. In the ImpECt scenario, "a particular group of users could be interested in working on data related to the Tyrrhenian Sea, to oil spill pollution and in the time span between year 2000 and 2002" is another example.

A deep analysis of these scenarios reveals the whole VO characterization given in the previous section. Furthermore, several considerations can be done on the nature of these specific user communities with respect to the larger VO:

- Users involved in the accomplishment of specific tasks are members of the VO too
- Resources needed for the fulfilment of the task have some visibility within the VO
- Users aren't expected to have fewer privileges on resources than those granted to them in the VO, and no more than those granted to the VO by the resource provider.
- The specific user community is expected to maintain the same level of trust as the VO does.

These considerations lead to a well-defined relation between a VO and its sub-VOs: given a VO *v* and one of its sub-VOs *s*, the following relations hold:

- The set of resources in *s* is a subset of resources in *v*
- The set of users in *s* is a subset of users in *v*
- The sub-VOs inherits roles, role hierarchy and role-permission associations from *v*.
- In *s*, new roles can be added to the hierarchy as descendants of existing roles (i.e. more privileged roles can be created) and new permissions can be granted to new nodes, but the set of permission granted to inherited nodes can't be changed.
- The sub-VO *s* inherits user-role associations from *v*. Users can be given further roles in the context of the sub-VO.
- Sharing rules for a resource in *v* still hold for that resource in the context of *s*, reflecting the fact that, from an external point of view, a sub-VO is still part of a trusted VO.

Figure 10 puts together all the concepts introduced till now.

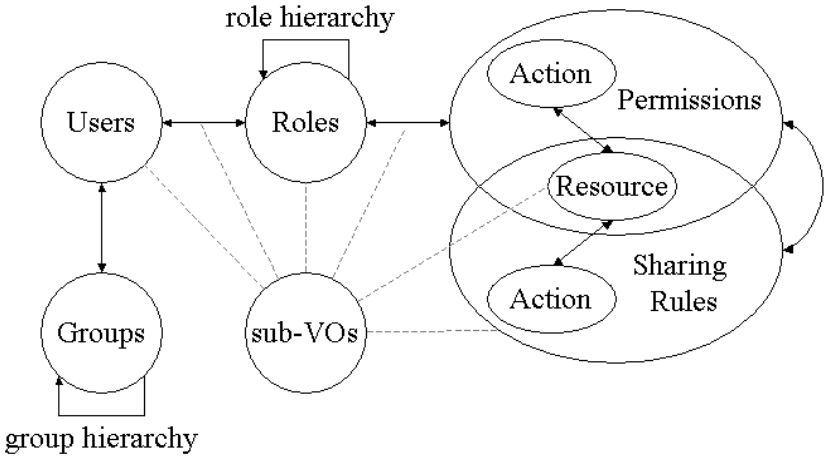

Figure 10: VO structure with sub-VO

4 DL CREATION AND MANAGEMENT

4.1 Introduction

This specification has been partitioned in five main areas that group the functionalities of Digital Library Creation and Management. These areas are:

- DLs Management

  It groups functionalities related to the definition, creation and management of new Virtual Digital Libraries. Both the ImpECt and ARTE requirements analysis reports underline the need to have these features in order to define their own DLs.

- VOs Management

  A VO is a dynamic collection of distributed resources shared by dynamic collection of users from one or more organizations. In DILIGENT there will be one global VO, named DILIGENT VO, and a number of nested VOs that represent at the first level the user communities, and at the second level the DLs of a specific user community. These functionalities provide mechanisms to build and support the above-mentioned scenario.

- Users Management

  Users Management includes functionalities that allow to add new users to DILIGENT, to create and manage groups of users, to invite users to something (a DL, a Collection, a Group, etc.) and to search and browse existing users.

- Resources Management

  This area deals with the resources used to create and manage DLs and those that can be used from DILIGENT services instances (e.g. other services instances or pieces of software with self-contained tasks). These resources must be registered as DILIGENT resources (i.e. registered in the DILIGENT VO) and the owner must provide a resource profile (e.g. description, copyright, data about the provider, etc.) and set the sharing policies (e.g. who can do what, on what, in what context, etc.) for each resource.

- Notification Management

  This area deals with the mechanisms the system offer to notify various DILIGENT users about certain action that have been executed or must be execute by the actor itself.

4.2 DLs Management

One of the main objectives of DILIGENT is the creation of Virtual Digital Libraries, i.e. transient DLs based on shared computational, multimedia and multi-type content and applications resources that can be dynamically created and moved on the underlying grid infrastructure thanks to the adoption of an advanced and innovative architectural model of DL. The *DLs Management* includes the functionalities that can be used to set up new DLs. In this context, main actors are the *DL Designer* and the *DL Manager*.

Due to its complexity, the top-level use-case diagram with the big picture of this area has been split into two figures, Figure 11 and Figure 12. In the first diagram the main actor is the *DL Designer* that is in charge to start the process of creation of new DLs. The actions that she/he can perform are primarily related with the DL definition (by selecting archives and services, and configuring the web portal of the DL). Then she/he can interact with the DL Manager in order to modify or dispose existing DLs (of which it is the owner); finally she/he can also make proposals to integrate new archives and/or service addressed to DILIGENT Resource Manager.

The second diagram in Figure 12 illustrates the management of DLs from the *DL Manager* point of view. This actor is responsible for the generation and the maintenance of DLs. Following the definition criteria specified by the DL Designer, DL Manager oversees the generation process of DL Resources and their allocation on the grid nodes. This process ends with the generation of the web portal of the rising digital library. This second group of functionalities has strong relationships both with the VOs Management and the Resources Management area. In fact, at VO level, a new DL is a new sub-VO that must be created at generation time (this is the reason why a DL Manager is also a VO Manager, see Figure 7: DILIGENT actors). Interactions with the resources area are required because DL Resources are basically DILIGENT Resources, so it is necessary to add them to the DL sub-VO and sets their sharing policies.

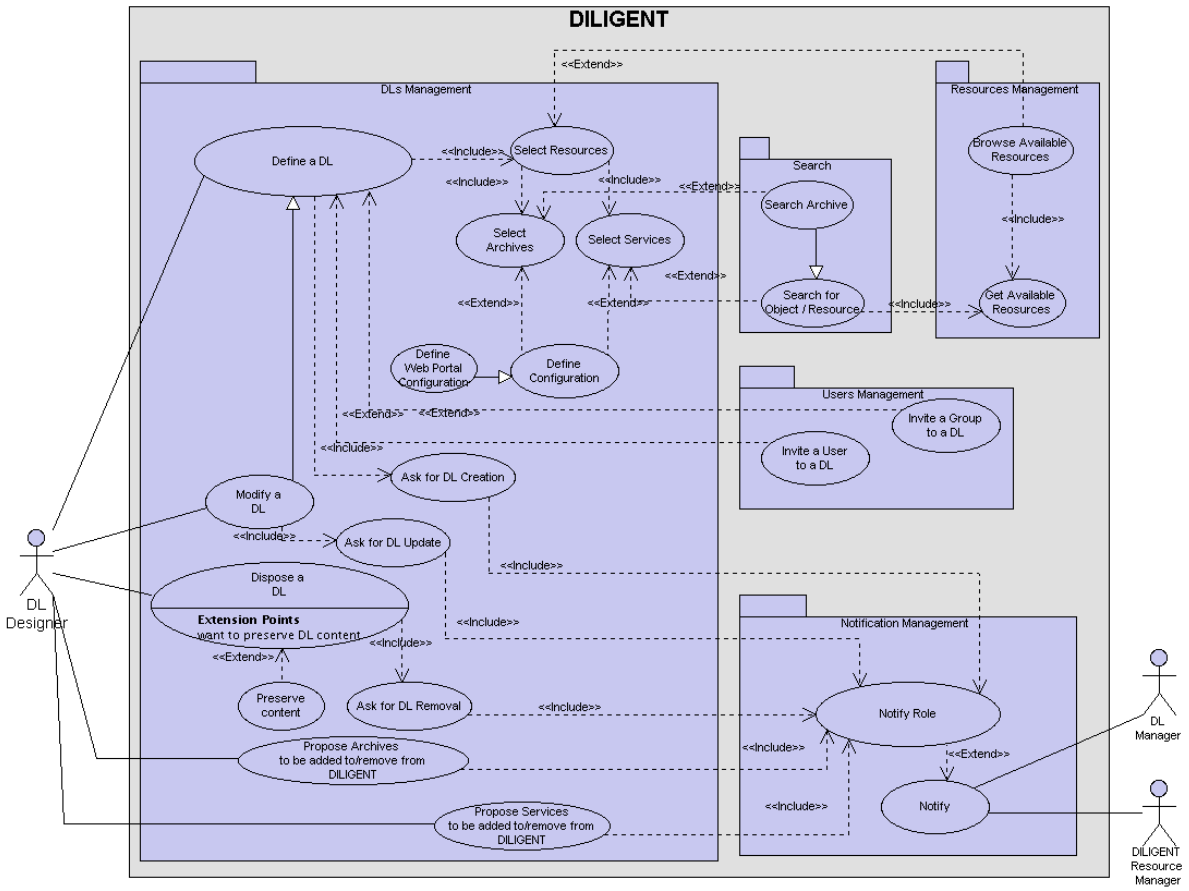

Figure 11: DL Management – DL Definition (use case diagram)

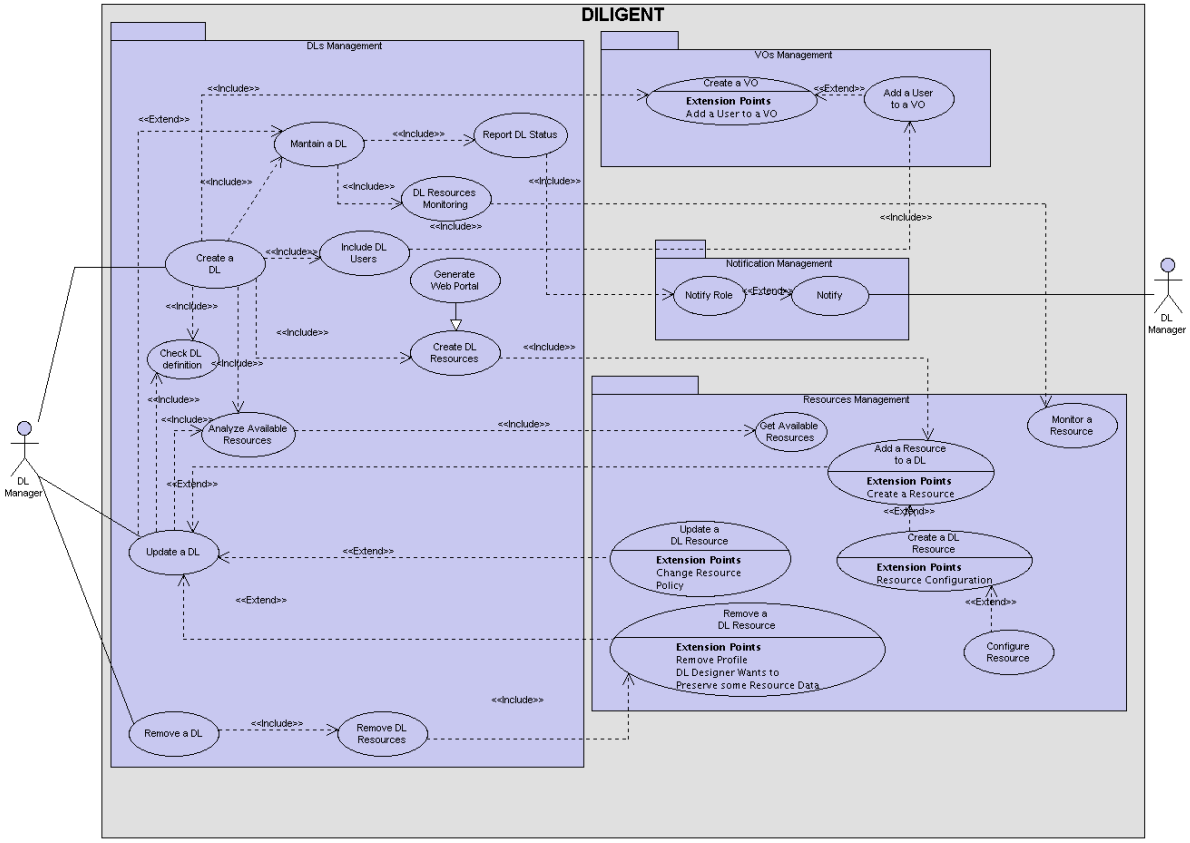

Figure 12: DL Management – DL Generation and Maintenance (use case diagram)

4.2.1 Define a DL

Description and priority

Defining a DL is the first step in the two-steps process that permits the creation and setting up of a new DL. The definition starts with an in-depth inspection of the resources that can be used according to the sharing rules defined by resources owners.

The entire process includes the following actions:

- Selection of Archives which define the knowledge to use in the DL
- Selection of available operations (different types of search and browsing operations, annotations, collection management, data generations, workflow management, etc.)
- Invitation of users and groups of users to access to the DL
- Configuration of the Web Portal used to access to the new DL
- Ask for DL creation to the *DL manager* actor

The result of these actions represents the *DL Definition Criteria*. This feature of the system has an high-priority since without it DLs can not be created.

User Requirements Fulfilled

This functionality fulfills a number of requirements that come into evidence from ARTE_ucd001 (*ARTE DILIGENT Management*, *Define an ARTE DL*) and ImpECt_ucd03 (*Create a DL - partially*).

Numbers

We do not expect that DL definition is a frequent action. It can be executed only by a restrict set of authorized users.

Constraints and Assumptions

It assumes that there exists a DILIGENT portal that permits the operations described above and an Information System from which is possible to extract information about available archives and services.

UML Diagrams

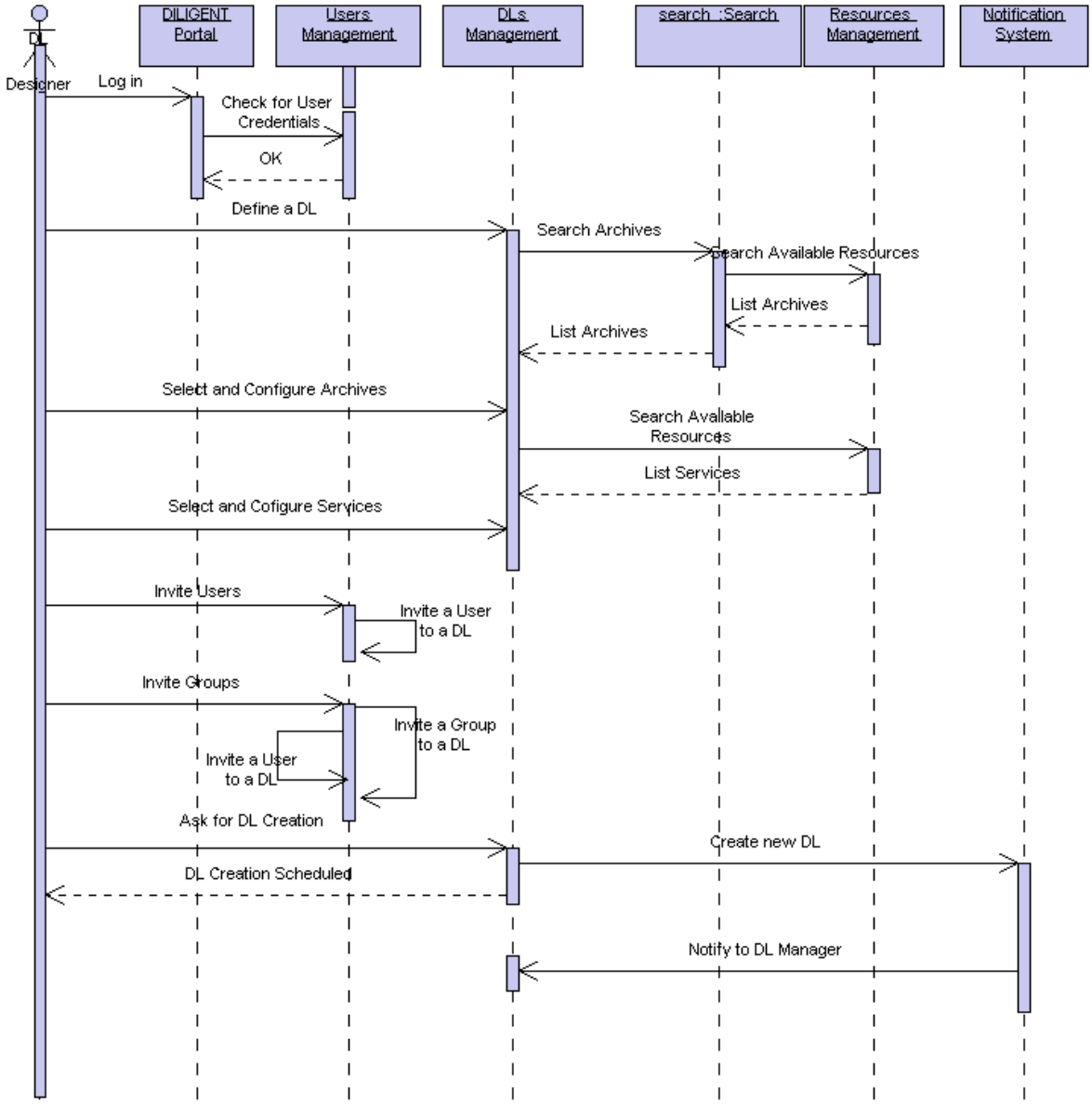

Figure 13: Define a DL (sequence diagram)

See also Figure 11.

Grid Exploitation

Grid usage comes into play only in the selection of archives and services due both discovering and Information System features of a Grid middleware are used to perform this operation.

Mapping between functionalities and DILIGENT services (system integration)

The VDL Generator Service with the support of the Information Service will implement this functionality. The DILIGENT portal provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

The ARTE project director and her team decide the definition criteria to create an ARTE DL, i.e. archives, resources and services which might be useful to support the activities to be held within the ARTE project (research, courses, workshops, expositions, etc.). Assuming that there exists an ARTE DILIGENT Portal, the following actions should be possible for defining any ARTE Digital Library:

1) SELECTING KNOWN ARCHIVES: The user wants to select one or more archives to insert them in the DL, whether to create it ex-novo or to update it. The user knows the archive ID or some elements of the archive description (Name, Publishing Institution, keywords, etc.) and uses search/browse operations to find the archive s/he wants to select. Then s/he looks at the archive description to verify if selection criteria regarding access/policy rights etc. are met, and if everything is OK, selects that archive for the ARTE DL s/he is creating. If in the archive description the user has seen a link to another archive of possible interest, then s/he accesses that archive, evaluates its content, policy rights, etc., and repeats the selection/inclusion operations.

2) SEARCHING ARCHIVES CONTAINING GIVEN IMAGES AND INCLUDE THESE ARCHIVES IN THE DIGITAL LIBRARY: After selecting well-known archives for inclusion in the ARTE DL, the user wants to enrich the ARTE DL with other possibly interesting archives. The user doesn't know any description element of the archives s/he is interested in, s/he knows instead which content s/he is interested in. For example, the user presumes that a given image contained in a literary text has later been used to illustrate texts of different kinds, for example texts of medicine or science history. S/he wants to verify such an hypothesis and see: i) whether archives different from the well-known ones contain that image; and, if so, ii) whether those archives are to be included in the ARTE DL. To this aim, the user searches the archives (all or a class of them, if DILIGENT can organize archives into subject classes) by submitting that image as a query. Then the user accesses each of the retrieved images and, if it is of interest, identifies the archive where it is contained in order to include that archive in the Digital Library.

3) SELECTING SERVICES: The people in charge for defining the DL want to select one or more services for equipping the DL. In order to do it they must be authorized to search among the DILIGENT available services and select those that will satisfy their needs, i.e. the needs of the users of the DL. The ARTE team judges the following services to be very useful: text and image searching, personalization of objects views, browsing of archive content or indexes, definition of virtual collections, annotation of documents, personalization of access rights, define user categories and associated rights.

4) SELECTING USERS: The ARTE Director or a Director Assistant wants also to define the users who will have access to the DL. In order to do it, the ARTE Director or a Director Assistant has to be able to i) search for potential users already registered inside DILIGENT and ii) to register new users to DILIGENT.

5) EDITING THE WEB PORTAL PROPERTIES: Finally, the ARTE Director or a Director Assistant wants to define the characteristics (e.g. the layout, the look&feel, etc.) of the Web portal that will be the DL access point.

Lastly, the form defining an ARTE DL is to be filled in at least with the following data:

- Name of the DL to be created
- Description

- Information about the issuing (or cooperating) institution(s)
- Information about access rules/rights

Testing issues

This functionality needs to be tested from a "semantic" point of view as well as from a "performance" point of view. The former testing issue is related with the correctness of the DL definition criteria with respect to the choices performed by the DL Designer. The latter one is related with the responsiveness of this functionality, i.e. the user must be able to define its DL using an online process that must take just the time to perform the choices without overhead due to acquire a picture of the system.

Related non functional requirements

The main non functional requirement is related with the usability of this functionality. It is mandatory to realize a user friendly process for defining a DL as the DL designer, that could not have technical skills, must put it effort on indicating the requirements s/he is interested in, avoiding details and other technicalities that will take her/him attention off its goal.

4.2.2 Select Archives

Description and priority

This use case represents one of the steps the DL designer must perform in order to define the DL definition criteria, i.e. the selection of the Archives that will constitute the knowledge to use in the DL. Moreover, if an archive allows to identify the parts of its content to use, it will be possible to specify the archive content that will be part of the DL using the configuration functionality (see 4.2.4). This feature has an high-priority since without the DLs definition process can not be completed.

User Requirements Fulfilled

This functionality fulfils a number of requirements that come into evidence from ARTE_ucd01 (*Define an ARTE DL, Select archives*) and ImpECt_ucd03 (*Create a DL – partially*).

Numbers

As this functionality is a part of the DL definition process and we do not expect that this activity will be frequent, the select archives action will take place not so often and moreover will be executed only by a restrict set of authorized users.

Constraints and Assumptions

It assumes that there exists a DILIGENT portal that permits the operations described above and an Information System from which is possible to extract information about available archives.

UML Diagrams

See Figure 13: Define a DL.

Grid Exploitation

Selection of archives asks for description and discovering features. A grid infrastructure will allow these operations via the information system.

Mapping between functionalities and DILIGENT services (system integration)

The VDL Generator Service with the support of the Information Service will implement this functionality. The DILIGENT portal provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

See 4.2.1 Define a DL.

Testing issues

Mainly related with semantic and performance. See 4.2.1 Define a DL.

Related non functional requirements

Usability is the main issue as reported in 4.2.1 Define a DL.

4.2.3 Select Services

Description and priority

This use case represents one of the steps the DL designer must perform in order to define the DL definition criteria, i.e. the selection of available operations (different types of search and browsing operations, annotations, collection management, data generations, workflow management, etc.) the DL will be equipped with. Moreover, if these functionalities are configurable, it will be possible to customize the functionality that will be offered by the DL using the configuration functionality (see 4.2.4). This feature has an high-priority since without it the DLs definition process can not be completed.

User Requirements Fulfilled

This functionality fulfils a number of requirements that come into evidence from ARTE_ucd01 (*Define an ARTE DL, Select Services*) and ImpECt_ucd03 (*Create a DL – partially*).

Numbers

As this functionality is a part of the DL definition process and we do not expect that this activity will be frequent, the select services action will take place not so often and moreover will be executed only by a restrict set of authorized users.

Constraints and Assumptions

It assumes that there exists a DILIGENT portal that permits the operations described above and an Information System from which is possible to extract information about available services.

UML Diagrams

See Figure 13: Define a DL sequence diagram.

Grid Exploitation

Selection of services asks for description and discovering features. A grid infrastructure will allow these operations via the information system.

Mapping between functionalities and DILIGENT services (system integration)

The VDL Generator Service with the support of the Information Service will implement this functionality. The DILIGENT portal provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

See 4.2.1 Define a DL.

Testing issues

Mainly related with semantic and performance. See 4.2.1 Define a DL.

Related non functional requirements

Usability is the main issue as reported in 4.2.1 Define a DL.

4.2.4 Define Configuration

Description and priority

This use case represents the configuration action that is performed during the DL definition phase, in particular during the selection of services and archives the DL will be equipped with. The aspects and characteristics that can be personalized/customized are different and are usually related with the particular kind of resource the designer is configuring. For instance, the configuration aspects of an archive can be related to a) the parts of its content to make accessible (e.g. all the PDF objects, all the object belonging to a Set or a collection, etc.), b) the format of data exposed, c) the action allowed on data exposed (e.g. an object is completely accessible or just an excerpt is available for free), etc. On the other hand, configure a service may deal with a) the kind of data the service is able to "process" (e.g. a feature extraction tool for image is able to manage MPEG-7 images), b) the performance level the service must offer (e.g. the DL must be equipped with a search service capable to reply to hundreds of users concurrently), c) the data (information sources) the service must act on (e.g. the DL is equipped with a classifier service, i.e. a service capable to automatically classify objects according to a classification schema, and the designer must be able to specify the repository off objects this service must be capable to manage), etc. Finally, notice that if a resource is not customizable at all, this action is not allowed.

User Requirements Fulfilled

This functionality fulfils the ImpECt_ucd03 Services Configuration UC as well as an hidden functionality of the ARTE_ucd01 Define an ARTE DL UC.

Numbers

As this functionality is a part of the DL definition process and we do not expect that this activity will be frequent, the configuration action will take place not so often and moreover will be executed only by a restrict set of authorized users.

Constraints and Assumptions

The main assumption underlying this functionality is related with the customizability of the services and archives, i.e. it will be possible to configure a service or an archive if and only if that archive or service have been designed to be configured. Moreover, the configuration parameters are resource specific and must be discoverable, e.g. published within the service/archive description data.

UML Diagrams

See Figure 13: Define a DL sequence diagram.

Grid Exploitation

There are no particular aspects related with the use of the Grid except some configuration parameters, e.g. if a service is configurable with respect to the number of parallel processes that can create/use.

Mapping between functionalities and DILIGENT services (system integration)

Even if this functionality is service/archive centric we figure out that it will be realized via the VDL Generator Service with the support of the Information Service and of the DILIGENT portal. This latter component provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

See 4.2.1 Define a DL.

Testing issues

Mainly related with semantic and performance. See 4.2.1 Define a DL.

Related non functional requirements

Usability is the main issue as reported in 4.2.1 Define a DL.

4.2.5 Define Web Portal Configuration

Description and priority

This use case represents the definition of the Portal configuration that is performed during DL definition phase. The aspects and characteristics that can be personalized/customized are clearly dependant by the configuration aspects offered by the DILIGENT Portal service. Some of these aspects are related with the layout the portal must presents, e.g. the colours to use, the fonts, the title, the logo and its placement, the services and its placement, etc. The latter point is particularly interesting as we are figuring out that each service that needs to interact with users via a GUI is responsible to supply it using a technology (e.g. Portlet) that can be easily integrated into the portal engine.

User Requirements Fulfilled

This functionality fulfils the ImpECt_ucd03 'Services Configuration' and, partially, 'Access Configurations' UCs as well as the ARTE_ucd01 'Edit an ARTE DL web portal properties' UC.

Numbers

See 4.2.4 Define Configuration.

Constraints and Assumptions

The main assumption underlying this functionality is related with the customizability of the Portal.

UML Diagrams

See Figure 13: Define a DL sequence diagram.

Grid Exploitation

There are no particular aspects related with the use of the Grid.

Mapping between functionalities and DILIGENT services (system integration)

This functionality will be fully covered by the DILIGENT portal that provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

See 4.2.1 Define a DL.

Testing issues

See 4.2.4 Define Configuration.

Related non functional requirements

Usability is the main issue as reported in 4.2.1 Define a DL.

4.2.6 Ask for DL Creation

Description and priority

This functionality models the action performed by the DILIGENT system at the end of the DL definition phase in order to notify the *DL Manager* about the new DL the *DL Designer* has asked to create. The notification action can be performed using the push modality, i.e. the *DL Manager* receives a notify message (e.g. an email), or the pull modality, the request is stored on the system and the *DL Manager* asks to the systems for pending requests.

User Requirements Fulfilled

This functionality fulfils, in part, the ImpECt_ucd03 'Create a DL' and the ARTE_ucd01 'Define an ARTE DL' UCs.

Numbers

This functionality is involved in a macro functionality that is quite rare in time, i.e. the definition of a DL; as a consequence this functionality is executed with scarce frequency.

Constraints and Assumptions

This functionality asks for a notification mechanism, i.e. a mechanism allowing the system to inform the DL Manager about a pending request. There are no particular requests for the notification model, i.e. the push model as well as the pull model can be realized and supported.

UML Diagrams

See Figure 13: Define a DL sequence diagram.

Grid Exploitation

This functionality has no particular issues in a Grid environment.

Mapping between functionalities and DILIGENT services (system integration)

The VDL Generator Service, with the support of the DILIGENT portal, is the service in charge for realizing this task.

Use Stories

See 4.2.1 Define a DL.

Testing issues

There are no particular issues in testing this functionality.

Related non functional requirements

Security aspect related with the notification mechanism is mandatory in order to prevent improper actions.

4.2.7 Modify a DL

Description and priority

This use case models the functionality that allows the *DL Designer* to modify the DL definition criteria of an already defined DL. As it is a specialization of the Define a DL use case, the steps to perform are the same, i.e.:

- Inspection of the resources that can be used according to the sharing rules defined by resources owners
- Selection of Archives which define the knowledge to use in the DL
- Selection of available operations (different types of search and browsing operations, annotations, collection management, data generations, workflow management, etc.)
- Invitation of users to access to the DL
- Configuration of the Web Portal used to access to the DL

All these actions have to be intended also as operations performed on a pre-existing definition, e.g. selection of archives means also de-selection of archives previously identified as useful, etc. However, the result of these actions represents the new *DL Definition Criteria*. This phase ends with the request for a DL update operation performed to the *DL Manager* actor.

User Requirements Fulfilled

This functionality fulfils the ARTE_ucd01 'Redefine an ARTE DL' and the ImpECt_ucd03 'Update a DL', in part.

Numbers

As reported by the two user communities the operation of updating of a DL is executed several times per month.

Constraints and Assumptions

This functionality assumes that the DILIGENT portals enables authorized users to perform the steps described above as well as that the Information Service supplies to it the needed information.

UML Diagrams

The logical flow of this task is the same as that for the DL definition presented in Figure 11: DL Management – DL Definition except that all the operations are executed having a sort of 'background knowledge', e.g. when the DL Designer select the archives to be included into the DL she/he find the same archives already selected.

Grid Exploitation

The use of grid technologies comes into play in the selection of archives and services. In particular grid technologies and mechanisms will be used during the resources discovering phase based on the resource descriptions that are made available. This discovering phase must also take into account the limitations due to the resource sharing rules.

Mapping between functionalities and DILIGENT services (system integration)

The VDL Generator Service with the support of the Information Service will implement this functionality. The DILIGENT portal provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

The ARTE Director and/or a Director assistant decides how to update a given ARTE DL definition, e.g. when new archives/services/users must be included, or which existing ones must be removed. Using the DILIGENT Portal she/he will compile this list and then stored it into the system in order to allow the ARTE Administrator to effectively update the given ARTE DL.

Testing issues

This functionality needs to be tested from a "semantic" point of view as well as from a "performance" point of view. The former testing issue is related with the correctness of the DL definition criteria with respect to the choices performed by the DL Designer. The latter one is related with the responsiveness of this functionality, i.e. the user must be able to redefine its DL using an on line process that must take just the time to perform the choices without overhead due to acquire a picture of the system.

Related non functional requirements

The main non functional requirement is related with the usability of this functionality. It is mandatory to realize a user friendly process for redefining a DL as the DL designer, that could not have technical skills, must put it effort on indicating the requirements s/he is interested in, avoiding details and other technicalities that will take her/him attention off its goal.

4.2.8 Ask for DL Update

Description and priority

This functionality models the action performed by the DILIGENT system at the end of the DL definition update phase in order to notify the *DL Manager* about the new characteristics the *DL Designer* have asked for a DL. The notification is made using the notification mechanisms described in Section 4.6.2.

User Requirements Fulfilled

This functionality fulfils, in part, the ARTE_ucd01 'Redefine an ARTE DL' and the ImpECt_ucd03 'Update a DL' UCs.

Numbers

This functionality is related with the 'Modify a DL' functionality and thus it will be executed several times per month.

Constraints and Assumptions

This functionality asks for a notification mechanisms, i.e. a mechanisms allowing the system to inform the DL Manager about a pending requests. There are no particular requests for the notification model, i.e. the push model as well as the pull model can be realized and supported.

UML Diagrams

See 4.2.7 Modify a DL - UML Diagrams.

Grid Exploitation

This functionality has no particular issues in a Grid environment.

Mapping between functionalities and DILIGENT services (system integration)

The VDL Generator Service, with the support of the DILIGENT portal, is the service in charge for realizing this task.

Use Stories

See 4.2.7 Modify a DL.

Testing issues

There are no particular issues in testing this functionality.

Related non functional requirements

Security aspect related with the notification mechanism is mandatory in order to prevent improper actions.

4.2.9 Dispose a DL

Description and priority

This functionality is related with the removal of a defined DL. Via this action the *DL designer* is enabled to ask for the removal of a DL (see 4.2.11) and is also enabled to specify the action to perform on the DL content in order to prevent its loss (see 4.2.10).

User Requirements Fulfilled

This functionality fulfils the ARTE_ucd01 'Dispose an ARTE DL' and, in part, the ImpECt_ucd03 'Dispose a DL' UCs.

Numbers

The requirements expressed by the two user communities conducted us to conclude that this task is executed rarely, usually at the end of the activity that bring up the DL.

Constraints and Assumptions

This functionality assumes that the DILIGENT portal enables authorized users to perform the steps described above in a user-friendly fashion. Particular attention must be posed on the process allowing the DL Designer to take a decision about the DL content to preserve.

Grid Exploitation

There are no particular aspects related with Grid technologies.

Mapping between functionalities and DILIGENT services (system integration)

The DILIGENT portal provides a user-friendly interface to perform the tasks in an intuitive way.

Use Stories

The DL designer selects the DL to remove and expresses the requirements about the preservation via the user-friendly mechanisms offered by the Portal. At the end of this phase a notification is sent to the DL Manager in order to physically remove the resources forming the DL and performing the tasks needed for respecting the preservation requirements expressed.

Testing issues

The testing issues related with this functionality are: (i) its correctness, i.e. the action notified to the DL manager must comply with those specified by the DL designer, and (ii) its performance, i.e. the time required to the DL Designer for expressing its needs and the time needed to notify the DL Manager.

Related non functional requirements

Security aspect related with the notification mechanism is mandatory in order to prevent improper actions.

4.2.10 Preserve content

Description and priority

This functionality regards the preservation of the DL content to perform before the DL disposal operation. This need is due to the fact that during the DL lifetime new objects could be created and they could be relevant for other community than those of the DL by the *DL designer*. The *DL designer may* ask to make them permanently available as a new DILIGENT resource. In this case a collection owned by DL Designer will be created.

User Requirements Fulfilled

This functionality fulfils, in part, the ARTE_ucd01 'Dispose an ARTE DL' and the ImpECt_ucd03 'Dispose a DL' UCs.

Numbers

This task belongs to a non-frequent functionality; as a consequence, it is executed few times.

Constraints and Assumptions

The main assumption underlying this functionality is related with the presence of a storage area within DILIGENT that allows the eventually specified preservation activities to take place. On the other hand, we can figure out the previous assumption as a constraint, i.e. this functionality will be available if and only if an available storage area exists.

UML Diagrams

No Diagram is needed.

Grid Exploitation

The only use of Grid technologies within this task is about the discovering and eventually reserving of a storage area where preserved content can be stored successfully.

Mapping between functionalities and DILIGENT services (system integration)

The DILIGENT portal provides a user-friendly interface to perform the tasks in an intuitive way. The Information Service allows the discovering of available storage resources.

Use Stories

See 4.2.9 Dispose a DL.

Testing issues

See 4.2.9 Dispose a DL.

Related non functional requirements

See 4.2.9 Dispose a DL.

4.2.11 Ask for DL Removal

Description and priority

This functionality models the action performed by the DILIGENT system at the end of the DL disposal phase in order to notify the *DL Manager* about the needs to physically remove an existing DL. The notification is made using the notification mechanisms described in Section 4.6.2

User Requirements Fulfilled

This functionality contributes to cover the ARTE_ucd01 'Dispose an ARTE DL' and the ImpECt_ucd03 'Dispose a DL' UCs.

Numbers

See the 4.2.9 Dispose a DL, i.e. the UC that invoke this task.

Constraints and Assumptions

This functionality asks for a notification mechanisms, i.e. a mechanisms allowing the system to inform the DL Manager about a pending requests. There are no particular requests for the notification model, i.e. the push model as well as the pull model can be realized and supported.

UML Diagrams

Due to the level of details adopted within this specification no further diagrams are needed.

Grid Exploitation

This functionality has no particular issues in a Grid environment.

Mapping between functionalities and DILIGENT services (system integration)

The DILIGENT portal is the service in charge for realizing this task.

Use Stories

See 4.2.9 Dispose a DL.

Testing issues

There are no particular issues in testing this functionality.

Related non functional requirements

Security aspect related with the notification mechanism is mandatory in order to prevent improper actions.

4.2.12 Propose Archives to be added to/removed from DILIGENT

Description and priority

The DILIGENT system will be able to manage just the pool of resources it knows. Via this functionality the DL designer notifies the Resource Manager, about a) new archives that are relevant for the community/communities it represents and that it plans to integrate into its DLs, i.e. the DLs s/he has defined/will define, and b) already registered archives that are not appropriate for community/communities it represents. Before to see and use the new archives or to remove the old ones, those must be added (4.3.1 Add a Resource to DILIGENT)/removed (4.3.8 Remove a Resource) to/from the DILIGENT resources.

User Requirements Fulfilled

This functionality contributes to cover the ARTE_ucd01 'Propose Archives to be added to/removed from DILIGENT' and the ImpECt_ucd03 'Import' UCs.

Numbers

Even if we hope that this task will be executed many times in order to enrich the DILIGENT infrastructure, we figure out that the pool of archives of interests for a community will be limited and will increase slower. As a consequence this task will not be executed with an high frequency.

Constraints and Assumptions

The main assumptions underlying this activity are related with (a) the networked status of the archive, i.e. it must be possible to have access to the archive content via a network connection, (b) the DILIGENT system capability, via wrapper and other similar technologies, to have access to the archive content.

UML Diagrams

As this functionality is just related with the gathering of information about the archives to import into DILIGENT, no diagrams are needed.

Grid Exploitation

The exploitation of Grid technologies is related with the kind of import mechanism that will be allowed by the archive and by DILIGENT as well. For example, if the archive open its content to the DILIGENT infrastructure it will be possible to figure out that the DILIGENT system gathers the content of the archive and uses the distributed storage capacity to maintain those documents. On the contrary, if the archive maintains the documents by itself, no exploitation of grid technologies will be possible.

Mapping between functionalities and DILIGENT services (system integration)

As this functionality is related with the gathering of archive descriptive information, the DILIGENT portal is the appropriate service for realizing this task.

Use Stories

An authorized user (the ARTE Director or a Director Assistant) proposes new archives to be included in or removed from the DILIGENT resources set by filling in the appropriate request form. For each archive, minimal data to be filled in are:

- Name of the proponent
- Archive name
- Archive type (web portal, data base, etc.)
- Archive address (e.g. if archive is accessible using Internet the URL)
- Archive access rights

After having filled the form, the request is notified to the appropriate manager.

Testing issues

There are no particular issues in testing this functionality. An important aspect to verify is the syntactical check of inserted data.

Related non functional requirements

Security aspect related with the notification mechanism is mandatory in order to prevent improper actions.

4.2.13 Propose Services to be added to/removed from DILIGENT

Description and priority

The DILIGENT system will be able to manage just the pool of resources it knows. Via this functionality the *DL designer* notifies the *Resources Manager*, about a) new services that are relevant for the community/communities it represents and that it plans to integrate into its DLs, i.e. the DLs s/he has defined/will define, and b) already registered services that are considered not appropriate for community/communities it represents. Before to see and use the new services or to remove the old ones, those must be added (4.3.1 Add a Resource to DILIGENT)/removed (4.3.8 Remove a Resource) to/from the DILIGENT resources.

User Requirements Fulfilled

This functionality contributes to cover the ARTE_ucd01 'Propose Services to be added to/removed from DILIGENT' and the ImpECt_ucd03 'Import' UCs.

Numbers

Even if we hope that this task will be executed many times in order to enrich the DILIGENT infrastructure, we figure out that the pool of services of interests for a community will be limited and will increase slower. As a consequence this task will not be executed with an high frequency.

Constraints and Assumptions

The main assumptions underlying this activity are related with (a) the networked status of the service, i.e. it must be possible to have access to the service via a network connection, (b) the DILIGENT system capability to have full control of the service. In fact, it will be possible to figure out two types of services, those that will be hosted by the Institution supplying it and those whose code is make available and will be hosted on DILIGENT infrastructure resources.

UML Diagrams

As this functionality is just related with the gathering of information about the services to import into DILIGENT, no diagrams are needed.

Grid Exploitation

The exploitation of Grid technologies is related with the kind of import mechanism that will be allowed by the service characteristics and by DILIGENT as well. For example, if DILIGENT will have full control on the service, i.e. its source code is make available to the system, the infrastructure will be enabled to use the distributed computing capacity to maintain the service instances it needs up and running with the desired QoS. On the contrary, if the Institution offering the service plans to host and maintain the service by itself, no (or partial) exploitation of grid technologies will be possible.

Mapping between functionalities and DILIGENT services (system integration)

As this functionality is related with the gathering of archive descriptive information, the DILIGENT portal is the appropriate service for realizing this task.

Use Stories

An authorized user (the ARTE Director or a Director Assistant) proposes new service to be included in or removed from the DILIGENT resources by filling in the appropriated request form. For each service/tool, minimal data to be filled in are:

- Name of the proponent
- Service/tool name
- Service type (web service, retrieval software, image processing software, etc.)
- Service/tool license (GNU, property, Open source, etc.)
- Service URI (the code or the instance address)

After having filled the form, the request is notified to the appropriate manager.

Testing issues

There are no particular issues in testing this functionality. An important aspect to verify is and the syntactical check of inserted data.

Related non functional requirements

Security aspect related with the notification mechanism is mandatory in order to prevent improper actions.

4.2.14 Create a DL

Description and priority

The second step of a DL creation is the real generation of the new DL as defined by the *DL designer*. This complex operation justifies the usage of a Grid infrastructure as foundation of the DILIGENT system. In fact, the new DL resources are created on the fly on the available grid nodes that matches both their software and hardware requirements.

Starting from the list of resources identified in the definition step, this procedure asks for the creation of a VO for the new DL, creates the necessary DL Resources (which means create new resources or share already existing ones), registers them in the VO (and also in the DILIGENT VO, if they are new), creates DL users (previously invited in the definition phase), and finally generates the DL portal.

All these actions and their sequence are documented in Figure 14: Create a DL.

User Requirements Fulfilled

This functionality fulfills the *Create an ARTE DL* use-case from ARTE_ucd001 and *Create a DL, Access Configuration, Populate DL, Register, Import* use-cases from ImpECt_ucd03.

Numbers

We do not expect that the DL creation is a frequent action. It can be executed only by a restrict set of authorized users.

Constraints and Assumptions

It assumes that there exists a DILIGENT portal that permits the operations described above. Moreover, there must exist an Information System (to extract information about available grid nodes and their configuration), a Broker Service (to implement matchmaking algorithms among services requirements and nodes equipment) and mechanisms for dynamic software deployment.

UML Diagrams

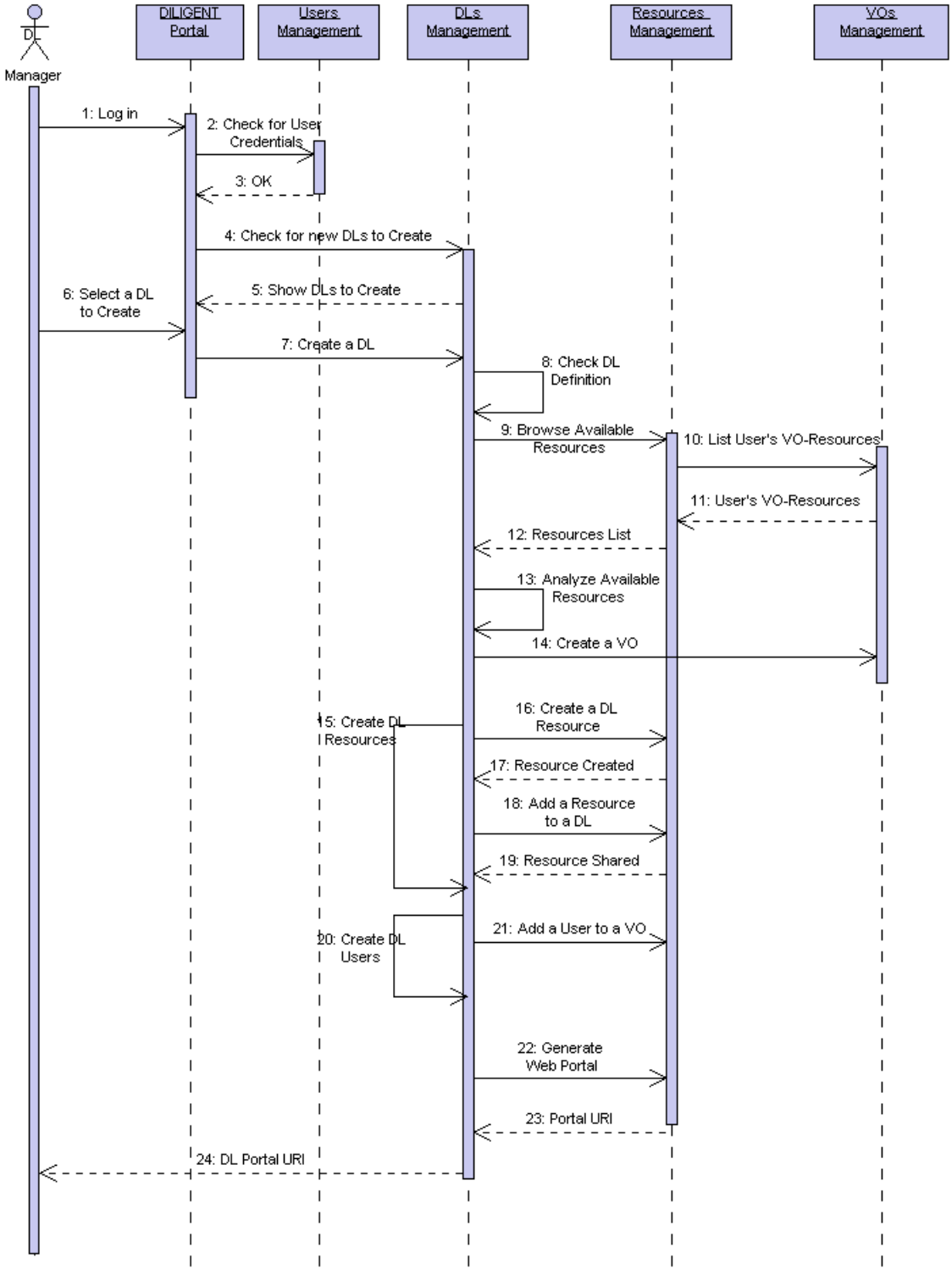

Figure 14: Create a DL (sequence diagram)

See also Figure 12: DL Management – DL Generation and Maintenance.

Grid Exploitation

During a DL generation, the potential Grid exploitation is very high. Within this functionality, the Grid infrastructure will be used as a single big computational environment where DL Resources can be dynamically located and moved in order to guarantee the Qualities of Services requested by *DL designers*.

Mapping between functionalities and DILIGENT services (system integration)

The Keeper Service is in charge for the creation and management of a DL. It first interacts with the Information Service in order to obtain a set of service instances that are able to satisfy the characteristics required by the DL specification criteria. In order to cover the un-satisfied requirements, it must also be able to build a complex list of mutually dependent services to submit to the Broker & MatchMaker (B&MM) Service. The B&MM responds with a list of priority-label Grid nodes for each service. Once the Keeper receives this list, it instances each service on the appropriate Grid node.

The B&MM should provide a parametric language for the incoming requests and a general, efficient and flexible matchmaking algorithm for the discovery of the optimal topology of a set of services. This activity is supported by the DILIGENT Information Service, that is capable to collect and process all the information needed for discovering the appropriate resources and nodes to use. The maintenance and update of these data is also assisted by the Dynamic VO Support component that notifies changes in the structure of the Virtual Organization (VO) to the Information Service.

Use Stories

The DL Manager creates a DL, in particular the resources forming it, according to the instructions given by the DL Designer during the design phase (see 4.2.1 Define a DL). Such instructions regard:

- Which name the DL is to be given;
- Which archives are to be included in the DL;
- Which services are to be associated; and
- Which are the users that will be entitled to access the DL.

At the end of this creation phase a DL, with its own portal, will be up and running and users entitled to have access will use the DL service.

During the DL lifetime the Manager will be notified about malfunctions of the DL, e.g. unavailable resources, low performance of services, etc. via an automatic notification mechanism. In this way it will be able to take action on the pool of resources forming the DL to ensure required QoS.

Testing issues

The test of this functionality is particular demanding and critic. The functionality we are speaking about is related with the instantiation of a distributed system in accordance with some definition criteria. First of all it must be identified an appropriate matching algorithm in order to verify that the created resources fulfils the expressed requirements. Secondly it must be identified and realized a sort of 'secure ping mechanisms' for testing the access to the services forming the DL in a controlled way.

Related non functional requirements

There are many non-functional requirements related with this task. The usability of the mechanisms offered to the DL manager in order to create the DL is an important issue. Security is another important issue, in particular only authorized users will use this functionality and the resources used to realize the DL must be those make available for that community. Interoperability and scalability are other issues related with the selection of resources to use.

4.2.15 Check DL definition

Description and priority

This functionality is related with the second phase of the DL creation action and represents the checking, executed by the *DL Manager*, of the DL definition criteria expressed by the *DL*

designer. This check operation consists in a set of formal controls about the consistency and compatibility of the expressed criteria. During this phase the DL Manager is able to modify the definition criteria in order to make them consistent and complete, e.g. the *DL designer* just express the requirements to execute queries on documents whose metadata are expressed in Dublin Core format, the *DL Manager* enrich this definition adding the requirements to have an index on those documents based on the Dublin Core metadata if those exists, otherwise s/he must also add the requirement to generate those metadata descriptions. Notice that these control must be done by the *DL Manager* because this actor have the technical skills to execute them, on the contrary the *DL designer* is a non technical actor that is able just to express its requirements.

User Requirements Fulfilled

This UC does not directly fulfil a user requirement, it has been derived to factorize out a commonality among the various DL creation functionalities.

Numbers

We do not expect that the DL creation is a frequent action. It can be executed only by a restrict set of authorized users. So also this checking operation is executed by the same set of authorized users.

Constraints and Assumptions

This functionality assumes that exists a formal specification language agreed among the DL designer and the DL Manager used to express the DL definition criteria. Moreover assumes that the resources descriptions contain all the data needed to make the compatibility checks needed.

UML Diagrams

At this level of detail, this functionality is considered as one step activity. No further diagrams are needed.

Grid Exploitation

The exploitation of Grid technology is related with the gathering of data about resources using the information system. In particular, this functionality needs also to acquire data about the computing and storage resources in order to check if exists enough resources to fulfil the user requirements expressed in terms of DL characteristics. For example, if to meet the DL definition criteria is needed a service that asks for 20GB of storage space and the infrastructure does not offer such kind of resource to the DL community then the DL cannot be instantiated.

Mapping between functionalities and DILIGENT services (system integration)

This functionality will be mainly supported by the Information Service that is in charge for supplying the information needed about the resources and by the VDL generator service that, via the DILIGENT Portal, will support the DL Manager to perform its task.

Use Stories

See 4.2.14 Create a DL.

Testing issues

In order to test this functionality diverse set of DL definition criteria will be supplied and matched against diverse pool of available resources in order to verify the correctness of the verification algorithm.

Related non functional requirements

Two of the main non functional requirements are the usability of this functionality, e.g. the DL Manager must be able to check the DL Definition criteria in a user friendly fashion, as

well as performance, i.e. the time required to perform the check test clearly depends by the complexity of the DL definition but must be a 'reasonable' time overhead in the process of DL creation.

4.2.16 Analyze Available Resources

Description and priority

This functionality is related with the second phase of the DL creation action and represents the analysis and selection of the physical resources that will be part of the DL. This operation is necessary in order to "translate" the definition requirements expressed by the *DL designer* into a pool of physical resources that will represent the DL, i.e. the *DL Manager* will identify the real archives, services and resources in general needed for fulfilling the DL definition criteria. In order to execute this selection operation the *DL manager* must be able to have a list of the available resources he can use.

User Requirements Fulfilled

This functionality does not arise directly from any user requirements but is obtained by factorizing out one of the steps involved in the DL physical creation.

Numbers

We do not expect that the DL creation is a frequent action. It can be executed only by a restrict set of authorized users. So also this checking operation is executed by the same set of authorized users.

Constraints and Assumptions

We assume that the resources descriptions contain all the data needed to make the analysis and selection of resources an easy task for the DL manager.

UML Diagrams

See Figure 14: Create a DL.

Grid Exploitation

The exploitation of Grid technology is related with the gathering of data about a distributed pool of resources using the information system.

Mapping between functionalities and DILIGENT services (system integration)

This functionality will be mainly supported by the Information Service that is in charge for supplying the information needed about the resources and by the Dynamic VO Support service that is in charge to identify the available resources, i.e. the pool of resources that DL users are entitled to see and use. Moreover, the DILIGENT Portal will support the DL Manager to perform its task.

Use Stories

See 4.2.14 Create a DL.

Testing issues

In order to test this functionality diverse pool of available resources associated with different Communities will be used to verify the correctness of the algorithm that will identify the appropriate resources.

Related non functional requirements

The usability of this functionality is the most relevant requirement. Usability means, in this case, that the process defined to perform this task must be as much intuitive and user-friendly as possible.

4.2.17 Include DL Users

Description and priority

This functionality is related with the second phase of the DL creation action and represents the inclusion of users that will be allowed to access and use the DL. (see the lower part of the diagram in Figure 14: Create a DL in Section 4.2.14).

User Requirements Fulfilled

This functionality contribute to fulfil the user requirements for DL creation, i.e. *Create an ARTE DL* from ARTE_ucd001 and *Create a DL, Access Configuration, Populate DL, Register,* and *Import* from ImpECt_ucd03.

Numbers

We do not expect that the DL creation is a frequent action. It can be executed only by a restrict set of authorized users. So also this operation is executed by the same set of authorized users.

Constraints and Assumptions

This functionality assumes that all the users of the DL are already DILIGENT Users, i.e. they have been already registered as system users. New users will be entitled to adhere to the DL via invitation mechanisms, covered by a set of functionalities offered by the User Management package, see Section 4.5.19 Invite a User.

UML Diagrams

See the lower part of the diagram in Figure 14: Create a DL in Section 4.2.14.

Grid Exploitation

Grid technologies already manage users, in particular deals with the management of their identities and credentials. This functionality will be built making use as much as possible of these already existing technologies. In particular, we figure out that DL resources will trust and accept the certificate associated with DL users, moreover the resources will be able to give the appropriate grant in terms of access and use to these 'new' users.

Mapping between functionalities and DILIGENT services (system integration)

The main service involved in this task is the Dynamic VO Support service. This service is in charge to create the trusting environment to operate the DL, as a consequence it is in charge to configure appropriately DL users and DL resources in order to allow the former to have access to the latter. Clearly, the DILIGENT Portal that will offer a user-friendly management modality will support this kind of arrangement.

Use Stories

See 4.2.14 Create a DL.

Testing issues

The testing environment to build in order to test this functionality must be composed by a set of existing users and a set of existing resources. Different pool of users to create will be supplied and the test activity must be capable to verify the correctness of trusted environment created, i.e. verify the correctness of the right granted to the users.

Related non functional requirements

There are many non functional requirements related with this task. First of all the security issue, the grant of rights to the users must be executed in a secure way in order to prevent inappropriate actions and preserve the controlled sharing of resource. Secondly the responsiveness of this task is important as the users will perceive the DL as soon as possible these will have access to the DL resources. Finally, the usability of the task is a sensible

aspect, much simpler and intuitive the process of adding users to a DL will be, much limited the introduction of errors will be.

4.2.18 Create DL Resources

Description and priority

This functionality is related with the second phase of the DL creation action and represents the "creation of resources" that will constitute and realize the DL. Creation of resources must be intended as the creation/sharing of single DILIGENT resource instances, i.e. new resources instances can be physically created (see Section 4.3.11 Create a DL Resource) or already existing resource can be shared just updating its sharing rules (see Section 4.3.10 Add a Resource to a DL).

User Requirements Fulfilled

This functionality contributes to realize the user requirements for DL creation, i.e. *Create an ARTE DL* from ARTE_ucd001 and *Create a DL, Access Configuration, Populate DL, Resources management, Register,* and *Import* from ImpECt_ucd03.

Numbers

We do not expect that the DL creation is a frequent action. It can be executed only by a restrict set of authorized users. So also this operation is executed by the same set of authorized users.

Constraints and Assumptions

There are two main assumptions underlying this functionality, i.e. the existence of the resource to create and the capability to create them. It is clear that in order to create a resource, the system must be capable to identify it and make the appropriate action (giving grant to the DL users on an already existing resource or create a new instance of it) needed to associate the resource to the DL.

UML Diagrams

See the middle part of the diagram in Figure 14: Create a DL in Section 4.2.14.

Grid Exploitation

Grid technologies are particularly appropriate to support this task. In particular the possibility to acquire the pool of computing and storage resources needed to deploy and host a service offering a new functionality is one of the main issue in order to offer a new resource to the DL audience. Another aspect is related with the sharing of a resource, via this mechanism an authorized user is entitled to have access to a resource without being the owner of it, as a consequence the user perceives the resource as its own even if the resource virtually belongs to the DL.

Mapping between functionalities and DILIGENT services (system integration)

The Information Service will be used to maintain the map of the existing resources. The Dynamic VO Support service is in charge to create the trust environment enabling authorized users to have access to the resource.

Use Stories

See 4.2.14 Create a DL.

Testing issues

Testing this functionality will be challenging. First of all is needed to have a testing environment composed by a set of resources of different types, e.g. deployable resources, hosting nodes, storage elements, resources just sharable as could be archives. Then it must

be built some configuration of pool of resources and verify how these poll will be created and satisfied by this functionality.

Related non functional requirements

The reliability of this functionality is an important issue, it is mandatory that resources will be effectively created. Moreover, the scalability of this functionality is critic, the number of resources to create is not known a priori and the process must be capable to scale well when this number increases. Security is another aspect to take care, security means that only authorized users will have access to this task and that authorized users to create a certain resource are not capable to create other kind of resources.

4.2.19 Generate Web Portal

Description and priority

This functionality is related with the second phase of the DL creation action and in particular with the Create DL Resources. In fact, the Web Portal can be considered as one of the resources constituting the DL. So, please refer to Section 4.2.18.

User Requirements Fulfilled

The Web Portal each DL will be equipped with is considered a DL resource, as a consequence this functionality cover the same requirements as reported in Section 4.2.18 Create DL Resources. Moreover it partially fulfils the ARTE_ucd01 'Edit an ARTE DL web portal properties' and ImpECt_ucd02 'Personalize Portal' and 'Load Configuration' UCs.

Numbers

We do not expect that the DL creation is a frequent action. It can be executed only by a restrict set of authorized users. So also this task is executed by the same set of authorized users.

Constraints and Assumptions

This functionality assumes that the Portal will be a resource that can be configured and instantiate using the resource configuration and instantiation mechanisms supplied for DL resources.

UML Diagrams

See the lower part of the diagram in Figure 14: Create a DL in Section 4.2.14.

Grid Exploitation

See Section 4.2.18 Create DL Resources.

Mapping between functionalities and DILIGENT services (system integration)

See Section 4.2.18 Create DL Resources. Moreover, as in this case the Portal is the resource to create, part of this functionality is covered by it.

Use Stories

See 4.2.14 Create a DL.

Testing issues

Testing of this functionality consists in having a portal, with its own configuration parameters, to instantiate and a pool of available hosting node where this portal can be instantiated. The test aims at verifying the appropriate instantiation of the resource.

Related non functional requirements

See 4.2.14 Create a DL.

4.2.20 Maintain a DL

Description and priority

Once a DL is up and running, the system must be able to maintain the level of QoS required and guarantees the responsiveness of the DL Resources by monitoring their status and changing the DL network dynamically by creating and/or moving DL Resources. This functionality provides a set of notification mechanisms (we plan to adopt both the push and the pull model) to investigate the state of a DL Resource and reuses the same procedures used during the creation of a DL in order to re-design the DL.

User Requirements Fulfilled

This functionality satisfies the *Monitor DILIGENT Resources* use-case from ImpECt_ucd01 as well as a 'hidden' requirement underlying the ARTE scenario.

Constraints and Assumptions

To date, a number of notification mechanisms has been already standardized, developed and tested in a different distributed environment. We do not want to implement our own technology to solve this trouble but reuse one of these existing ones.

Moreover, there must exist an Information System that collects and maintains both static and dynamic information about the DL Resources.

UML Diagrams

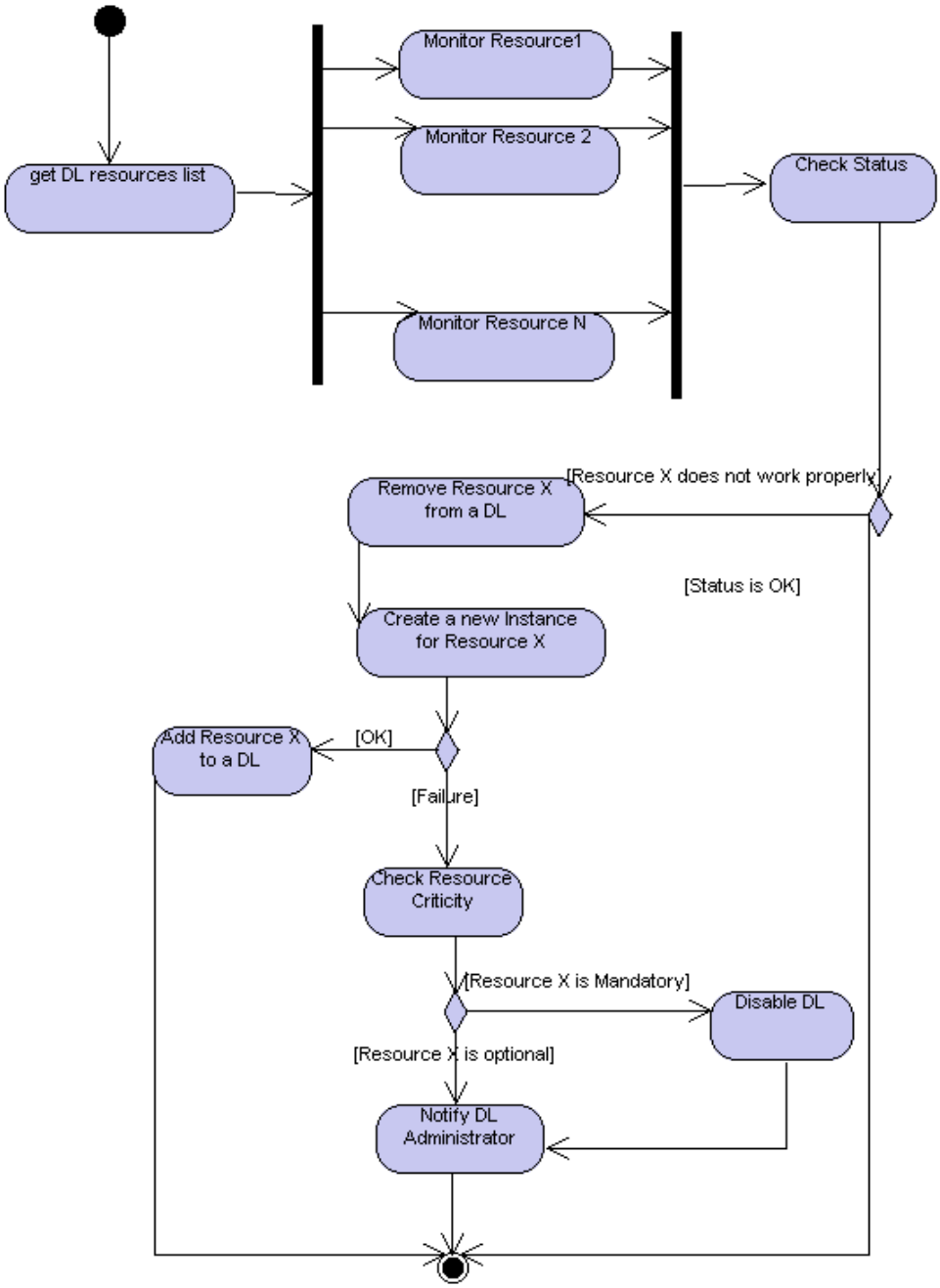

Figure 15: Maintain a DL (activity diagram)

Grid Exploitation

Resource notifications and monitoring are common issues in a Grid environment. We expect that the underlying Grid middleware will provide a set of reusable tools capable to match the DILIGENT specific needs and to simplify our work.

Mapping between functionalities and DILIGENT services (system integration)

The Keeper Service is in charge to monitor DLs. Keeper instances are DL-specific and there must be one instance for DL, at least. It works in collaboration with the Information Service that is queried to obtain information about DL Resources.

Each DL Resource must describe itself in a well-defined format (descriptive metadata) and must have an interface that permits to investigate its status.

Use Stories

See 4.2.14 Create a DL.

Testing issues

In order to test this functionality, a testing environment consisting of a running DL must be prepared. At this point random malfunctions will be introduced within the federation of resource, e.g. removing a resource, making unreachable a shared resource, in order to measure if the system will notify about these 'mistakes' and in which amount of time.

Related non functional requirements

Maintenance of a DL is an important task. The most important requirements is the reliability of this mechanisms, the notification received must be real notification and must cover all the malfunctions and unavailability of resources. The second requirements is related with the scalability of the mechanisms, the system must be capable to notify each DL manager within the same time independently by the 'size' of the DL. Finally, the usability of the mechanism is an important aspect as the DL manager must be enabled to easily understand the problem and solve it.

4.2.21 DL Resources Monitoring

Description and priority

Once a DL is up and running, in order to maintain the required QoS level and guarantees the responsiveness of the DL Resources, the system must be capable to monitor the status of DL resources. This functionality represents the collector of notification coming from DL resources via notification mechanisms (we plan to adopt both the push and the pull model) and the production of alert messages to notify the systems about malfunctions.

User Requirements Fulfilled

This functionality concurs in fulfilling the functionality covered by the Maintain a DL task.

Numbers

The use of this functionality depends on two factors: i) the number of DLs that will be created during the DILIGENT lifetime, and ii) the complexity of them, i.e. the number of resources that forms each of them.

Constraints and Assumptions

This functionality assumes that the system will be capable to monitor the resources forming the DL, i.e. that each resource will supply a mechanism allowing the system to acquire information about its status. If a resource does not supply these data then the system will be just capable to say either if the resource is accessible or not, i.e. a sort of ping mechanism that will not be appropriate for tuning the resource usage within the DL.

UML Diagrams

See Figure 15: Maintain a DL.

Grid Exploitation

Resource notifications and monitoring are common issues in a Grid environment. We expect that the underlying Grid middleware will provide a set of reusable tools capable to match the DILIGENT specific needs and to simplify our work.

Mapping between functionalities and DILIGENT services (system integration)

The service in charge to cover this functionality is the Information Service that will be capable to supply data about the resources belonging to DILIGENT. However, in order to supply this information, it will be capable to acquire them directly from these resources that are in charge to offer them.

Use Stories

See Section 4.2.14 Create a DL.

Testing issues

See Section 4.2.20 Maintain a DL.

Related non functional requirements

See Section 4.2.20 Maintain a DL.

4.2.22 Report DL Status

Description and priority

As the system is capable to monitor the status of DL resources, it must be also capable to notify the *DL Manager* about malfunctions and errors that prevent the DL service. This functionality represents the notification made by the system of alert messages presented in Section 4.2.21. Notification can be done in different ways, e.g. a mail, or whatever kind of message could be automatically sent to the administrator.

User Requirements Fulfilled

This functionality concurs in fulfilling the functionality covered by the Maintain a DL task.

Numbers

The use of this functionality depends by three factors: i) the number of DLs that will be created, ii) the size of these DLs, and iii) the number of faults that will happen and will be discovered. The DL usability will be high if the third component of this list will be as lower as possible.

Constraints and Assumptions

This functionality assumes that the system will offer a notification mechanism. No assumption and constraint are posed on the kind of mechanism, i.e. push modality or pull modality.

UML Diagrams

See Figure 15: Maintain a DL in Section 4.2.20.

Grid Exploitation

This functionality has no particular issues in a Grid environment.

Mapping between functionalities and DILIGENT services (system integration)

The main service involved in this functionality is the Portal service that is in charge to render to the DL manager the notification directed to him by the system.

Use Stories

See Section 4.2.14 Create a DL.

Testing issues

See Section 4.2.20 Maintain a DL.

Related non functional requirements

See Section 4.2.20 Maintain a DL.

4.2.23 Update a DL

Description and priority

As already stated, the creation of DLs is made in two steps: the definition phase and the real generation of the DL. Also the operation of update follows this schema, i.e. there is a

definition phase (see 4.2.7 Modify a DL) where the *DL Designer* express the changes needed to its DL and a physical execution phase where the *DL Manager* execute the modification requested. The latter operation is represented by this functionality.

As for the *Create a DL* functionality, starting from the list of resources identified in the definition step, the manager/system will be capable to identify the resource to be added to/updated/removed from the DL and then execute these actions using functionality belonging to the Resources Management area.

User Requirements Fulfilled

This functionality fulfils the ARTE_ucd01 'Update an ARTE DL' and, in part, the ImpECt_03 'Update a DL' UCs.

Numbers

This functionality will be executed in accordance with the number of DLs that will be created within DILIGENT. This number will grow in accordance with the number of communities that will join the infrastructure. Moreover, this number will also depend on the dynamicity of the various DLs, e.g. if a DL is created with all the resources needed to fulfil the user needs for the whole DL lifetime this functionality will be never used, if a DL will be continuously redesigned via the 'Modify a DL' functionality this functionality will be executed in accordance.

Constraints and Assumptions

Assumptions for this functionality are similar to those identified in DL Creation, i.e. it exists a DILIGENT portal that permits the operations described above. Moreover, there must exist an Information System (to extract information about available grid nodes and their configuration, as well as a description of other available resources), a Broker service (to implement matchmaking algorithms among services requirements and nodes equipment) and mechanisms for dynamic software deployment.

UML Diagrams

This functionality is similar to the creation phase with the only difference that some of the resources forming the DL already exist. As a consequence see Figure 14: Create a DL in Section 4.2.14.

Grid Exploitation

See Section 4.2.14 Create a DL.

Mapping between functionalities and DILIGENT services (system integration)

During a DL update, the potential Grid exploitation is very high. Within this functionality, the Grid infrastructure will be used as a single big computational environment where DL Resources can be dynamically located and moved in order to guarantee the Qualities of Services requested by *DL designers*.

Use Stories

This functionality represents the second step of the story reported in Section 4.2.7 Modify a DL. During this phase the choice expressed by the DL designer take place and are transformed into physical rearrangement of the running DL. For example, if the DL designer has decided to add an archive to its DL, during this phase the archive is really added to the DL, i.e. at the completion of this activity the DL users are entitled to see the new archive and have access to it.

Testing issues

The test of this functionality is similar to those proposed for the creation phase. This functionality is related with the instantiation of a distributed system in accordance with

some definition criteria. First of all it must be identified an appropriate matching algorithm in order to verify that the created resources fulfils the expressed requirements. Secondly it must be identified and realized a sort of "secure ping mechanism" for testing the access to the services forming the DL in a controlled way.

Related non functional requirements

The usability of the functionality is mandatory. The DL manager will be enabled to easily identify the pool of resource to instantiate and also to express and execute all the rearrangement of the DL that she/he deem as relevant to fulfil the new DL requirements. The reliability is another main point: the operation performed by the DL manager must be reported on the physical pool of resources managed. Underlying this task there is also the needs to have a controlled access to this functionality, i.e. the system must prevent improper use of this functionality to unauthorized users.

4.2.24 Remove a DL

Description and priority

The operation of removal of a DL follows the two phase schema presented for other operation related to DLs, i.e. there is a sort of definition phase (see 4.2.9 Dispose a DL) where the *DL Designer* express the requirements to remove a DL and a physical execution phase where the *DL Manager* execute the request. The latter operation is represented by this functionality. In order to remove a DL, all the resources belonging to the DL must be removed (see 4.2.25) taking into account preservation requirements expressed by the *DL designer* (see 4.2.10 Preserve content).

User Requirements Fulfilled

This functionality fulfils the ARTE_ucd01 'Remove an ARTE DL' and, in part, the ImpECt_03 'Dispose a DL'.

Numbers

The utilization of this functionality is related with the dynamicity of the created DLs. It is clear that if the DILIGENT mechanism to create DLs will be easy to use an open to an high number of community, as a consequence a big number of DLs will be created to serve temporary user needs. When the community that brought up the DL ends its short activity the DL must be removed, i.e. its resources must be deallocated.

Constraints and Assumptions

This functionality has no particular assumption and constraint.

UML Diagrams

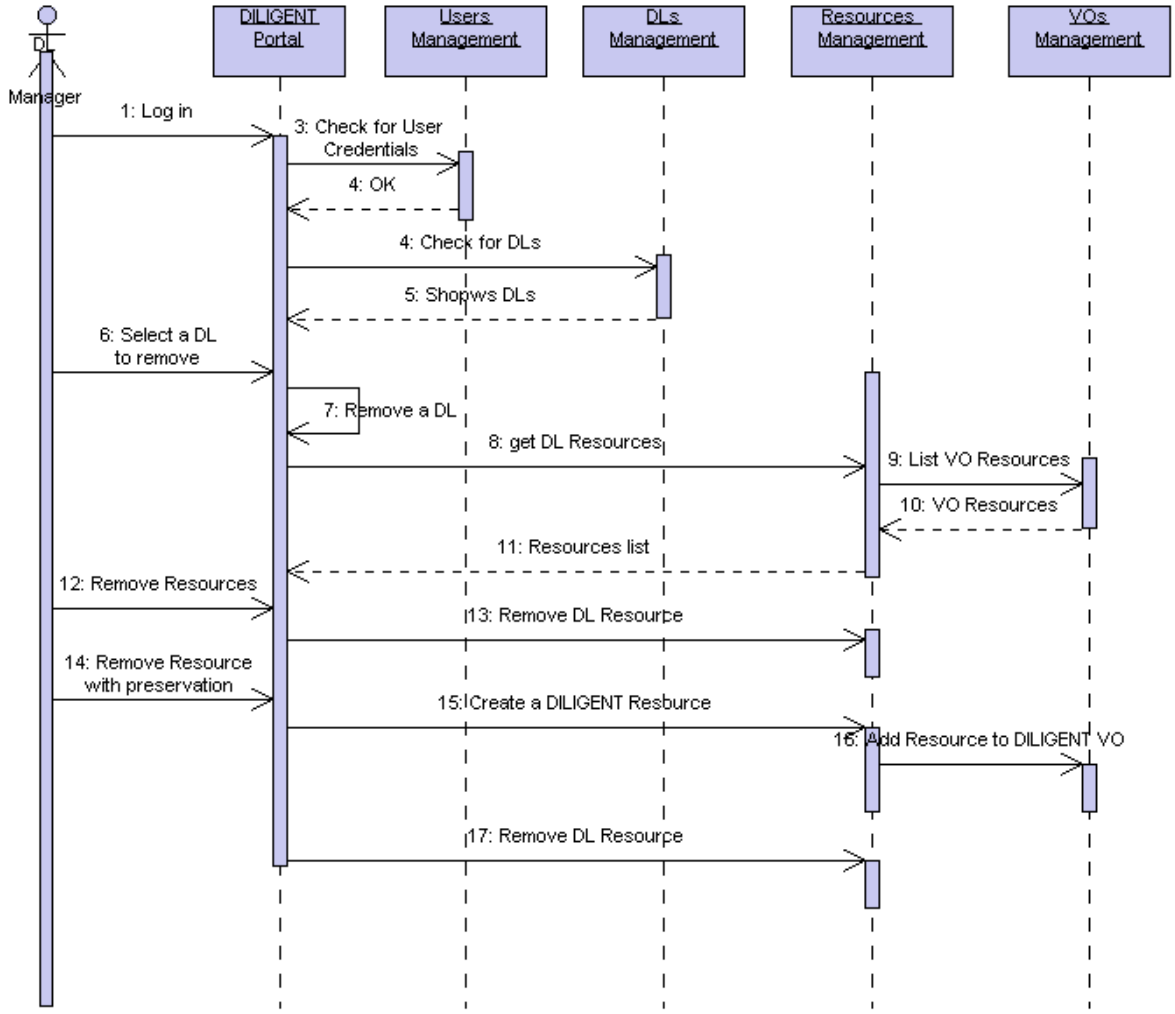

Figure 16: Remove a DL (sequence diagram)

Grid Exploitation

During the DL removal, the potential Grid exploitation is very limited. However this operation must be executed keeping in mind that the operational environment is a Grid environment where resources are shared across different DLs. This mean that if the resource to remove is owned by the DL it must be physically removed, on the contrary if the resource is shared, this operation is just in charge to update the resource data in order to prevent future access to the resource by users of this DL.

Mapping between functionalities and DILIGENT services (system integration)

Services involved into this functionality are the Dynamic VO Support service and the Information Service as back end element, and the Portal service as the front end element, i.e. the service offering this functionality in a user friendly fashion. The Dynamic VO Support service is in charge to rearrange the trust relationship among shared resources in order to restore the correct resource access rules. The Information Service must remove the entry about the resources that will be removed.

Use Stories

This task represents the second phase of the user story reported in Section 4.2.9 Dispose a DL. The DL manager performs the technical operation to physically removing the pool of resources forming the DL. All these operations are made in accordance with the

requirements and guidelines expressed by the DL designer. These guidelines are mainly related with preservation actions.

Testing issues

In order to test this functionality, various DLs will be built and different preservation directive should be indicated over these environments. The objective of the testing activity is to verify the *correctness*, i.e. only appropriate resources are physically removed while shared resources are 'virtually' removed, and *completeness*, i.e. all the resources forming the DL are removed, of the operation executed.

Related non functional requirements

Reliability of this functionality is a must, i.e. all the operation performed by the DL manager must take effectively place in order to prevent the existence of garbage resources within the infrastructure. The other must is the security, i.e. only authorized users must be entitled to have access to this functionality and execute it in a successful way.

4.2.25 Remove DL Resources

Description and priority

In order to remove a DL, all the resources belonging to the DL must be removed taking into account the preservation needs expressed by the *DL Designer* (see 4.2.10). As previously observed, resources may be of exclusive use/visibility of the DL or may be shared between DLs. Resources will be removed from the DL and/or from the DILIGENT system using the corresponding functionality belonging to the Resources Management area (see 4.3.15). This functionality is capable to take into account the preservation requirements expressed and the underlying sharing rules.

User Requirements Fulfilled

This functionality concurs to cover the requirements fulfilled by the Remove a DL functionality.

Numbers

This operation will be executed in accordance with the Remove a DL functionality.

Constraints and Assumptions

See Section 4.2.24 Remove a DL.

UML Diagrams

See Figure 16: Remove a DL in Section 4.2.24.

Grid Exploitation

See Section 4.2.24 Remove a DL.

Mapping between functionalities and DILIGENT services (system integration)

See Section 4.2.24 Remove a DL.

Use Stories

See Section 4.2.24 Remove a DL.

Testing issues

See Section 4.2.24 Remove a DL.

Related non functional requirements

See Section 4.2.24 Remove a DL.

4.3 Resources Management

In traditional computing systems, resource management is a well-studied problem. These resource management systems are designed and operate under the assumption that they have complete control of a resource and thus can implement the mechanisms and policies needed for effective use of that resource in isolation. Unfortunately, this assumption does not apply to the Grid.

The current research and investment into computational grids is motivated by the assumption that coordinated access to diverse and geographically distributed resources is possible. In our context, we expect that some of the of work will be covered by the grid middleware adopted. In the DILIGENT system, the role of *Resources Management* is to provide mechanisms to:

- Join resources with one or more specific DLs
- Manage and store sharing rules that allow coordinated access
- Dynamically allocate new resources

We have two kinds of resource: the DILIGENT Resources (simply called Resources) and the DL Resources. Example of the former is:

- Software packages that can be deployed as DILIGENT service
- Running instances of DILIGENT services
- Pieces of software with self contained procedures
- Pre-existing resources (legacy applications, persistent archives) accessible via public interfaces compliant with the DILIGENT specifications
- Complex objects, i.e. compound services, collections of objects, and for example reports of the ImpECt scenario and exhibition catalogues of the ARTE scenario.

A DL Resource is just a DILIGENT resource that becomes available for that DL.

Regarding the use and access rights, the resource owner is entitled to specify, for each resource, who (consumer) can have access to the resource to perform what (action). The consumer is a VO while an action is a resource-specific task.

The rule above should be read as: "*a community X is entitled to invoke an Action Y on Resource Z*".

Main actors of this area are the *DILIGENT Resource Manager* and the *DL Manager*.

DILIGENT Resource Managers act as the owners of the resources. They are in charge to register their resources to the DILIGENT infrastructure and to add them to one or more Virtual Organizations making them available in concordance with a set of policies.

DL Managers are authorized users that manage registered resources in the context of their digital libraries to provide functionalities to the end-users.

Generic DILIGENT users can only search, browse, and monitor resources belonging to their virtual organization.

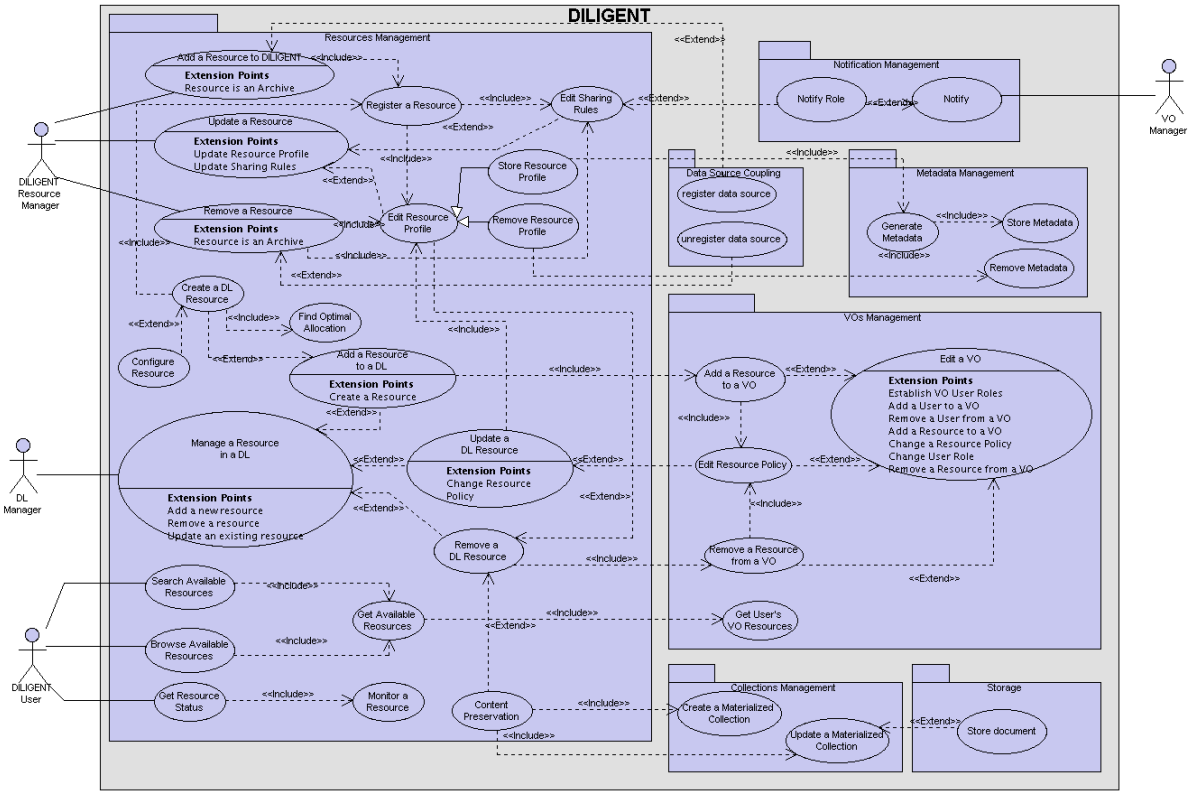

Figure 17: Resources Management (use case diagram)

4.3.1 Add a Resource to DILIGENT

Description and priority

This functionality models the action enabling Resource Manager to make available resources to the DILIGENT infrastructure. In order to add this resource to the system it must comply with the DILIGENT requirements that will be defined in the design phase.

User Requirements Fulfilled

This functionality covers the *'Add a Service'* and *'Add an Archive'* use-cases from ARTE_ucd001, *'Register'* use-case from ImpECt_ucd03 and *'Add Resources'* from ImpECt_ucd01.

Numbers

In this case, numbers depend on the success of the project. If the DILIGENT infrastructure will be widely joined the registration of new resources will occur frequently, otherwise, if it will be used just as a first experimental testbed, the operation will occur rarely.

Constraints and Assumptions

In order to successfully register a resource, this must be compliant with some requirements that will be identified in the following phases of the project, e.g. a resource description must be available and coded in a well-defined format.

UML Diagrams

See Figure 18: Register a Resource.

Mapping between functionalities and DILIGENT services (system integration)

The Information Service, with the support of the Portal for the presentation aspect, is in charge to offer this functionality. Another service involved will be the Dynamic VO Support Service that is in charge to register the resource within the correct sharing environment.

Use Stories

A Resource owner, e.g. an Institution hosting an archive or a service provider that is capable to supply a particular kind of functionality, plans to offer this resource to all the communities that are interested in using it. She/he establishes the rules for utilizing its resource and then notifies the DILIGENT infrastructure that in some way will grab the resource and offer it to its community. From this point on, each DL designer or DL manager is enabled to use the new resource for fulfilling the user requirements in a novel way. This kind of usage must be executed in accordance with the sharing rules expressed by the resource owner; the DILIGENT infrastructure is in charge for ensuring this controlled sharing.

Testing issues

Testing issues are related with the correctness and completeness of this operation. The goal of test is to verify that different kind of resources are correctly inserted within the infrastructure, i.e. that the infrastructure become aware of these resources in a reasonable amount of time.

Related non functional requirements

In order to enrich the DILIGENT infrastructure with the higher number of resources, the process of registering a new resource must be as much as possible easy and user friendly, as a consequence the usability of the functionality is an important aspects. On the contrary this registration activity must be executed in a secure fashion in order to prevent that garbage resources will invade the infrastructure.

4.3.2 Register a Resource

Description and priority

In order to be shared a resource must be prepared and registered. Resources that are managed by grid applications like DILIGENT or those even within a grid node are heterogeneous in nature, so prepare a resource is a resource-specific task. For instance, adding a new node capable to host DL resources means to equip it with the necessary software packages, trust it with the necessary security procedures and configure it; on the other hand, adding a pre-existing resource (a legacy application, a persistent archive, a storage element, etc.) means to configure it to be accessible via a public interface compliant with the DILIGENT specifications.

Once a resource is prepared, the Resource Manager must set its use and access rules that regulate the resource usage and notify the DILIGENT VO Manager about the new resource.

User Requirements Fulfilled

Resources registration satisfies a very basic requirement in the Grid environments. Moreover this functionality contributes to fulfill the requirements expressed in Add a Resource to DILIGENT section.

Numbers

The execution of this functionality will depend on the number of resource will be registered and make available via DILIGENT.

Constraints and Assumptions

A well-defined specification should result from the design phase. Following this specification, any type of resource could be added to the DILIGENT infrastructure.

UML Diagrams

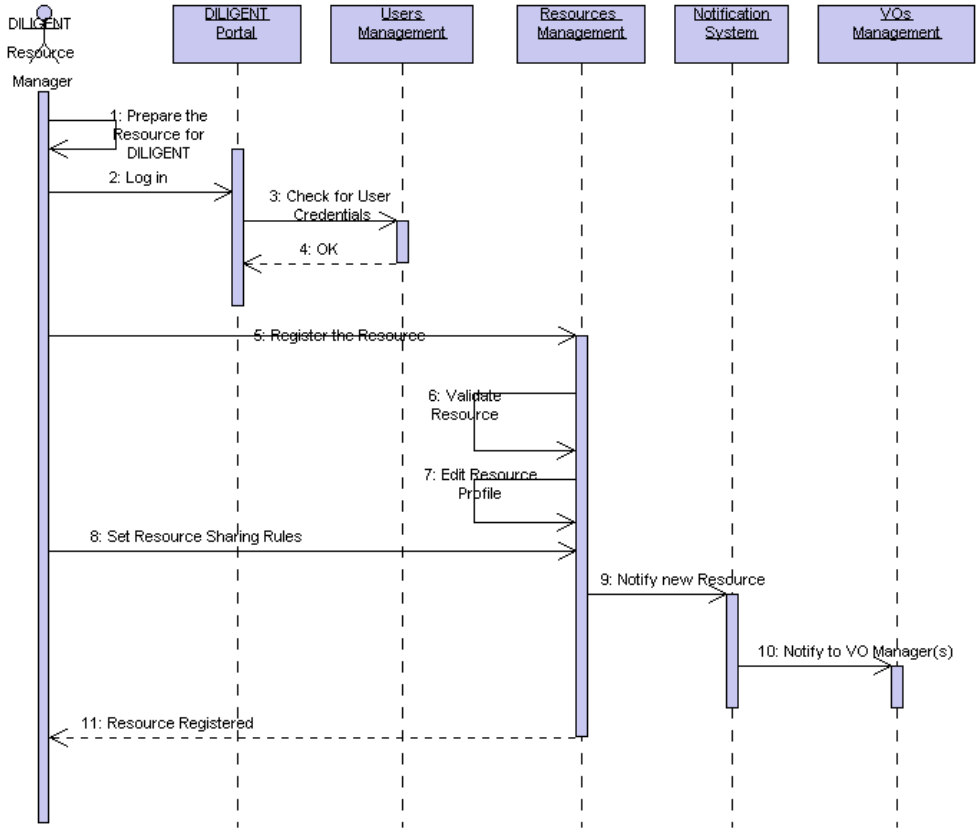

Figure 18: Register a Resource (sequence diagram)

See also Figure 17.

Grid Exploitation

Authorization and authentication mechanisms of the underlying middleware will be exploited in order to do this task. Standard ways to exchange authorization and authentication information (such as the Security Assertion Markup Language) will be taken into account in order to securely exchange resource policies.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support Service is in charge to offer this functionality.

Use Stories

See 4.3.1 Add a Resource to DILIGENT.

Testing issues

See 4.3.1 Add a Resource to DILIGENT.

Related non functional requirements

See 4.3.1 Add a Resource to DILIGENT.

4.3.3 Edit Sharing Rules

Description and priority

In the Resource Management framework, resource usage is governed by a set of "sharing rules" specifying the actions that consumers can perform on the resource. The consumer is a VO while an action is a resource-specific task.

Sharing rules are defined following the sharing rules specification that will be defined in the detailed design phase. It is important to note that this specification will be user-oriented.

User Requirements Fulfilled

This functionality concurs to fulfill the requirements reported in Section 4.3.1.

Numbers

It is supposed that setting, updating, or deleting sharing rules will occur frequently.

Constraints and Assumptions

Edit sharing rules is a sub-task of the *Register a Resource*, *Update a Resource*, and *Create a DL Resource* satisfying a very basic requirement in the Grid environment. Sharing rules specification should be defined during the detailed design phase. It is assumed that this specification will be user-oriented. Moreover, a sort of transactional protocol for this task must be supplied in order to manage concurrent access to the same resource sharing rules by different users.

UML Diagrams

See Figure 18: Register a Resource, Figure 19: Add a Resource to a DL, and Figure 20: Create a DL Resource.

Grid Exploitation

Grid technologies already have mechanisms for setting the sharing rules on resources. The exploitation consists in analyzing these mechanisms in order to identify its strengths and weakness.

Mapping between functionalities and DILIGENT services (system integration)

Services involved into this activity are: i) the Portal by witch authorized users are enabled to set and update the sharing rules about a resource, ii) the Dynamic VO Support service that must act in accordance to these sharing rules and rearrange the policies in order to enforce them.

Use Stories

See Section 4.3.1 Add a Resource to DILIGENT.

Testing issues

This functionality is critical, as a consequence its testing activity must be an heavy task. First of all, correctness and completeness of this functionality must be tested. Moreover, this task must be subject of stress tests as it will be used in a concurrent fashion by different other tasks.

Related non functional requirements

Usability and security are important requirements for this functionality as reported in Section 4.3.1.

4.3.4 Edit Resource Profile

Description and priority

Edit Resource Profile provides the mechanisms to maintain (add, update, and remove) detailed information about resources, i.e. about software packages, running instances of DILIGENT services, pieces of software with self contained procedures, pre-existing resources (legacy applications, persistent archives) accessible via public interfaces compliant with the DILIGENT specifications, complex objects (like Reports in the ImpECt scenario), collections of objects. This information is updateable anytime and is composed by an XML-based configurable set of properties.

User Requirements Fulfilled

This functionality concurs to fulfill the requirements reported in Sections 4.3.2, 4.3.7, and 4.3.8.

Numbers

The execution of this task depends by two factors: i) the 'dynamicity' of the resource, and ii) the information that will be maintained by the profile.

Constraints and Assumptions

Resource profile will be maintained as XML data and will be possible to store and retrieve them in an efficient way.

UML Diagrams

See Figure 18: Register a Resource, Figure 20: Create a DL Resource.

Grid Exploitation

This task can be considered a data intensive task. As a consequence the exploitation of Grid it related with the possibility to store the profiles in a distributed way making use of storage resources offered by the infrastructure. Clearly, information stored in this distributed way must be searchable in an efficient way and the system must take care about the preservation of these data, mainly the loss of them must be prevented.

Mapping between functionalities and DILIGENT services (system integration)

The Information Service will maintain resource profiles, i.e. we figure out that each service and/or user must query the Information Service to acquire data about resources. The service in charge for visualize these data is the Portal.

Use Stories

See Section 4.3.1, 4.3.7, and 4.3.8.

Testing issues

Correctness and completeness of this functionality must be tested. Moreover, this task must be subject of stress tests as it will be used in a concurrent fashion by different other tasks.

Related non functional requirements

Security, scalability and reliability of this functionality are orthogonal requirements that must be traded off in implementing it. With security we mean that only authorized users are entitled to act on resources profiles as well as that concurrent access will be correctly handles. Scalability is related with the performance of this task, i.e. the responsiveness does not degrade when the number of resources increases. Reliability means that resources profiles will be changed accordingly to the user expressed actions, i.e. unpredictable changes due to side effects are not allowed.

4.3.5 Store Resource Profile

Description and priority

Store Resource Profile models the storage action into the resources knowledge base of the information constituting the resource profile. This information is updateable anytime and is composed by an XML-based configurable set of properties.

User Requirements Fulfilled

This functionality has been obtained by factoring out a specific instance of the Set Resource Profile task.

Numbers

The execution of this task depends by two factors: i) the 'dynamicity' of the resource, and ii) the information that will be maintained by the profile.

Constraints and Assumptions

See Section 4.3.4 Edit Resource Profile.

UML Diagrams

See Figure 18: Register a Resource and Figure 20: Create a DL Resource.

Grid Exploitation

This task is a data intensive task. As a consequence the exploitation of Grid it related with the possibility to store the profiles in a distributed way making use of storage resources offered by the infrastructure. Clearly, information stored in this distributed way must be searchable in an efficient way and the system must take care about the preservation of these data, mainly the loss of them must be prevented as well as the synchronization of eventually replicated data.

Mapping between functionalities and DILIGENT services (system integration)

See Section 4.3.4 Edit Resource Profile.

Use Stories

See Section 4.3.4 Edit Resource Profile.

Testing issues

Correctness and completeness of this functionality must be tested, e.g. data will be stored in a complete way, replicated copies of data are modified accordingly, etc. Moreover, this task must be subject of stress tests as it will be used in a concurrent fashion by different other tasks.

Related non functional requirements

See Section 4.3.4 Edit Resource Profile.

4.3.6 Remove Resource Profile

Description and priority

Remove Resource Profile models the removal action from the resources knowledge base of the information constituting the resource profile. This information is updateable anytime and is composed by an XML-based configurable set of properties.

This functionality has been obtained by factoring out a specific instance of the Set Resource Profile task.

Numbers

The execution of this task depends by the 'dynamicity' of the resources, i.e. if resources will be continuously added to and removed from the infrastructure this task will be executed many times.

Constraints and Assumptions

See Section 4.3.4 Edit Resource Profile.

UML Diagrams

See Figure 17: Resources Management in Section 4.3 Resources Management, no further diagrams is needed.

Grid Exploitation

This task is a data intensive task. As a consequence the exploitation of Grid it related with the possibility to remove profiles that potentially are stored in a distributed way making use of storage resources offered by the infrastructure.

Mapping between functionalities and DILIGENT services (system integration)

See Section 4.3.4 Edit Resource Profile.

Use Stories

See Section 4.3.4 Edit Resource Profile.

Testing issues

Correctness and completeness of this functionality must be tested, e.g. data will be removed in a complete way, replicated copies of data are removed accordingly, etc. Moreover, this task must be subject of stress tests as it will be used in a concurrent fashion by different other tasks.

Related non functional requirements

See Section 4.3.4 Edit Resource Profile.

4.3.7 Update a Resource

Description and priority

Registered resources can be updated in order to change their profiles or to modify the sharing rules. Both tasks are human-driven. The former is a task that can be activated to modify the property values related to a resource, or to change the set of properties used to describe the resource. The latter can be activated to modify, add, or remove consumers and/or actions.

User Requirements Fulfilled

This functionality completes the pool of classical management operations, i.e. add, update and remove resources.

Numbers

The execution of this task depends on the dynamicity of the resource, i.e. the number of times it is needed to update its profile as well as the number of times it is necessary to update its sharing rules. All these changes are executed by a Resource Manager in order to enlarge or restrict the usage of the resource within DILIGENT.

Constraints and Assumptions

We assume that the only authorized users are entitled to execute this task.

UML Diagrams

This functionality is quite similar to the functionality that add a new resource to the system, in particular it deals with modifications of resource sharing rules (see 4.3.3 Edit Sharing Rules) and/or with modifications of resource data (see 4.3.4 Edit Resource Profile).

Grid Exploitation

See Section 4.3.1.

Mapping between functionalities and DILIGENT services (system integration)

This functionality involves two services, the Information Service is in charge to update resource profile data and the Dynamic VO Support service will be take care about new resource sharing rules.

Use Stories

A Resource Manager has registered a resource within DILIGENT and has allowed a community of mathematicians to use it. A new multidisciplinary community joins DILIGENT and contact the Resource Manager in order to have access to the resource as they deem it particularly relevant for their activity. The Resource Manager decides to open up its resource for free for a trial period to this community. In order allow the new community to use the resource, she/he update the resource. At the end of the trial period, the Resource Manager can update the resource if the community is not capable or does not agree to 'pay' for use it.

Testing issues

Testing issues are related with the correctness and completeness of this operation. The goal of test is to verify that different kind of resources are correctly updated within the infrastructure, i.e. that the infrastructure become aware of resources changes in a reasonable amount of time mainly for sharing issues.

Related non functional requirements

In order to enrich the DILIGENT infrastructure with the higher number of resources, the process of update of a resource must be as much as possible easy and user friendly in order to give to the DL Manager complete control on resource usage, as a consequence the usability of the functionality is an important aspects. On the contrary this change activity must be executed in a secure fashion in order to prevent that unauthorized usage of resources will be possible.

4.3.8 Remove a Resource

Description and priority

In a Grid environment resources are not under the control of a single organization or institution. Resources made available by their owners can be removed through a specific human–based request or can be automatically dropped out for a scheduled life-time management request or because they appear unstable or not correctly managed with respect to the rules established by the DILIGENT specification. When a resource is dropped out each VO manager is notified. If the resource is used within a DL the DL Manager is notified by the monitor resource functionality (push modality).

User Requirements Fulfilled

This functionality fulfils ARTE_ucd01 'Remove a Service', 'Remove an Archive' and ImpECt_ucd01 'Delete' UCs.

Numbers

The number of times this functionality will be used depends on the number of third party resources that will be added to the system as well as by the correctness and secure resource usage mechanisms offered by DILIGENT.

Constraints and Assumptions

UML Diagrams

This functionality deals with the modification of resource sharing rules (see 4.3.3 Edit Sharing Rules), the removal of resource data (see 4.3.4 Edit Resource Profile) and, finally, the physical removal of the resource, e.g. free resources like storage elements used by the resource, etc.

Grid Exploitation

This functionality is resource specific as well as the grid exploitation. As a consequence the removal of a resource could imply the necessity to free the other Grid resources that have been reserved to this resource.

Mapping between functionalities and DILIGENT services (system integration)

Services involved are i) the Information Service, which is in charge to remove the entry about the resource from its knowledge base, ii) the Dynamic VO Support service that must rearrange the sharing rules removing those related with the resource, and iii) the Portal that will supply the user interface enabling the Resource Manager to perform this action. Removal of a resource involves also all the Keeper services that are in charge to maintain the various DL where the resource is used, those services must find alternative resources to equip with their DL or must notify the DL Manager about the impossibility to maintain the DL.

Use Stories

The DILIGENT Administrator after having analyzed the pool of resources registered within the infrastructure discover that exists an archive whose content have changed a lot from the time this archive has been published within the system. Today this archive contains questionable quality documents. Moreover, this archive makes use of storage resources offered by the infrastructure. In order to guarantee a certain quality of the resource offered as well as optimize the use of them the DILIGENT Administrator decides to remove the archive and makes its storage capacity available for others usage.

Testing issues

The goal of the testing activity is to verify the correctness and completeness of this task. Different kind of resources must be created and then removed. These resources will be realized as single components as well as resources that make use of other resources offered by the infrastructure and reserved to them. The removal of the former is more easy, while the removal of the latter impose also to free the reserved resources.

Related non functional requirements

Security is the first requirements that this functionality asks for; only authorized users must be entitled to remove a resource. Reliability is the other requirement; in particular an authorized user that remove a resource must have assurance that the resource is really dropped out.

4.3.9 Manage a Resource in a DL

Description and priority

This functionality represents the aspects related with the management of a Resorce within a DL. This management activity can be decomposed into three sub-functionalities:

- the adjunction of a Resource to a DL (see 4.3.10);
- the updating of a DL Resource (see 4.3.14);
- the removal of a resource from a DL (see 4.3.15);

User Requirements Fulfilled

See the more specific functionalities.

Numbers

See the more specific functionalities.

Constraints and Assumptions

See the more specific functionalities.

UML Diagrams

See the more specific functionalities.

Grid Exploitation

See the more specific functionalities.

Mapping between functionalities and DILIGENT services (system integration)

See the more specific functionalities.

Use Stories

See the more specific functionalities.

Testing issues

See the more specific functionalities.

Related non functional requirements

See the more specific functionalities.

4.3.10 Add a Resource to a DL

Description and priority

Registered resources can be associated with a DL in order to fulfill specific DL requirements. This functionality can be activated by a human-driven request or can be started by DILIGENT services to share a service instance capable to fulfill the DL requirements and/or to balance the workload, to manage temporarily peak of requests, or to overcome malfunction of servers or partitioning of the network. Human-based requests have to specify the policy for the DL resource; automatic generated DILIGENT services requests set for the DL resource the same policy of other similar resources.

User Requirements Fulfilled

This functionality is a consequence of the DL creation model adopted into DILIGENT as well as by the kind of resources the DILIGENT infrastructure plans to manage. In fact, after the DL design performed by the DL Designer, the DL Manager is in charge for identifying the pool of resources needed for fulfilling the designer needs and physically add them to the federation of resources forming the DL. This task may require the creation of new resources (see 4.3.11 Create a DL Resource) as well as the sharing of already existing ones.

Numbers

This task will be executed a number of times that depends on the number of DLs that will be created as well as by the size of each DL.

Constraints and Assumptions

Resources are already registered as DILIGENT resources. Moreover, the system will be capable to instantiate the resources that can be instantiated as well as must be capable to appropriately rearrange the pool of sharing rules in order to make the resource available to the DL.

UML Diagrams

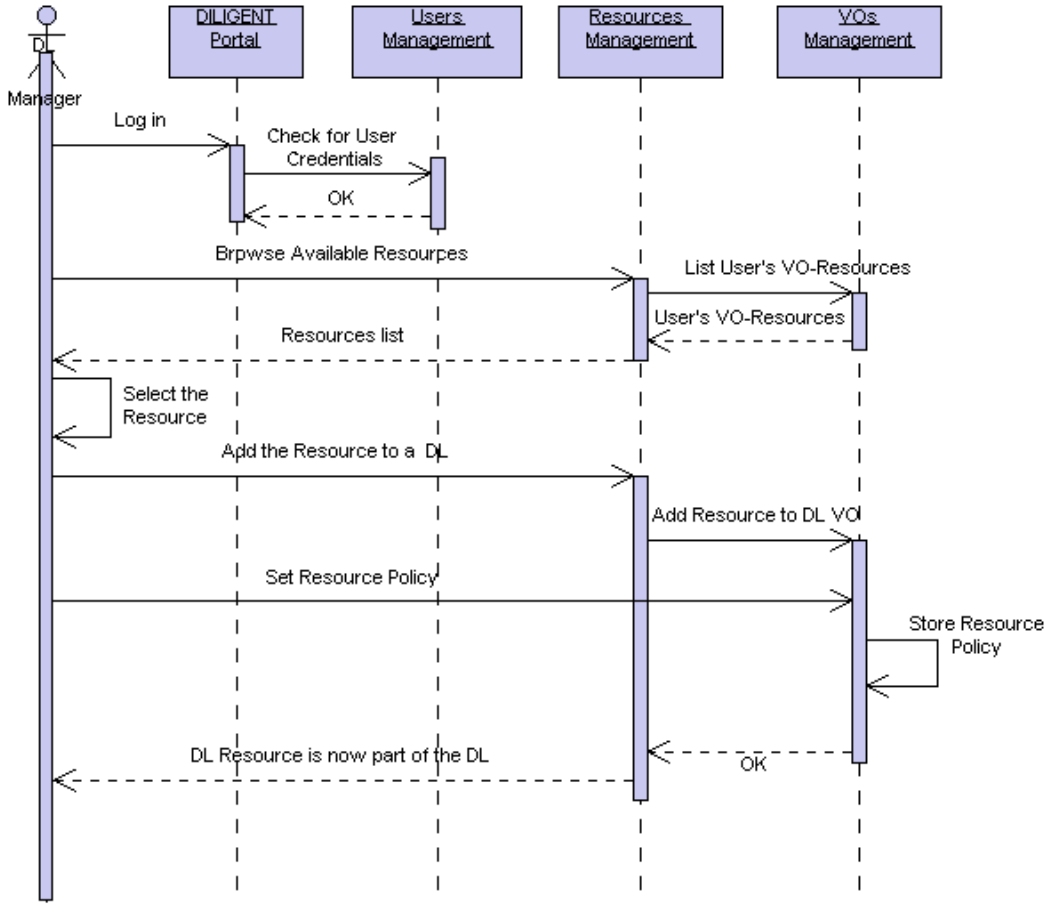

Figure 19: Add a Resource to a DL (sequence diagram)

See also Figure 18: Register a Resource.

Grid Exploitation

This task will take many advantages by using Grid technologies. As previously stated, if it is needed to create a new resource the system may ask for an hosting node that will be gathered by the infrastructure, as well as for a storage element that again is gathered by the infrastructure. On the other hand, if the DL Manager should be capable to share an already existing resource it will rearrange the policy statements.

Mapping between functionalities and DILIGENT services (system integration)

Services involved within this task are the Information Service, the Dynamic VO Support service and the Keeper Service. Information Service will maintain the knowledge base of all the resources available within the DILIGENT infrastructure, as a consequence each time a new resource is created it must be notified. The Dynamic VO Support service maintains the knowledge base of the sharing rules holding within the infrastructure, i.e. it is in charge to say if a certain resource is accessible within an environment (a DL, a community, etc.).

Use Stories

In order to fulfill the DL requirements expressed by the DL Designer, the DL Manager is in charge to identify the pool of resources needed and then adding them to the DL environment. At this point, with the support of the system, the DL manager will be capable to create the new resource instances he identifies as needed as well as 'share' already existing ones, i.e. make already existing resource available and accessible to the DL community.

Testing issues

The main goal of the testing activity is verifying the correctness and completeness of this action. Moreover, the time needed to create a new DL resource is another important aspect to be considered. As a consequence, a pool of representative resources among those the system is capable to manage must be selected and diverse infrastructure environment must be used to instantiate/share them.

Related non functional requirements

Due to the importance of this functionality, its reliability is a must, i.e. if a DL manager asks for a new DL resource then this resource must be effectively made available to DL users. Another aspect to keep in mind is the scalability of this task, the response time must be as much as possible independent by the number of resources to create. Interoperability of this functionality is also important, where we intend that this functionality must be capable to deal with the greater number of resource possible, e.g. it must be capable to instantiate a search service as well as an index service or a repository, it must be capable to share an archive as well as a running instance of an index. Finally, security is another mandatory requirement: only authorized users must be entitled to execute this kind of functionality.

4.3.11 Create a DL Resource

Description and priority

As mentioned above, a DL Resource is a Resource that can be used by a DL. It can be created for a specific DL as well as shared among a number of DLs. This functionality covers the creation of a new resource for a DL in order to fulfill the functionalities of the DL, to balance the workload, to manage temporarily peak of requests, or to overcome malfunction of servers or partitioning of the network. For example, a service instance is created from scratch when a new DL is set up or when becomes necessary a workload balancing.

The new resource must be registered as DILIGENT Resource on behalf of the *DL Manager* (this is the reason why *DL Manager* is a specialization of *DILIGENT Resources Manager* actor, see Figure 7: DILIGENT actors). Human-based requests have to specify the sharing rules; automatic generated DILIGENT services requests set for the DL resource the same sharing rules of other similar resources. At the end the DL Resource will be added to the DL VO.

User Requirements Fulfilled

This functionality is a consequence of the DL creation model adopted into DILIGENT as well as by the kind of resources the DILIGENT infrastructure plans to manage. As some of the DILIGENT resources can be deployed making use of other resources, this functionality covers this activity.

Numbers

This task will be executed a number of times that depends by the number of DLs that will be created as well as by the size of each DL and by the number of resources within them that can be deployed.

Constraints and Assumptions

The main assumption underlying this functionality is the presence of DILIGENT resource that can be dynamically deployed. Usually resource to deploy are services, as a consequence to deploy a resource it is mandatory for the system to be capable identify an appropriate hosting node where the service can be deployed. Moreover, if the DILIGENT service will supply resources that can be virtualized, e.g. a storage element capable to offer a 10GB capacity can be virtualized into two storage elements of 5GB each, the creation of a resource can be simply the virtualization of an existing one. The second aspect is also

possible if the resource support a sort of *reservation* mechanisms, i.e. creating a 5GB storage element means that the original resource reserve this amount of space to the particular application or community.

UML Diagrams

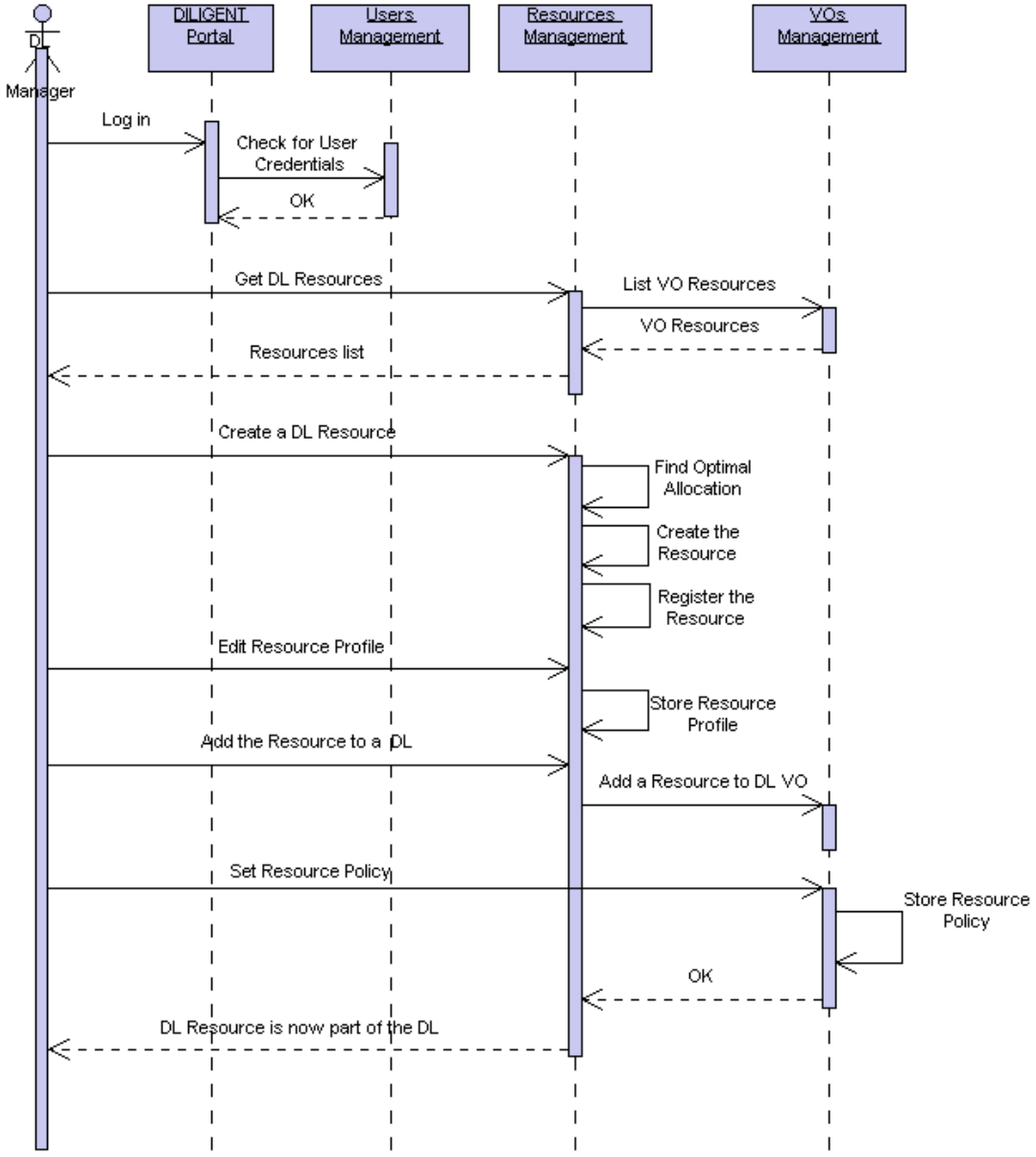

Figure 20: Create a DL Resource (sequence diagram)

See also Figure 17: Resources Management.

Grid Exploitation

This task will take many advantages by using Grid technologies. As previously stated, the creation of a new resource may asks for an hosting node that will be gathered by the infrastructure, as well as for a storage element that again is gathered by the infrastructure.

Mapping between functionalities and DILIGENT services (system integration)

The two services involved into this functionality are the Information Service and the Dynamic VO Support. The former is in charge to add the new resource to its knowledge base, the latter will prepare the environment allowing to use the resource in accordance with the sharing rules expressed by the DL manager. The Information Service is also in

charge to supply all the information needed to deploy the resource, e.g. identification of an available hosting node.

Use Stories

The DL Manager, monitoring the status of the resources forming the DL, discover that a new instance of a search service is needed in order to fulfil the new user requirements expressed by the DL Designer. As a consequence she/he select the search service appropriate for the DL, create a new instance of this service and rearrange the DL resource topology.

Testing issues

Creation of new DL resources is an important task. The goal of the testing activity must aim at verifying the correctness and completeness of this action. Moreover, the time needed to create a new resource instance is another important aspect to consider. As a consequence, a pool of representative resources among those the system is capable to instantiate must be selected and diverse infrastructure environment must be used to instantiate them.

Related non functional requirements

Due to the importance of this functionality, its reliability is a must, i.e. if a DL manager asks for a Resource creation then this resource must be effectively created. Another aspect to keep in mind is the scalability of this task, the response time must be as much as possible independent by the number of resources to create. Interoperability of this functionality is also important, where we intend that this functionality must be capable to instantiate the greater number of resource possible, e.g. it must be capable to instantiate a search service as well as an index service or a repository. Finally, security is another mandatory requirement, only authorized users must be entitled to execute this kind of functionality.

4.3.12 Find Optimal Allocation

Description and priority

The identification of an optimal allocation for each resource is the foundation to improve the reliability of the DILIGENT infrastructure and to mitigate faults during new DL deployment and management. During the detailed design different tasks will be designed to accommodate different kind of resources. The common goal of each task will be to acquire a deeper understanding of the potential and actual allocation weaknesses with the intent of enhancing the reliability of the system by removing the cause of the weakness or mitigating its effects. The outcome from each task is not expected to be an absolute optimal allocation because all tasks must take into account the DL set of resources already allocated and the same allocation weaknesses may be overcome through a number of choices. Moreover, the opportunity to enhance the system reliability is traded off against the costs of a distributed management.

User Requirements Fulfilled

This functionality is related with the model underlying the DILIGENT DL creation. We figure out that resources can be shared as well as created on demand and that this second aspect must be executed with the goal to promote an optimal allocation and use of resources. This task cover the latter aspect, i.e. it is in charge for identifying the most appropriate hosting node needed to create a new resource instance.

Numbers

See Sections 4.3.11 Create a DL Resource.

Constraints and Assumptions

This functionality assumes that a pool of hosting nodes will be available within the DILIGENT infrastructure, as well as that an appropriate description of them and of the resources that can be instantiated will be available in order to support the matchmaking mechanism.

UML Diagrams

See Figure 20: Create a DL Resource.

Grid Exploitation

See Section 4.3.11 Create a DL Resource.

Mapping between functionalities and DILIGENT services (system integration)

The main service involved into this task is the Broker & MatchMaker. This service will implement the matchmaking algorithm making use of resource descriptions acquired via the Information Service.

Use Stories

See Section 4.3.11 Create a DL Resource.

Testing issues

The goal is to test the correctness, completeness and performance of the matchmaking algorithm. With correctness we mean that the algorithm will identify one of the best hosting nodes among those available, with completeness we mean that the algorithm will perform a search among all the resource available, and finally, performance mean that this 'search' task will be executed in a reasonable amount of time. To test these aspects an environment containing different hosting nodes, i.e. hosting nodes with diverse characteristics, will be prepared. This environment can be also a virtual environment, i.e. realized just with the description of the hosting nodes. Against this environment must be executed the requests for identifying the hosting nodes for different kind of resources having different needs.

Related non functional requirements

See Section 4.3.11 Create a DL Resource.

4.3.13 Configure Resource

Description and priority

When a new resource is created, it will be possible to personalize its behaviour. The configuration parameters and their amount is clearly a resource specific characteristic. However, the choice of the resources forming the DL and their configuration aims at fulfilling the DL definition criteria specified by the DL designer.

User Requirements Fulfilled

This functionality represents the back end part of the user requirements fulfilled by the Define Configuration defined in Section 4.2.4. Moreover, the configuration of a service can be also due to a side effect of expressed user requirements or a need of operating the DL.

Numbers

This functionality will be executed in accordance with the Create a DL Resource in Section 4.3.11. Moreover, it depends by the number of customizable resources each DL will be equipped with.

Constraints and Assumptions

The customizability of a resource, i.e. a resource must be as much as possible customizable in order to promote its usage in different contexts. Moreover these configurations parameters must be available, e.g. included within the resource descriptive data.

UML Diagrams

See Figure 20: Create a DL Resource.

Grid Exploitation

The exploitation of grid technologies does not influence directly this functionality. However, if a resource has been designed for being deployed on a grid infrastructure its configuration parameters can be dependent by the usage this resource plan to do about other shared resources. For instance, a configuration parameter for a resource representing a computer intensive task can be the number of parallel jobs this task can be split in.

Mapping between functionalities and DILIGENT services (system integration)

The configuration of a resource depends by the resource itself. However, the service in charge to create a resource instance is the Keeper service and it is also in charge to specify the configuration parameters.

Use Stories

See Section 4.3.11 Create a DL Resource.

Testing issues

See Section 4.3.11 Create a DL Resource.

Related non functional requirements

See Section 4.3.11 Create a DL Resource.

4.3.14 Update a DL Resource

Description and priority

The DILIGENT DLs are dynamic and changeable in any time in order to be capable to follow the changeable requirements of the user communities. In particular, DL resources can be updated to modify their sharing rules: new users can be authorized to perform actions on the resource; a more restrictive or permissive set of policy can be applied to other users; new roles can be used to define the policy of the resources; etc.

User Requirements Fulfilled

This functionality contributes to fulfill the requirements related with the update of a DL (4.2.23 Update a DL). Moreover, update of a resource can be also executed by the Keeper service in response to the presence of a new resource instance that fits better then the previous one with the DL needs.

Numbers

The execution of this functionality will depends by the number of DLs as well as by their size and by the dynamicity of the DL resources.

Constraints and Assumptions

It assumes the presence of a versioning mechanism, i.e. a mechanism allowing identifying and comparing different version or instance of the same resource.

UML Diagrams

See Figure 17: Resources Management.

Grid Exploitation

This functionality aims at promoting an optimal usage of available resources, where optimal is related with the usage of the best pool of resources to support a functionality. For instance, if a new and powerful computing element will be available to the DL community then the services forming the DL will be capable to reply in faster fashion.

Mapping between functionalities and DILIGENT services (system integration)

This task will be executed by the DL Manager, in order to fulfil a new user needs, as well as by Keeper Service, in order to promote an efficient usage of available resources.

Use Stories

A DL fulfilling user requirements have been created making use of a search service capable to identify similar images with a certain level of similarity. During the DL lifetime, the provider of the similarity search service produce a new version of it that is capable to identify similar images with a more precise algorithm. The provider releases this new version of the service and registers it also to the DILIGENT infrastructure. The system identifies this new version and as it matches better with the DL definition criteria, automatically updates the DL resources making this new functionality available to the DL community.

Testing issues

Similar to those proposed in Section 4.3.11 Create a DL Resource.

Related non functional requirements

See Section 4.3.11 Create a DL Resource.

4.3.15 Remove a DL Resource

Description and priority

DL resources can be removed through a specific human–based request or can be automatically dropped out for a scheduled life-time management request or because they appear unstable or not correctly managed with respect to the rules established by the DL specification. Removing a resource from a DL implies also the automatic removal of the resource from the DL VO.

User Requirements Fulfilled

This functionality contributes to the requirements fulfilled in Section 4.2.25 Remove DL Resources.

Numbers

It is executed in accordance with the number of times the Remove DL Resources task is executed and to the size of each DL.

UML Diagrams

See Figure 17: Resources Management.

Grid Exploitation

The exploitation of the Grid is a direct consequence of the exploitation of it during the resource creation phase. For instance, if the resource make exclusive use of shared resource like a computing element or a storage element, when the resource is removed these grid resources must be make available to the infrastructure.

Mapping between functionalities and DILIGENT services (system integration)

Services involved are the Information Service, which must remove this resource from its resources knowledge base, the Dynamic VO Support service, which must remove the resource from all the trusting environments the resource is known in.

Use Stories

See 4.2.25 Remove DL Resources.

Testing issues

This task can be tested in conjunction with the testing of Remove DL Resources functionality, in particular the same testing environment could be used.

Related non functional requirements

The execution of this task must be done in a secure and reliable fashion, i.e. only authorized users are entitled to remove a DL resource and if this task is executed the resource must be effectively removed from the infrastructure.

4.3.16 Search Available Resources

Description and priority

Search Available Resources allows to identify resources with desired properties. The domain of the search is automatically restricted to the VO pertinent to the actor that activates this functionality. The query language supported will be defined in the design phase.

User Requirements Fulfilled

This functionality covers a lot of user requirements expressed both by the ARTE and ImpECt communities. These communities expressed a mechanism allowing them identify the resources that can be used within a DL or, in a more general sense, to be aware of the resources the infrastructure offer (ARTE_ucd01 'Select Services' and 'Search archives', ImpECt_ucd01 'Search/Browse Diligent').

Numbers

This functionality is executed in many contexts and as a consequence will be 'called' many times.

Constraints and Assumptions

This functionality will be capable to interpret and use the diverse trusting environments managed by the Dynamic VO Support service, as a consequence it must be context aware. For instance, if it is executed within a DL then those forming the DL will compose the pool of available resources. Moreover, it must be capable to understand and manage resources descriptive data.

UML Diagrams

This task will be executed in different contexts, see for instance Figure 13: Define a DL.

Grid Exploitation

The exploitation of grid technologies is related with the distributed discovery of them. Usually, grid environments are equipped with a collector of data (the information service) that acquires data about the available resources. When a user asks to this collector for available resources, it replies with the list of those the user is entitled to have access to.

Mapping between functionalities and DILIGENT services (system integration)

This functionality is covered by the information Service and by the Dynamic VO Support service. The former, with the help of the latter, will be capable to supply the most updated list of resources a user is entitled to have access to within an operational context, e.g. a DL,

a VO. Moreover, this functionality will be accessible in a user-friendly fashion, the Portal is in charge to host the presentation aspect of the functionality.

Use Stories

This functionality is used in different contexts, for instance see 4.2.1 Define a DL.

Testing issues

This functionality will be used in different contexts; in order to introduce errors due to its eventual imperfect implementation, a testing environment must be realized. Within this simulation environment diverse virtual resources will be registered and diverse operational environment will be configured to have access to different resources. Then a set of search activities will be executed with different goal and within different contexts in order to verify the correctness and completeness of the functionality.

Related non functional requirements

The reliability of the functionality is the most important issue, the resources identified must be effectively available. Its scalability is also a critical aspects, the system must be capable to identify the resources in a reasonable time that will be as much as possible independent by the number of resources the infrastructure hosts and by their distribution, i.e. their physical location.

4.3.17 Get Available Resources

Description and priority

This functionality factorizes out a common action needed when a search for resource is done, i.e. the gathering of the list of available resources.

User Requirements Fulfilled

This functionality does not fulfil any particular requirement but is used in many other contexts.

Numbers

Due to the generality of this task, it will be executed many times.

Constraints and Assumptions

This functionality will be capable to interpret and use the diverse trusting environments managed by the Dynamic VO Support service, as a consequence it must be context aware.

UML Diagrams

This task will be executed in different contexts, see for instance Figure 13: Define a DL.

Grid Exploitation

See 4.3.16 Search Available Resources.

Mapping between functionalities and DILIGENT services (system integration)

This functionality is covered by the information Service and by the Dynamic VO Support service. The former, with the help of the latter, will be capable to supply the most updated list of resources a user is entitled to have access to within an operational context, e.g. a DL, a VO. Moreover, this functionality will be accessible in a user-friendly manner, the Portal is in charge to host the presentation aspect of this functionality.

Use Stories

See 4.3.16 Search Available Resources.

Testing issues

See 4.3.16 Search Available Resources.

Related non functional requirements

See 4.3.16 Search Available Resources.

4.3.18 Browse Available Resources

Description and priority

Browse Available Resources allows to access to a list of resources described with the specified properties. The result is automatically restricted to the VO pertinent to the actor that activates this functionality. The list of presented properties will be defined in the design phase.

User Requirements Fulfilled

See 4.3.16 Search Available Resources.

Numbers

See 4.3.16 Search Available Resources.

Constraints and Assumptions

See 4.3.16 Search Available Resources.

UML Diagrams

This functionality has many aspects related with the Search Available Resources, the difference is related with the selection mechanism. No further diagrams are needed.

Grid Exploitation

See 4.3.16 Search Available Resources.

Mapping between functionalities and DILIGENT services (system integration)

See 4.3.16 Search Available Resources.

Use Stories

This functionality can be executed in all the contexts where a user driven resource discovery mechanism is needed.

Testing issues

This functionality will be used in different contexts; in order to introduce errors due to its eventual imperfect implementation, a testing environment must be realized. Within this simulation environment diverse virtual resources will be registered and diverse operational environment will be configured to have access to different resources. Then a set of browsing activities will be executed with different goal and within different contexts in order to verify the correctness and completeness of the functionality.

Related non functional requirements

First of all usability of this task is a mandatory requirements, users must be enabled to have access to this discovery functionality with the most user-friendly mechanism. Reliability of the functionality is also important, the resources identified must be effectively available. Its scalability is another critical aspects, the system must be capable to identify the resources in a reasonable time that will be as much as possible independent by the number of resources the infrastructure hosts and by their distribution, i.e. their physical location.

4.3.19 Get Resource Status

Description and priority

This functionality allows users to access to the information about the status of a resource. It implements the user interface of a monitoring functionality.

User Requirements Fulfilled

DILIGENT, and in particular its DLs, will be a distributed environment composed by a distributed pool of resources. This monitoring activity is a mandatory functionality in such kind of systems. Even if it is not a proper user requirement, the ImPECt community have explicitly expressed it, see ImpECt_ucd001 'Monitor Diligent Resources'.

Numbers

This functionality can be executed in any time for the whole lifetime of the resource.

Constraints and Assumptions

The main assumption is related with the capability to acquire information about the status of various kinds of resources. Push modalities, i.e. the information are acquired on demand by issuing a sort of query against the resource, as well as pull modalities, i.e. the resource notifies registered subscribers about changes in its status, can be implemented.

Grid Exploitation

Grid environment are usually equipped with a sort of monitoring mechanism. The exploitation of these technologies is highly recommended within the DILIGENT project.

Mapping between functionalities and DILIGENT services (system integration)

The Information Service, i.e. the service in charge for supplying the 'map' about DILIGENT resource and their related information, is the service that will execute and support this task with the support of the Portal.

Use Stories

A Resource Manager having registered an archive within DILIGENT decides to have a look at the status of its resource. She/he logs in the DILIGENT system and via the administrative section of the Portal is enabled to visualize the current status of the archive.

Testing issues

See Section 4.3.20.

Related non functional requirements

Reliability and security are the two main requirements that this functionality must fulfill. In fact, in managing in a sound fashion a distributed pool of resources the system must firmly rely on the information about the status of these resources. Moreover, the usability of the functionality is also important, e.g. the DL Manager must be capable to easily identify the information about resources she/he is interested in.

4.3.20 Monitor a Resource

Description and priority

"Monitoring mechanisms are concerned with obtaining, distributing, indexing, archiving, and otherwise processing information about the configuration and state of resources. In some cases, the motivation for collecting this information is to enable discovery of resources; in other cases, it is to enable monitoring of their status."

Therefore in any distributed system and in particular in a Grid environment, where system spans multiple locations managed by different people and organizations where no one has detailed knowledge of all components, monitoring become crucial for the success of the system.

Monitoring allows us to detect and diagnose the many problems that can arise in such contexts. Two modalities, pull and push modalities, are normally supported and allow to be aware about changes in resource properties: the former is supported through a query

mechanism, the latter via a subscription mechanism that automatically sends a notification message to registered subscribers.

It is important to take into account that this functionality must be able to collect data from any arbitrary information source, whether XML-based or not, and that historical data must be stored to collect statistical information.

User Requirements Fulfilled

DILIGENT, and in particular its DLs, will be a distributed environment composed by a distributed pool of resources. This monitoring activity is a mandatory functionality in such kind of systems. Even if it is not a proper user requirement, the ImPECt community have explicitly expressed it, see ImpECt_ucd001 'Monitor Diligent Resources'.

Numbers

This functionality will be executed in a continuative fashion. For instance, the Information Service execute this kind of activity at predefined interval of time for the whole lifetime of the resource in order to maintain the information about the resource in line with the real status of the resource.

Constraints and Assumptions

The main assumption is related with the capability to acquire information about the status of various kinds of resources. Push modalities, i.e. the information are acquired on demand by issuing a sort of query against the resource, as well as pull modalities, i.e. the resource notifies registered subscribers about changes in its status, can be implemented.

UML Diagrams

This functionality can be utilized in different contexts, moreover it is considered as atomic at this level of specification. No further diagrams are needed.

Grid Exploitation

Grid environment are usually equipped with a sort of monitoring mechanism. The exploitation of these technologies is highly recommended within the DILIGENT project.

Mapping between functionalities and DILIGENT services (system integration)

The Information Service, i.e. the service in charge for supplying the 'map' about DILIGENT resource and their related information, is the service that will execute and support this task.

Use Stories

The Keeper service, after having created the set of resources forming the DL, subscribes to the Information Service the notification mechanisms allowing it to be notified about the status of DL resources. Each time a DL resource status change, the Information Service notifies the Keeper that can act in accordance to its maintenance procedure.

Testing issues

Testing this functionality means to create a set of resources having different characteristics and then investigating the different notification mechanisms. In order to create a notification, it must be possible to introduce resources changes within the testing environment. This testing activity must be direct to investigate i) the performance in terms of time required in discovering resource changes and ii) the completeness of the mechanisms, i.e. that all the changes introduced are identified and notified.

Related non functional requirements

Reliability and security are the two main requirements that this functionality must fulfill. In fact, in manage in a sound fashion a distributed pool of resources the system must firmly rely on the information about the status of these resources. Moreover, the scalability of this task is important in order to allow the system to react in a quasi real time to system

changes. Finally, the usability of the functionality is also important, e.g. the DL Manager must be capable to easily identify the information about resources she/he is interested in.

4.4 VOs Management

The DLs that DILIGENT plans to realize are logical entities dynamically created to satisfy a specific user community need, and whose resources are allocated and provisioned on-demand. In order to address sharing and access control issues within these logical entites, DILIGENT will exploit mechanisms, built on the concept of Virtual Organization (VO), that glues together users and resources through a set of rules.

A detailed description of the Virtual Organization model adopted in the DILIGENT project is included in the Basic Concepts Section.

This model is build around the concepts of Role, Action and Resource. The establishment of a controlled access environment relies on the definition of a set of relations:

<*Role*, *Action*, *Resource*>

where:

- A *Role* is a job function with associated semantics about the authority and responsibility conferred to the User or Service assigned to the Role
- An *Action* is a Resource-specific task

The above rule should be read as: "*a User with a Role X can perfrom an Action Y on Resource Z*" in the contest of a specific VO.

VO Management includes basic functionalities like creation (see 4.4.1), editing (see 4.4.11) and removal (see 4.4.12) of VOs.

Further functionalities are needed in order to manage the internal structure and status of individual VOs, e.g. the inclusion of users and resources in a VO, the establishement of user roles, the setting of roles permissions, etc.

Finally, other functionalities allow querying the status of the VO and the retrieval of information about entities: e.g. list the VOs in DILIGENT (see 4.4.15), list the users of a VO (see 4.4.16), and list the VO Resources a user is entitled to use (see 4.4.17).

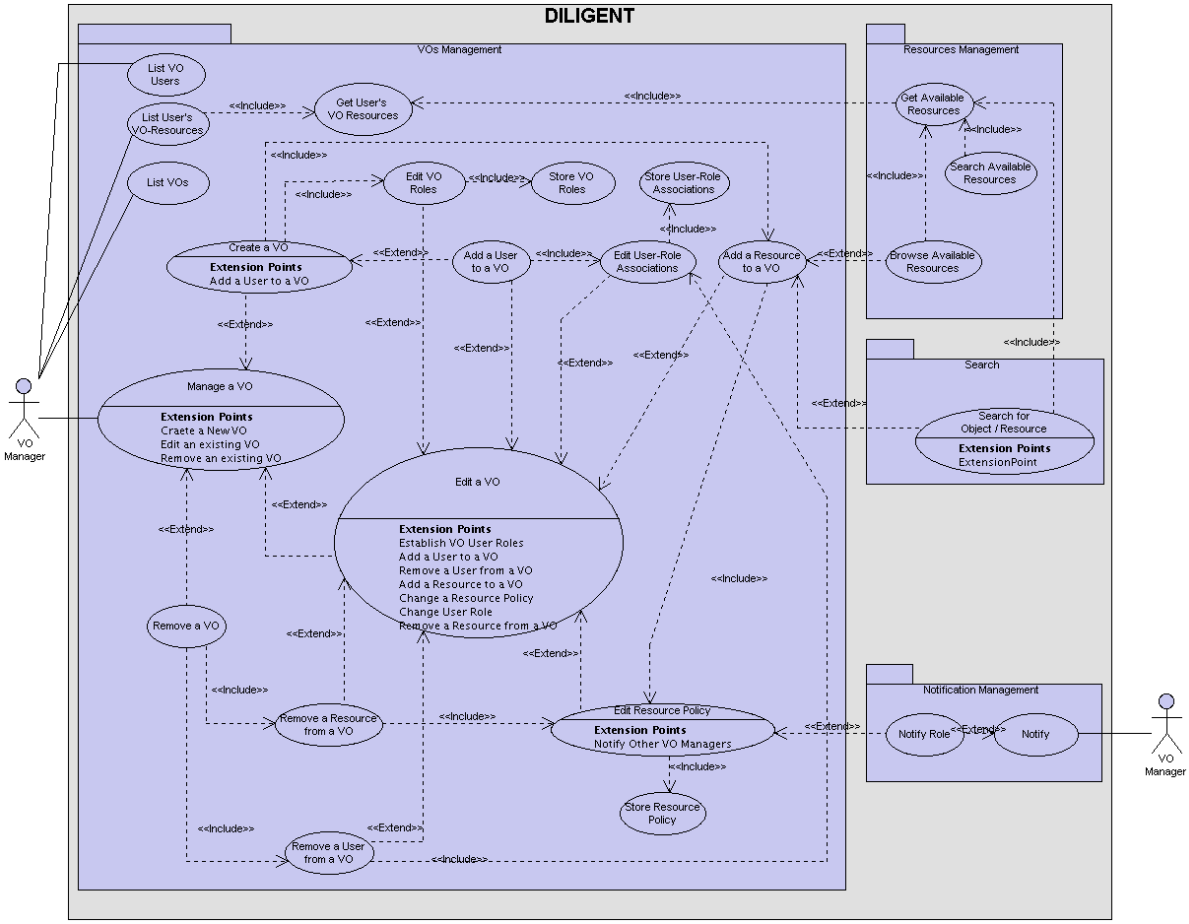

Figure 21: VOs Management (use case diagram)

4.4.1 Manage a VO

Description and priority

This functionality represents the main task of VO Management. This task deals with:

- The creation of a new VO (see 4.4.2);
- The editing of an existing VO (see 4.4.11);
- The removal of an existing VO (see 4.4.12).

User Requirements Fulfilled

This functionality factorizes out a number of other functionalities and represent the user access point to them.

Numbers

As it represents the management activities related to VOs, the execution frequency of this task depends on the number of VOs that will be created within DILIGENT. It should also be taken into account that the creation of a new DL implies the creation of a new VO.

Constraints and Assumptions

None.

UML Diagrams

As previously stated, this functionality gives just access to a pool of other functionality. See Figure 21: VOs Management. No further diagrams are needed.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

None

Use Stories

None

Testing issues

None

Related non functional requirements

None

4.4.2 Create a VO

Description and priority

This functionality represents the task of VO creation. This task deals with:

- The management of the initial set of VO roles (see 4.4.7);
- The inclusion of Users to the VO (see 4.4.6);
- The inclusion of Resources to the VO (see 4.4.3); the VO Manager can choose the pool of resources to be included among those s/he has access to.

User Requirements Fulfilled

None of the user communities has expressed requirements for the VO mechanism, at least in an explicit fashion. However both communities ask for a controlled sharing of resources. VOs have been introduced to satisfy this need.

Numbers

This functionality is not supposed to be a frequent action as it's strictly related to the creation of new user communities and the set-up of new DLs.

Constraints and Assumptions

Within grid environment already exists mechanisms for creating VOs, even if VO are perceived as quite static environment. Moreover, the DILIGENT VO concept is different from the Grid ones for other reasons, e.g. the resources forming a VO are different and of diverse types, the membership mechanism will be more flexible and easily, etc.

The assumptions underlying this functionality is the *capability to create VO on demand*, i.e. having more flexible mechanisms enabling the system to create new sharing environment in a dynamic fashion.

UML Diagrams

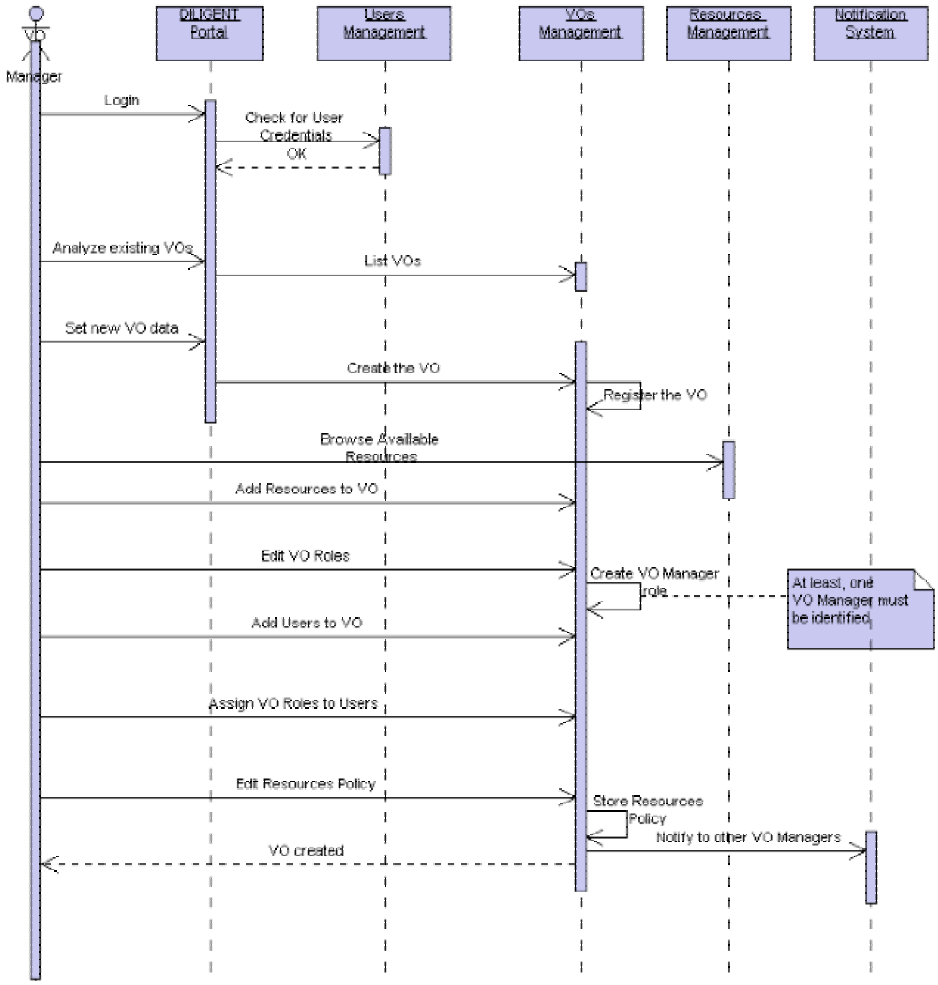

Figure 22: Create a VO (sequence diagram)

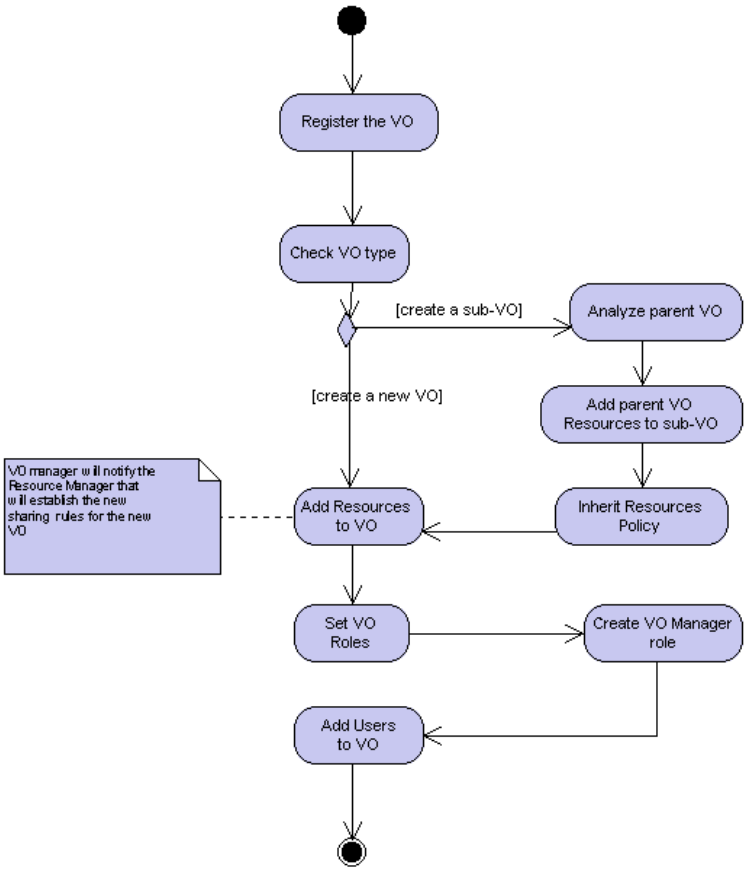

Figure 23: Create a VO (activity diagram)

See also Figure 21.

Grid Exploitation

As previously stated, within the grid community mechanisms capable to create trusted environment already exist even if they appear to be not so flexible as DILIGENT needs. DILIGENT plans to build on top of existing mechanisms to supply more flexible, user friendly dynamic ones.

Mapping between functionalities and DILIGENT services (system integration)

Dynamic VO Support Service is in charge to realize this task

Use Stories

This functionality represents one of the back end functionalities needed for creating a new DL. Another usage is related with the management of the user community.

The ARTE director receives a request from a group of researchers about the needs to share an archive to a controlled pool of users, a restrict subset of the ARTE users. In order to support this need, the ARTE VO Manager, i.e. the identified VO manager for the ARTE community, create a new VO representing this restricted set of users as a sub-VO of the ARTE VO. After having executed this activity, it will be possible to share the archive in a secure way just to this community.

Testing issues

Due to the relevance of this task, its testing activity must verify the correctness and completeness of it. Diverse VO, having diverse 'size' and kind of resources, must be created and then an attempt to access to these resources must be executed making use of respectively an authorized and an unauthorized identity.

Related non functional requirements

Reliability and security mechanisms are must given to the relevance of this task. Security means that only authorized users will be entitled to execute this activity. Reliability means that the creation of a VO must be executed effectively within the infrastructure creating a new trusted environment.

4.4.3 Add a Resource to a VO

Description and priority

This functionality models the action of inclusion of a resource in a VO, i.e. this resource will be added to the pool of resources accessible by VO users. As previously stated, the VOs are logical entities governed by policies, so this functionality aims also at setting new resource policies for the resource.

User Requirements Fulfilled

See 4.4.2 Create a VO.

Numbers

This functionality depends on the dynamicity of the VO, e.g. if a VO represents a DL and a new resource must be added to the DL this functionality will be executed.

Constraints and Assumptions

None.

UML Diagrams

See Figure 22: Create a VO.

Grid Exploitation

See 4.4.2 Create a VO.

Mapping between functionalities and DILIGENT services (system integration)

Dynamic VO Support Service is in charge to realize this task

Use Stories

See 4.4.2 Create a VO.

Testing issues

The testing environment is similar to those proposed for the Create a VO task.

Related non functional requirements

See 4.4.2 Create a VO.

4.4.4 Edit Resource Policy

Description and priority

This functionality covers the operations of add, update, and remove of access policies for a resource. It represents the interface to a policy knowledge base and via this interface authorized users are entitled to change the access rules for a given resource.

User Requirements Fulfilled

See 4.4.2 Create a VO.

Numbers

This functionality will be executed each time a resource is added to a VO, each time the access policies for a resource change within a VO, and each time a resource is removed from a VO.

Constraints and Assumptions

The existence of knowledge base about resource policies where this information is stored and retrievable.

UML Diagrams

At this level of detail no further diagrams are needed.

Grid Exploitation

This functionality may be considered a data intensive task. As a consequence a mechanism for realizing it as a distributed task must be taken into account. However, a mechanism for make secure these sensible data must be also considered.

Mapping between functionalities and DILIGENT services (system integration)

Dynamic VO Support Service is in charge to realize this task.

Use Stories

See 4.4.2 Create a VO.

Testing issues

This functionality represents the access to a repository of policy. As a consequence stress tests to verify the performance must be executed.

Related non functional requirements

Due to the importance of the information this task deal with, reliability and security are the two major requirements. Security deals with a secure access to these data, i.e. only authorized users must be capable to have access to and operate on these information, as well as preservation, i.e. data must be always available. Reliability means that the execution of the task must be performed correctly, a sort of transaction mechanism must be implemented in order to prevent concurrent access in modification to data as well as a rollback in order to remove uncompleted operation.

4.4.5 Store Resource Policy

Description and priority

This functionality represents the storage of a resource policy within the policy knowledge base maintained by the system.

User Requirements Fulfilled

See 4.4.4 Edit Resource Policy.

Numbers

See 4.4.4 Edit Resource Policy.

Constraints and Assumptions

See 4.4.4 Edit Resource Policy.

UML Diagrams

See 4.4.4 Edit Resource Policy.

Grid Exploitation

See 4.4.4 Edit Resource Policy.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support is in charge to supply this functionality.

Use Stories

See 4.4.4 Edit Resource Policy.

Testing issues

See 4.4.4 Edit Resource Policy.

Related non functional requirements

See 4.4.4 Edit Resource Policy.

4.4.6 Add a User to a VO

Description and priority

This functionality models the action of adjunction of an user to a VO, i.e. this user will be added to the pool of VO users. As previously stated, the VOs are logical entities governed by policies, so this functionality aims also at assigning a set of Roles for that user within the VO.

User Requirements Fulfilled

This functionality is an administrative task needed for managing VO concepts.

Numbers

This task will be executed each time it is needed to add a new user to a VO.

UML Diagrams

At this level of detail no further diagrams are needed.

Grid Exploitation

This functionality does not ask for any particular grid activity. However, this community has already studied these aspects and this functionality will be build as much as possible using existing mechanisms.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support is in charge to supply this functionality.

Use Stories

See 4.4.2 Create a VO.

Testing issues

Stress tests will be executed in order to verify scalability and response time of this task.

Related non functional requirements

See 4.4.2 Create a VO.

4.4.7 Edit VO Roles

Description and priority

This functionality models the management of the Roles in the VO. Actions covered by this functionality are the creation and removal of roles (including the management of the Role hierarchy), as well as the modification of permissions associated to a role. If involved role belongs to a hierarchy, the modifications will influence all the children roles in that hierarchy.

Numbers

The task will be executed many times.

Constraints and Assumptions

None.

UML Diagrams

Due to the level of details of these specifications no further diagrams are needed.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support is in charge to supply this functionality.

Use Stories

The VO manager needs to differentiate VO users in order to give them more fine-grained policy on VO resources. In order to be capable to manage a pool of users having more rights than others she/he creates a new role within the VO termed 'super user'. Then she/he assign the new role to all the users that needs these particular rights and changing the policies granted to the role it will be able to grant the rights to the appropriate users.

Testing issues

Stress tests must be executed in order to verify the scalability of this task.

Related non functional requirements

Security is a mandatory requirement when operations related with rights are executed.

4.4.8 Store VO Roles

Description and priority

This functionality represents the storage of a associations between a role and its permissions within a VO.

User Requirements Fulfilled

See 4.4.7 Edit VO Roles.

Numbers

See 4.4.7 Edit VO Roles.

Constraints and Assumptions

See 4.4.7 Edit VO Roles.

UML Diagrams

Due to the level of details of these specifications no further diagrams are needed.

Grid Exploitation

See 4.4.7 Edit VO Roles.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support is in charge to supply this functionality.

Use Stories

See 4.4.7 Edit VO Roles.

Testing issues

See 4.4.7 Edit VO Roles.

Related non functional requirements

See 4.4.7 Edit VO Roles.

4.4.9 Edit User-Role Associations

Description and priority

This functionality covers the operations of set/unset of associations between a user and its Roles within a VO. It represents the interface to the association knowledge base and via this interface will be possible to change the set of roles associated to a user.

User Requirements Fulfilled

This functionality is obtained factorizing out the activity of maintenance of the data needed to identify the association among users and roles within the VO.

Numbers

This functionality will be executed each time it's needed to change the associations between a user and its role within a VO.

UML Diagrams

Due to the level of details of these specifications no further diagrams are needed.

Grid Exploitation

The exploitation of grid technologies can be related with the maintenance of the data needed in a distributed fashion, i.e. making use of the distributed storage capacity offered by the grid infrastructure.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support is in charge to supply this functionality.

Use Stories

See 4.4.7 Edit VO Roles.

Testing issues

Stress tests will be executed in order to verify the scalability of this functionality.

Related non functional requirements

Reliability and security are the two major requirements. Security deals with a secure access to these data, i.e. only authorized users must be capable to have access to and operate on these information, as well as preservation, i.e. data must be always available. Reliability means that the execution of the task must be performed correctly, a sort of transaction mechanism must be implemented in order to prevent concurrent access in modification to data as well as a rollback in order to remove uncompleted operation.

4.4.10 Store User-Role Associations

Description and priority

This functionality represents the storage of an association between a user and its roles within a VO.

User Requirements Fulfilled

See 4.4.9 Edit User-Role Associations.

Numbers

See 4.4.9 Edit User-Role Associations.

Constraints and Assumptions

See 4.4.9 Edit User-Role Associations.

UML Diagrams

Due to the level of details of these specifications no further diagrams are needed.

Grid Exploitation

See 4.4.9 Edit User-Role Associations.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service will be in charge for supplying this functionality.

Use Stories

See 4.4.9 Edit User-Role Associations.

Testing issues

See 4.4.9 Edit User-Role Associations.

Related non functional requirements

See 4.4.9 Edit User-Role Associations.

4.4.11 Edit a VO

Description and priority

Updating a VO has different aspects, all covered by this macro functionality. In particular the following actions are modelled:

- Adding a Resource to those accessible within the VO;
- Removing a Resource from those accessible within the VO;
- Modifying the resource access policies within a VO
- Adding a User to the VO
- Removing a User from the VO
- Changing the User-Roles associations within a VO, i.e. entitle an user to have access to resources or deny user to access to resources previously usable;
- Adding a new Role to the VO in order to establish a new class of policies.

User Requirements Fulfilled

This functionality complete the management tasks required to operate with VOs.

Numbers

This functionality will be executed many times, each time the VO manager consider an update of the DL as needed.

UML Diagrams

See Figure 21: VOs Management.

Grid Exploitation

See the following more specific update functionalities.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic Support service will be in charge to cover this functionality.

Use Stories

See the more specific update functionalities.

Testing issues

More details are reported within the more specific update functionalities.

Related non functional requirements

See the related more specific update functionalities.

4.4.12 Remove a VO

Description and priority

This functionality models the removal of a VO executed by the VO manager. Associating users and resources composes a VO, removing a VO deals with the removal of the users having access to the VO (see 4.4.14 Remove a User from a VO) and the disposal of the resources belonging to it (see 4.4.13 Remove a Resource from a VO).

User Requirements Fulfilled

This functionality completes with the 'Create a VO' and 'Edit a VO' the set of VO management task.

Numbers

This functionality will be executed in accordance with the number of VOs created during the DILIGENT lifetime.

Constraints and Assumptions

The removal of a VO implies the removal of all its sub-VOs.

UML Diagrams

Due to level of detail used within this specification, no further diagrams are needed. See Figure 21: VOs Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to supply this functionality.

Use Stories

See 4.2.9 Dispose a DL.

Testing issues

This functionality can be tested together with the Dispose a DL functionality.

Related non functional requirements

Reliability is an important requirement. Security is also important in order to prevent inappropriate removal of needed VOs.

4.4.13 Remove a Resource from a VO

Description and priority

This functionality covers the removal of a Resource from a VO. As this operation aims also at deny to the VO users to have access to it, this functionality uses the Set Resource Policy functionality in order to unset all the policies previously defined on the resource.

User Requirements Fulfilled

This functionality factorizes out an aspect of the Remove a VO functionality.

Numbers

This task is executed each time a VO is removed and for each resource belonging to the VO.

UML Diagrams

Due to the level of details adopted within this specification no further diagrams are needed. See Figure 21: VOs Management.

Grid Exploitation

This task does not require for particular grid features.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to supply this task.

Use Stories

See 4.4.12 Remove a VO.

Testing issues

Stress tests will be executed to verify the removal mechanism.

Related non functional requirements

The security of this task is a must. Only authorized users must be capable to remove VO resources.

4.4.14 Remove a User from a VO

Description and priority

This functionality covers the removal of a User from a VO. It uses the Edit User Roles functionality in order to unset all the roles previously assigned to the user and so to unset all the grants on VO resources.

User Requirements Fulfilled

This functionality factorizes out an aspect of the Remove a VO functionality.

Numbers

This task is executed each time a VO is removed and for each user belonging to the VO.

UML Diagrams

Due to the level of details adopted within this specification no further diagrams are needed. See Figure 21: VOs Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to maintain all the data inherent to the VOs.

Use Stories

See 4.4.12 Remove a VO.

Testing issues

See 4.4.12 Remove a VO.

Related non functional requirements

See 4.4.12 Remove a VO.

4.4.15 List VOs

Description and priority

This functionality aims at make available a list of the VOs the DILIGENT systems manages.

User Requirements Fulfilled

This functionality completes with the 'Create a VO', 'Edit a VO' and 'Remove a VO' the set of VO management task.

Numbers

This functionality can be executed many times, for instance each time a manager wants to know the DILIGENT living VOs.

UML Diagrams

Due to the level of details adopted within this specification no further diagrams are needed. See Figure 21: VOs Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to maintain the needed data and to supply this functionality.

4.4.16 List VO Users

Description and priority

This functionality aims at supply the list of users that are entitled to have access to resources of a VO.

User Requirements Fulfilled

This functionality, together with the other list functionalities, contributes to supply the management functionality related with VOs.

Numbers

This functionality can be executed many times.

Constraints and Assumptions

The existence of a knowledge base containing the associations between the users and their associated VO/VOs.

UML Diagrams

Due to the level of details adopted within this specification no further diagrams are needed. See Figure 21: VOs Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to maintain these data and to supply this functionality.

Use Stories

During the canonical management activity the VO Manager needs to know the list of the users belonging to its VO. One of the parameters included within the information listed by the system is the role the user play within the VO. The VO manager decides then to assign another role to a user because its profiles match with the skill needed to play that role.

Testing issues

See 4.4.15 List VOs.

Related non functional requirements

See 4.4.15 List VOs.

4.4.17 List User's VO-Resources

Description and priority

This functionality aims at producing the list of VO resources a user have access to. This kind of functionality is particularly critical and used as the information it supplies is fundamental in order to establish the rights a user has.

User Requirements Fulfilled

This functionality, together with the other list functionalities, contributes to supply the management functionality related with VOs.

Numbers

This functionality can be executed many times. For instance it will be executed each time it is necessary to identify the pool of resources a user is entitled to have access to.

UML Diagrams

This functionality does not need per se of other diagrams. See Figure 21: VOs Management. However, it is used in different contexts, for instance see Figure 14: Create a DL.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to maintain these data and thus to supply this functionality.

Use Stories

This functionality is used in different contexts, for instance see 4.2.14 Create a DL.

Testing issues

See 4.4.15 List VOs.

Related non functional requirements

See 4.4.15 List VOs.

4.4.18 Get User's VO Resources

Description and priority

This functionality arise factorizing out a part of the task that is in charge to list the resources a user is entitled to have access within a VO, i.e. List User's VO-Resources. This functionality represents the back end of the other task that is responsible to show the data retrieved making use of the data acquired via this activity.

User Requirements Fulfilled

This functionality can be used in different contexts.

4.5 Users Management

The management of information about users and groups of users is mainly divided into four functional parts *users management*, *groups management*, *selection*, and *invitation* as reported into the Use Case Diagram in Figure 24: Users Management.

Users management part covers functionalities for adding and removing users as well as to edit user profiles and request user rights.

Groups management part includes use cases to create and remove group of users as well as to edit the group profile. The model underlying the adjunction of a user to a group imply that a user has been invited to be part of a group; then the invited user accept or reject the invite; finally, the User Manager include into the group the users that have accepted the invitation.

Selection part deals with search and browse of users and groups.

Invitation part deals with invitee users and groups to be part of a DL, to get access to a Collection, etc.

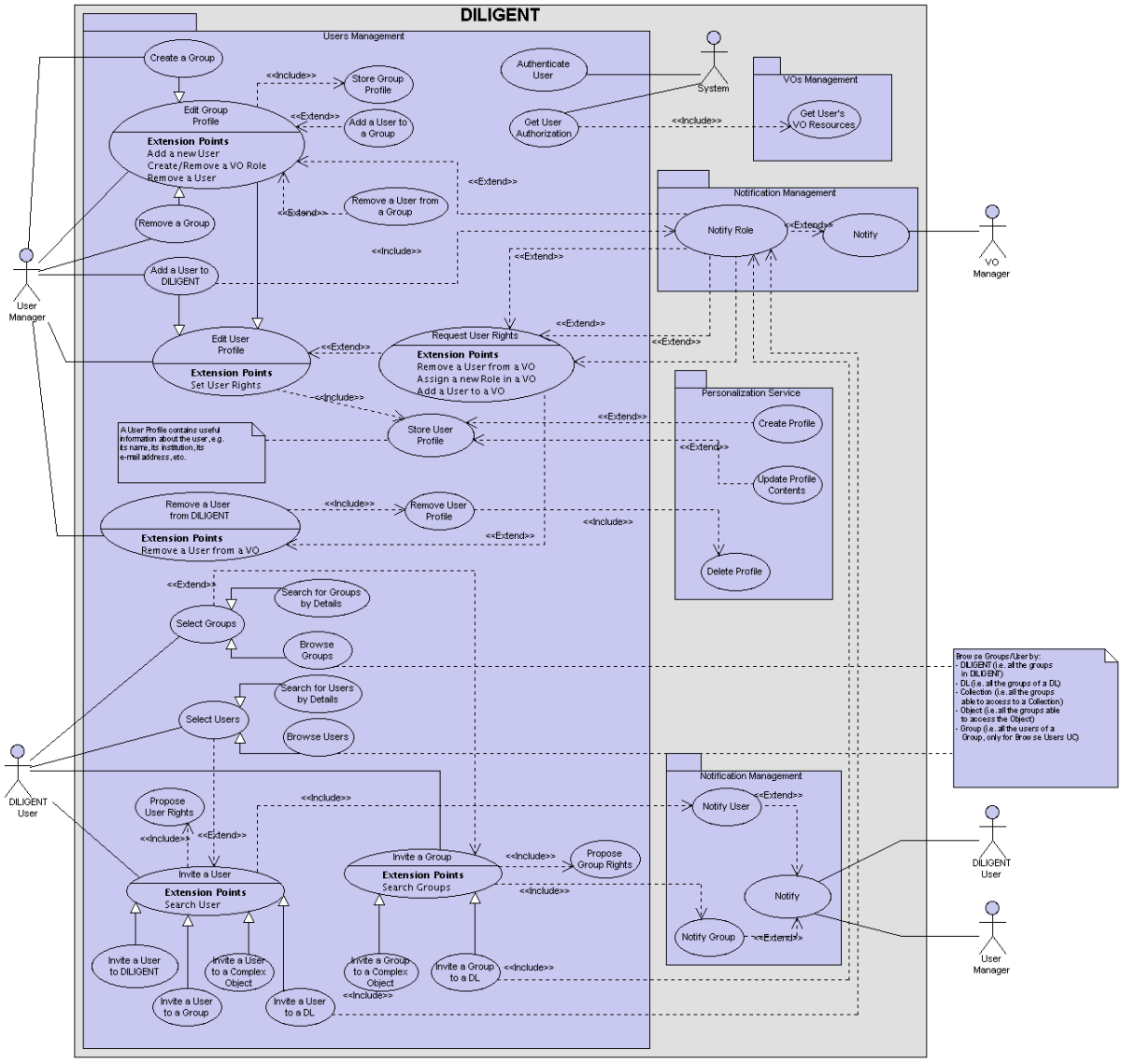

Figure 24: Users Management (use case diagram)

4.5.1 Create a Group

Description and priority

This functionality represents the operation allowing to create a new Group entity enabling thus to put together set of users and manage then as a single entity. This operation is performed by a *User Manager* actor and include editing of a "group profile", i.e. the set of group information like group name, group description, group contact user, group members, etc.

User Requirements Fulfilled

Even if user communities do not have expressed explicitly the need to have group management functionalities, we consider it an important concept that eases the users management task.

Numbers

This functionality can be executed many times, i.e. each time a group manager needs to create a new group.

Constraints and Assumptions

None.

UML Diagrams

Due to the level of detail exposed within this functional specification, no further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

This functionality will be covered by the Dynamic VO Support service while the Portal service will give access it.

Use Stories

The ARTE Group manager decides to create two groups: the student group and the teacher group. Users belonging to the first group will have less rights that users belonging to the second group. For instance, she/he will be capable to make a request to the VO Manager in order to grant students a permission to 'read' an archive and teachers a permission to 'read/write' the archive.

Testing issues

The classical testing activity related with an object creation must be executed.

Related non functional requirements

Security is an important issue. Only authorized users must execute the creation of a group. Usability of this functionality is also important. More easy to use the functionality will be, more error free usage of the functionality will be performed.

4.5.2 Edit Group Profile

Description and priority

Once a group has been created the *User Manager* can edit the group profile. The editing of a group profile means one of the following operations: adding a user to the group (see Section 4.5.4), removing a user from the group (see Section 4.5.5). These operations imply the storage of the group profile (see Section 4.5.3) and a potential modification of profiles of users belonging to the group (see Section 4.5.8).

User Requirements Fulfilled

See 4.5.1 Create a Group.

Numbers

See 4.5.1 Create a Group.

UML Diagrams

Due to the level of detail exposed within this functional specification, no further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to supply this functionality while the Portal service will give access it.

Use Stories

See 4.5.1 Create a Group.

Testing issues

See 4.5.1 Create a Group.

Related non functional requirements

See 4.5.1 Create a Group.

4.5.3 Store Group Profile

Description and priority

This functionality represents the storage of the group profile within group profiles knowledge base maintained by the system.

User Requirements Fulfilled

This functionality factorizes out the storage aspects related with the group information.

Numbers

This functionality will be executed each time an update on group data is performed.

Constraints and Assumptions

This functionality assumes that a storage area needed to maintain the user profiles will be available.

UML Diagrams

Due to the level of detail exposed within this functional specification, no further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

This functionality may use the storage area offered by the grid infrastructure, i.e. the group profiles may be stored in a distributed fashion on the grid.

Mapping between functionalities and DILIGENT services (system integration)

See 4.5.1 Create a Group.

Use Stories

See 4.5.1 Create a Group.

Testing issues

This functionality is a data intensive task Stress tests will be performed in order to verify the performance.

Related non functional requirements

Reliability of this functionality is an important requirement. The maintenance of these data is important for all the other group tasks. Moreover, security is another requirements. It is important that only authorise users will have access to this information.

4.5.4 Add a User to a Group

Description and priority

This functionality covers the adjunction of a user to a group. Only already registered DILIGENT users can be added to a group.

User Requirements Fulfilled

See 4.5.1 Create a Group.

Numbers

This functionality can be executed many times. This number will depend by the number of groups that will be created as well as by their size.

UML Diagrams

See 4.5.1 Create a Group.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

See 4.5.1 Create a Group.

Use Stories

See 4.5.1 Create a Group.

Testing issues

See 4.5.1 Create a Group.

Related non functional requirements

See 4.5.1 Create a Group.

4.5.5 Remove a User from a Group

Description and priority

Via this functionality, the User Manager is entitled to remove a user from a group.

User Requirements Fulfilled

See 4.5.1 Create a Group.

Numbers

This functionality can be executed many times. This number will depend by the number of groups that will be created as well as by their size.

UML Diagrams

Due to level of detail adopted within this document this task no further diagrams are needed.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

Dynamic VO Support service will supply this functionality with the support of the Portal for the presentation part.

Use Stories

See 4.5.1 Create a Group.

Testing issues

See 4.5.1 Create a Group.

Related non functional requirements

See 4.5.1 Create a Group.

4.5.6 Remove a Group

Description

Via this functionality, the Group Manager is entitled to remove a group.

4.5.7 Add a User to DILIGENT

Description and priority

This is one of the main UC about User Management, which deals with the addition of new user to the DILIGENT infrastructure. The User Manager adds a new user to DILIGENT by filling in the user data (user profile), i.e. name, contact address, telephones, home page, organization, etc. After having registered the user data, the User Manager can notify the DILIGENT VO Manager in order to add the user to the DILIGENT VO.

User Requirements Fulfilled

User requirements fulfilled: ImpECt_ucd03 'Create User' and ARTE_ucd08 'Add user'.

Numbers

Hopefully, this functionality will be executed many times.

Constraints and Assumptions

The physical person that asks for being registered within the system must supply all the data needed for identifying it.

UML Diagrams

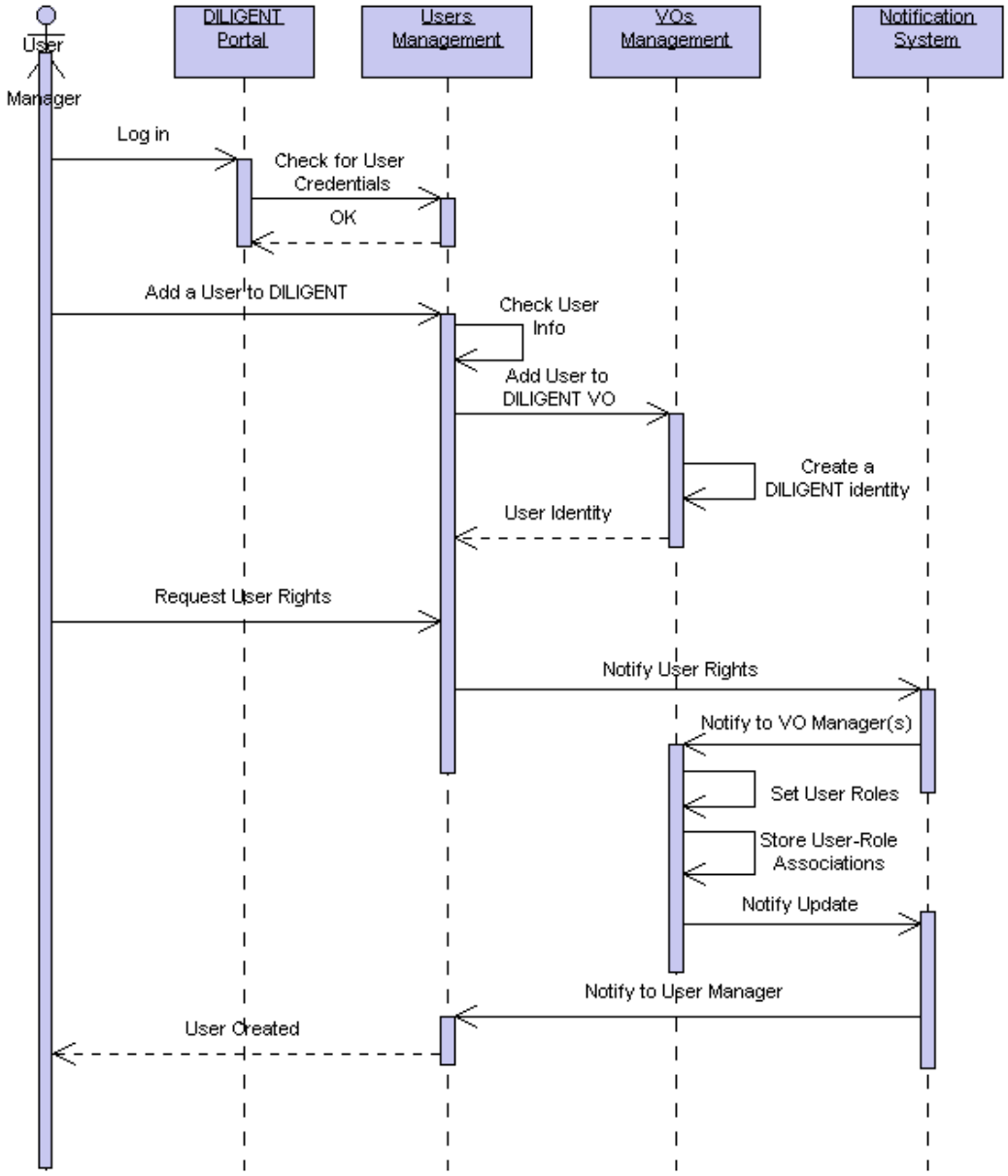

Figure 25: Add a User to DILIGENT (sequence diagram)

See also Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service will be in charge to supply this functionality with the support of the Portal for the presentation part.

Use Stories

A new researcher, Mr. Black, joins the ESA research group in Frascati - Rome. Mr. Jones, one of the ESA people involved within the same project as Mr. Black, notifies it about the presence of a DL supporting the project and then invites it to take part in this environment. At the same time, a notification of this invitation activity is sent to the ImpECt User manager. When Mr. Black accept the invitation, the user manager add a new user to the DILIGENT system and from now on the researcher is entitled to have access to the DL and

enjoy with it. During the DL lifetime, the ImpECt user manager identifies in Mr. Black the people identified by the ImpECt community to become a user manager. In order to give him this grant, the manager updates the set of roles associated with Mr. Black

Testing issues

This functionality will be subject to the testing activities needed to perform this classical operation. Particular attention must be posed on the security aspect of this functionality.

Related non functional requirements

Security is a mandatory requirement to fulfil. Only authorized users must be entitled to add new users to the system.

4.5.8 Edit User Profile

Description and priority

Each DILIGENT user has an associated profile. This profile contains useful information about the user, e.g. its name, its e-mail address, its institution, etc. Performing this use case, the User Manager has access to these data and is enabled to edit them. In particular, via this functionality the User manager can send a request to the VO Manager in order to grant new rights to the user.

User Requirements Fulfilled

This functionality fulfils the requirement described in the ARTE_ucd08 'Edit user profile'. However, due the generality of this activity it can be considered as a general user requirement.

Numbers

This functionality is potentially executable many times.

UML Diagrams

This functionality is well known; No further requirements are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the help of the Portal, will support this functionality.

Use Stories

See 4.5.7 Add a User to DILIGENT.

Testing issues

This functionality must be tested in accordance to its semantic, i.e. it represents an update task and the relative test must be executed. For instance, stress tests can be executed in order to verify the correct behaviour in case of concurrent access in modification.

Related non functional requirements

Security is clearly a must. User profiles maintains data about the rights a certain user has, as a consequence only authorized users must be entitled to modify them.

4.5.9 Request User Rights

Description and priority

This functionality enables the User manager to request more or few privileges for a user over a resource. For each VO the resource is registered in, the corresponding VO Manager is notified. He's in charge of performing the actual grant by giving the user the suitable roles.

User Requirements Fulfilled

This functionality represents a classical task in user management system.

Numbers

This functionality can be executed many times.

UML Diagrams

This functionality is used in different contexts, e.g. see Figure 25: Add a User to DILIGENT.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The main service involved into this activity is the Dynamic VO Support service that will act with the support of the Portal.

Use Stories

See 4.5.7 Add a User to DILIGENT.

Testing issues

The goal of the testing activity is to verify the performance and correctness of this task. Stress tests will be executed to measure the performance of this task, while controlled test must be set up in order to verify its correctness.

4.5.10 Store User Profile

Description and priority

This functionality models the storage of the user profile within the users knowledge base maintained by the system.

User Requirements Fulfilled

This functionality derives from test-bed community requirements related with user management, i.e. ARTE_ucd08 and ImpECt_ucd03.

Numbers

This functionality will be executed each time a request to add a new profile or update an existing one will be performed.

Constraints and Assumptions

A storage area to maintain user profiles is available.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to supply this functionality.

4.5.11 Remove User Profile

Description and priority

This functionality models the removal of the user profile from the users knowledge base maintained by the system.

User Requirements Fulfilled

This functionality derives from test-bed community requirements related with user management, i.e. ARTE_ucd08 and ImpECt_ucd03.

Numbers

This functionality will be executed each time a request to remove a user profile will be performed.

Constraints and Assumptions

The user exists and its profile is stored within the system.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to supply this functionality.

4.5.12 Remove a User from DILIGENT

Description and priority

The User Manager could decide to completely remove a user from the DILIGENT Infrastructure. The removal of a user implies the removal of its profile and the removal of the user from all the VO s/he has access to. This will be achieved notifying the various VO managers.

User Requirements Fulfilled

This functionality fulfils the two requirements from test bed user communities, the ImpECt_ucd03 'Remove User' use case and the ARTE_ucd08 'Remove User' use case.

Numbers

This functionality will be used each time a DILIGENT user must be removed from the system.

Constraints and Assumptions

The user is already registered and known to the system.

UML Diagrams

No further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service is in charge to supply this functionality.

Use Stories

At the end of a course organized by the ARTE community, the ARTE User Manager must be capable to remove all the students of this course. In order to perform this activity, he selects the whole set of users and then asks the VO Manager to remove them, if they are 'only' student of that course.

Testing issues

A set of dummy users must be created and then removed. Stress test on this functionality can be executed to measure the responsiveness of the system.

Related non functional requirements

Security is an important requirement in order to prevent inappropriate removal of users. Reliability is another requirement, if an authorized user decides to remove a user X from the

system then this activity must effectively be executed, i.e. from now on X does not be capable to log in the system.

4.5.13 Select Groups

Description and priority

This functionality covers the operations a user can perform in order to discover groups of users. For example, this functionality is used to identify already existing DILIGENT groups in order to invite them to collaborate or have access to a complex object. Two kinds of retrieval operations have been idengified in by analyzing the user requirements: the search of groups using data stored into the profile (see 4.5.14) and the browse of the list of groups (see 4.5.15).

User Requirements Fulfilled

The concept of group has been introduced in order to simplify the management of users

Numbers

This task will be executed each time a user wants to select one or more group for its purposes. For instance, if a DL designer needs to identify groups of users that are interested in the DL she/he is designing, performing this activity he/she will perform this selection in a user-friendly fashion.

UML Diagrams

No further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal for the presentation part, will supply this functionality.

Use Stories

One of the ARTE DL Designers is in charge to create a DL to support a workshop having a certain topic. In order to promote this workshop to the broadest audience possible, he/she performs a research about users and groups of users that have expressed the topic of the workshop or a similar one among those they are interested in. Having a look at the profiles of these users and these groups he/she will be able to select the subset of them that are deemed as more close to the workshop topic and audience. Finally, the designer invites them to participate to the DL in order to give them access to the information about the event.

Testing issues

Various dummy groups will be created with different characteristics. A set of selection operation must be performed against these dummy groups in order to verify the correctness and performance of the underlying algorithm.

Related non functional requirements

The usability of this task is the most important requirement. This process must be as much as possible intuitive and user friendly.

4.5.14 Search for Groups by Details

Description and priority

This functionality covers the search operations a users can perform in order to discover groups of users. This search is based on data stored into the profile. For example, it will be possible to retrieve all the DILIGENT groups that have expressed as a research field the term 'marine pollution' or to retrieve all the DILIGENT groups that have more than 100 user.

User Requirements Fulfilled

See 4.5.13 Select Groups.

Numbers

See 4.5.13 Select Groups.

UML Diagrams

See 4.5.13 Select Groups.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.13 Select Groups.

Testing issues

See 4.5.13 Select Groups.

Related non functional requirements

See 4.5.13 Select Groups.

4.5.15 Browse Groups

Description and priority

This functionality covers the browsing operations a users can perform in order to discover groups of users. Various granularity of browsing will be enabled by the system, e.g. the browsing of all the DILIGENT groups, the browsing of the group in a DL, the browsing of the group capable to access a certain collection.

User Requirements Fulfilled

See 4.5.13 Select Groups.

Numbers

See 4.5.13 Select Groups.

UML Diagrams

See 4.5.13 Select Groups.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.13 Select Groups.

Testing issues

See 4.5.13 Select Groups.

Related non functional requirements

See 4.5.13 Select Groups.

4.5.16 Select Users

Description and priority

This functionality covers the operations a users can perform in order to discover other DILIGENT users. For example, this feature is used to identify already existing DILIGENT users in order to invite them to collaborate or have access to a complex object, or to invite them in a DL as they have interests matching the DL content.

Analyzing the user requirements two kinds of retrieval operations have been identified: the search of users by using the data stored into the profile and the browse of the list of users.

User Requirements Fulfilled

This functionality fulfils the ARTE_ucd08 Select Users and ImpECt_ucd03 Search users.

Numbers

This functionality will be executed each time a user needs to select end discover other DILIGENT users. For instance, see also 4.5.13 Select Groups.

Constraints and Assumptions

The system maintains information about users, i.e. user profiles.

UML Diagrams

This functionality is quite simple, no further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.13 Select Groups.

Testing issues

Various dummy users will be created with different profiles. A set of selection operation must be performed against these dummy users in order to verify the correctness and performance of the underlying algorithm.

Related non functional requirements

The usability of this task is the most important requirement. This process must be as much as possible intuitive and user friendly.

4.5.17 Search for Users by Details

Description and priority

This functionality covers the search operations a users can perform in order to discover other DILIGENT users. This search is based on data stored in the profile. For example, it will be possible to retrieve all the DILIGENT users that have expressed as a research field the term 'marine pollution' as well as all the DILIGENT users whose mother tongue is the Italian.

User Requirements Fulfilled

This functionality matches the ARTE_ucd08 'Search by User Properties' as well as the ImpECt_ucd03 'Search Users' UCs.

Numbers

See 4.5.16 Select Users.

UML Diagrams

See 4.5.16 Select Users.

Grid Exploitation

See 4.5.16 Select Users.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.16 Select Users.

Testing issues

See 4.5.16 Select Users.

Related non functional requirements

See 4.5.16 Select Users.

4.5.18 Browse Users

Description and priority

This functionality covers the browsing operations a user can perform in order to discover other DILIGENT users. Various granularity of browsing will be enabled by the system, e.g. the browsing of all the DILIGENT users, the browsing of the users in a DL, the browsing of the user capable to access a certain collection.

User Requirements Fulfilled

See 4.5.16 Select Users.

Numbers

See 4.5.16 Select Users.

Constraints and Assumptions

UML Diagrams

See 4.5.16 Select Users.

Grid Exploitation

See 4.5.16 Select Users.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.16 Select Users.

Testing issues

See 4.5.16 Select Users.

Related non functional requirements

See 4.5.16 Select Users.

4.5.19 Invite a User

Description and priority

A DILIGENT user may want to perform an invitation to another user to be member of a Group or of a DL or to have access to a Collection, a Complex Object, etc. In order to do that the DILIGENT user discovers and selects the user to invite (see 4.5.16) and proposes the rights he/she will have on this new item/environment. At the end of this process both the invited users and the User Manager receive a notification about the operation. Invited users can accept or reject the invitation and User Manager, analyzing the users reply, will performs required operations to update the user profile. Finally, the User Manager notifies the VO Managers that, in order to make effective these rights, will transform them in terms of appropriate roles.

User Requirements Fulfilled

In order to simplify and promote the usage of various products of the DILIGENT infrastructure, e.g. DLs, Collections, Groups, and in accordance with the two user communities this invitation mechanism has been identified.

Constraints and Assumptions

For each user the system maintains all the information needed to notify her/him, e.g. the email address.

UML Diagrams

See Figure 13: Define a DL in section 4.2.1 Define a DL.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.16 Select Users.

Testing issues

A set of dummy users with a real email address must be created. Various invitations with different user rights must be sent to these users in order to test the performance and correctness of this mechanism.

Related non functional requirements

Usability is the most important requirement for this functionality.

4.5.20 Propose User Rights

Description and priority

The operation of invitation of a user to an item/environment implies the proposal of rights the user will have on this new item/environment. This functionality covers the latter activity. The DILIGENT system is in charge to establish the appropriate relationships between users and roles in order to accommodate the proposed set of rights.

User Requirements Fulfilled

This functionality contributes to complete the user requirements fulfilled by the Invite a User functionality.

Numbers

This functionality will be executed each time an invitation to a user is performed by the Invite a User task.

Constraints and Assumptions

A user-friendly language enabling to express user rights will be available.

UML Diagrams

No further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.19 Invite a User.

Testing issues

See 4.5.19 Invite a User.

Related non functional requirements

See 4.5.19 Invite a User.

4.5.21 Invite a User to a DL

Description and priority

This functionality is a specialization of Invite a User (see 4.5.19) and deals with the making of a proposal to a user to join a DL. The invitation implies that a notification message is sent to the user, to the User Manager and to the VO Manager. Invited user may accept or reject. According to the user decision, the User Manager updates the user profile and the VO Manager gives him suitable roles.

User Requirements Fulfilled

See 4.5.19 Invite a User.

Numbers

This functionality will be executed each time a DILIGENT user needs to invite another one to participate in a DILIGENT DL.

UML Diagrams

See 4.5.19 Invite a User.

Grid Exploitation

See 4.5.19 Invite a User.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.2.1 Define a DL.

Testing issues

A set of dummy users with a real email address must be created. A pool of dummy DL will be also created. Various invitations with different user rights to different DLs must be sent to these users in order to test the performance and correctness of the mechanism.

Related non functional requirements

See 4.5.19 Invite a User.

4.5.22 Invite a User to a Group

Description and priority

This functionality is a specialization of Invite a User (see 4.5.19) and deals with the making of a proposal to a user to be part of a Group. The invitation implies that a notification message is sent to the user and to the User Manager. Invited user may accept or reject. According to the user decision, the User Manager updates the user profile and the VO Manager gives him suitable roles.

User Requirements Fulfilled

See 4.5.19 Invite a User.

Numbers

This functionality will be executed each time a DILIGENT user needs to invite another one to participate in a DILIGENT group.

Constraints and Assumptions

The user is already registered within the system, the group exists or it is in the definition phase.

UML Diagrams

See 4.5.19 Invite a User.

Grid Exploitation

See 4.5.19 Invite a User.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.19 Invite a User.

Testing issues

A set of dummy users with a real email address must be created. A pool of dummy groups will be also created. Various invitations with different user rights to different groups must be sent to these users in order to test the performance and correctness of the mechanism.

Related non functional requirements

See 4.5.19 Invite a User.

4.5.23 Invite a User to a Complex Object

Description and priority

This functionality is a specialization of Invite a User (see 4.5.19) and deals with the making of a proposal to a user to acquire access rights over a complex object. Complex objects are: workflows, ImpECt reports, ARTE exhibition catalogues, Collections, etc. The invitation implies that a notification message is sent to the user and to the User Manager. Invited user may accept or reject. According to the user decision, the User Manager updates the user profile and the VO Manager gives him suitable roles.

User Requirements Fulfilled

See 4.5.19 Invite a User.

Numbers

This functionality will be executed each time a DILIGENT user needs to invite another one to participate in a DILIGENT complex object, e.g. a Compound Service, a Report.

Constraints and Assumptions

The user is already registered within the system, the Object exists or it is in the definition phase.

UML Diagrams

See 4.5.19 Invite a User.

Grid Exploitation

See 4.5.19 Invite a User.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.19 Invite a User.

Testing issues

A set of dummy users with a real email address must be created. A pool of dummy complex objects, e.g. compound services, reports, will be also created. Various invitations with different user rights to different objects must be sent to these users in order to test the performance and correctness of the mechanism.

Related non functional requirements

See 4.5.19 Invite a User.

4.5.24 Invite a Group

Description and priority

A DILIGENT user may want to perform an invitation to a group of users to join a DL or to have access to a Complex Object, etc. In order to do it the DILIGENT user discovers and selects the group to invite (see 4.5.13) and proposes the rights this group will have on the new item/environment (see 4.5.25). At the end of this process the Group Manager receive a notification about the operation. Invited groups can accept or reject the invitation via the

Group Manager that updates the group profile and notifies the VO Manager in order to give suitable roles to group members.

User Requirements Fulfilled

In order to simplify and promote the usage of various products of the DILIGENT infrastructure, e.g. DLs, Collections, and in accordance with the two user communities this invitation mechanism has been identified.

Numbers

This functionality will be executed each time a DILIGENT user needs to invite a group of users to participate in a DILIGENT activity.

Constraints and Assumptions

For each group the system maintains all the information needed to notify it, e.g. the email address of the group manager.

UML Diagrams

Due to the level of detail adopted within this functional specification document and to the simplicity of this functionality, no further diagrams are needed. Moreover this functionality can be used in different contexts, e.g. see Figure 13: Define a DL.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the help of the Portal, will supply this functionality.

Use Stories

See 4.5.13 Select Groups.

Testing issues

A set of dummy groups with a real email address for the group manager must be created. Various invitations with different group rights must be sent to these groups in order to test the performance and correctness of this mechanism.

Related non functional requirements

Usability is the most important requirement for this functionality. Reliability is another important aspect because the group must receive just valid invitation.

4.5.25 Propose Group Rights

Description and priority

The operation of invitation of a group to an item/environment implies the proposal of the rights the group will have on this new item/environment. This functionality covers the latter activity. It is important to point out that any operation made on a group is reflected on any user belonging to it.

User Requirements Fulfilled

This functionality contributes to complete the user requirements fulfilled by the Invite a Group functionality.

Numbers

This functionality will be executed each time an invitation to a user is performed by the Invite a Group task.

Constraints and Assumptions

A user-friendly language enabling to express group rights will be available.

UML Diagrams

The functionality is quite intuitive no further diagrams are needed. See Figure 24: Users Management.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.24 Invite a Group.

Testing issues

See 4.5.24 Invite a Group.

Related non functional requirements

See 4.5.24 Invite a Group.

4.5.26 Invite a Group to a DL

Description and priority

This functionality is a specialization of Invite a Group (see 4.5.24) and deals with the making of a proposal to a group to join a DL. The invitation implies that a notification message is sent to the User Manager that may accept or reject changing accordingly the Group profile. Another notification message is sent to the VO Manager who, following the VO rules and the group choice will give suitable roles to group members.

User Requirements Fulfilled

See 4.5.24 Invite a Group.

Numbers

This functionality will be executed each time a DILIGENT user needs to invite a group to participate in a DILIGENT DL.

UML Diagrams

See 4.5.24 Invite a Group.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.2.1 Define a DL.

Testing issues

A set of dummy groups with a real group administrator email address must be created. A pool of dummy DLs will be also created. Various invitations with different group rights to

different DLs must be sent to these groups in order to test the performance and correctness of the mechanism.

Related non functional requirements

See 4.5.24 Invite a Group.

4.5.27 Invite a Group to a Complex Object

Description and priority

This functionality is a specialization of Invite a Group (see 4.5.24) and deals with the making of a proposal to acquire access rights over a complex object. Complex objects are: workflows, ImpECt reports, ARTE exhibition catalogues, Collections, etc. The invitation implies that a notification message is sent to User Manager that may accept or reject changing accordingly the Group profile.

User Requirements Fulfilled

See 4.5.24 Invite a Group.

Numbers

This functionality will be executed each time a DILIGENT user needs to invite a group to participate in a complex object, e.g. Compound Services, Reports.

UML Diagrams

See 4.5.24 Invite a Group.

Grid Exploitation

None.

Mapping between functionalities and DILIGENT services (system integration)

The Dynamic VO Support service, with the support of the Portal, is in charge to supply this functionality.

Use Stories

See 4.5.24 Invite a Group.

Testing issues

A set of dummy groups with a real group administrator email address must be created. A pool of dummy Objects, e.g. compound services, reports, will be also created. Various invitations with different group rights to different Objects must be sent to these groups in order to test the performance and correctness of the mechanism.

Related non functional requirements

See 4.5.24 Invite a Group.

4.6 Notifications Management

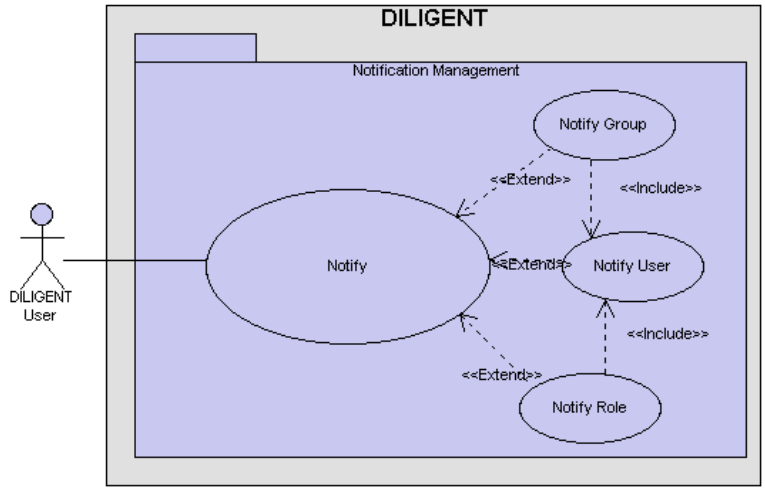

Figure 26: Notification Management (use case diagram)

4.6.1 Notify

Description and priority

This functionality models a generic notification action to inform a user about an operation to execute. This action can be performed using the push modality, i.e. the user receives a notify message (e.g. an email), or the pull modality, the request is stored on the system and the users ask to the systems for pending requests.

User Requirements Fulfilled

This functionality has been introduced in order to model the generic part of the various notifications the system will offer.

Numbers

See the more specific notifications functionalities.

Mapping between functionalities and DILIGENT services (system integration)

The DILIGENT portal is the main service involved into this task.

4.6.2 Notify Role

Description and priority

This functionality models a generic notification action to inform all the users with a given role within the context of a VO. This action can be performed using the push modality, i.e. the user receives a notify message (e.g. an email), or the pull modality, the request is stored on the system and the user asks to the systems for pending requests.

User Requirements Fulfilled

This functionality has been extracted factoring various functionalities, e.g. Ask for DL Creation, Ask for DL Update.

Numbers

The frequency of this functionality depends to the frequency of the 'caller' UCs. As the callers UCs are instantiated to perform operations having scarce frequency, this functionality is executed in accordance.

Constraints and Assumptions

This functionality asks for a notification mechanisms, i.e. a mechanisms allowing the system to inform the DL Manager about a pending requests. There are no particular requests for

the notification model, i.e. the push model as well as the pull model can be realized and supported.

UML Diagrams

See 4.6 Notifications Management

Grid Exploitation

None

Testing issues

There are no particular issues in testing this functionality.

Related non functional requirements

Security aspects related with the notification mechanism is mandatory in order to prevent improper actions. Moreover the availability of the notification mechanisms must be acceptable in order to prevent loss in communication.

4.6.3 Notify User

Description and priority

This functionality models a generic notification action to inform a particular user within the DILIGENT Infrastructure. This action can be performed using the push modality, i.e. the user receives a notify message (e.g. an email), or the pull modality, the request is stored on the system and the user asks to the systems for pending requests.

User Requirements Fulfilled

See 4.6.1 Notify and 4.6.2 Notify Role

Number

See 4.6.1 Notify and 4.6.2 Notify Role

Constraints and Assumptions

See 4.6.1 Notify and 4.6.2 Notify Role

UML Diagrams

See 4.6 Notifications Management

Grid Exploitation

None

Testing issues

There are no particular issues in testing this functionality.

Related non functional requirements

See 4.6.1 Notify and 4.6.2 Notify Role

4.6.4 Notify Group

Description and priority

This functionality models a generic notification action to inform a particular group of users within the context of a VO. This action can be performed using the push modality, i.e. the user receives a notify message (e.g. an email), or the pull modality, the request is stored on the system and the user asks to the systems for pending requests.

User Requirements Fulfilled

See 4.6.1 Notify and 4.6.2 Notify Role

Number

See 4.6.1 Notify and 4.6.2 Notify Role

Constraints and Assumptions

See 4.6.1 Notify and 4.6.2 Notify Role

UML Diagrams

See 4.6 Notifications Management

Grid Exploitation

None

Testing issues

There are no particular issues in testing this functionality.

Related non functional requirements

See 4.6.1 Notify and 4.6.2 Notify Role

5 CONTENT AND METADATA MANAGEMENT

5.1 Introduction

This functionality has been partitioned in three areas that group the functionalities of Content and Metadata management. These areas are:

- **Content Management**

  This area covers the Use-Cases related with the content management service as part of WP1.3 ("Content and Metadata Management"). These Use-Cases stem from both the ARTE and the ImpECt scenario that include:

  - *Objects Management*

    ARTE scenario: Access Objects, Save Object, Remove Object, Browse Objects, Browse Archives

    ImpECt: Remove [Object], Import [Object], Browse [Object]

  - *Collections Management*

    ARTE scenario: Collection Management, Create a Collection, Remove a Collection, Update a Collection, Define Membership Criteria

  - *Workspaces Management*

    ARTE scenario: Manage Workspace, Manage Student Workspace, Manage an ARTE DL Workspace

  Moreover, the DoW lists some basic functionalities, that Content Management will provide within the DILIGENT infrastructure. These functionalities were discusses and agreed upon at the CERN meeting on December 16th:

  - Distributed storage of arbitrarily sized documents on dedicated DILIGENT storage nodes including replication and partitioning of the data, dynamic re-organization for improved performance through exploitation of existing technology from distributed databases and file-based grid storage techniques
  - Maintenance of global data dictionary holding location, replication, and partitioning information for the data and mapping to system-wide unique URIs as logical document identifiers which conceal physical storage details (database or file-based, location, partitioning, replication)
  - Access of documents stored on external 3rd party data sources including "wrapping" of the data sources with potentially limited access patterns (translation of access request, data transformation to DILIGENT document format)
  - Change tracking in DILIGENT storage nodes and 3rd party data sources through trigger/polling mechanisms, maintenance of registry for interested DILIGENT services to be notified upon change
  - Transaction-aware change and removal mechanisms for documents with a given URI including distributed change in/removal of all replicas, triggering of change notification

  A fundamental principle is to re-use existing techniques from (1) distributed database products and (2) the EGEE middleware for file-based storage. The main focus of "Content Management" within DILIGENT will be on (a) change tracking and notification mechanisms and (b) 3rd party data source coupling.

  Subsequently, we have organized the Use-Cases (UC) that express the Content Management's "core" functionality into five packages, namely:

- **URI Management** (maintenance of URIs as system-unique logical identifiers for DILIGENT documents)
- **Change Tracking** (registration of services "listening" to changes in data sources, observed by triggering/polling mechanisms)
- **Storage** (all UC related to storage and removal a document with all partitions and replicas on dedicated DILIGENT storage nodes)
- **Data Source Coupling** (all UC related to coupling and wrapping 3rd party data sources, including transformation of access requests)
- **Access and Content** (all UC related to access one document's content by picking a replica-hosting node and possibly re-assembling it from its partitions).

- **Metadata Management** (see Section 5.11)
- **Annotation Management** (see Section 5.12)

5.2 Content Management: Objects Management

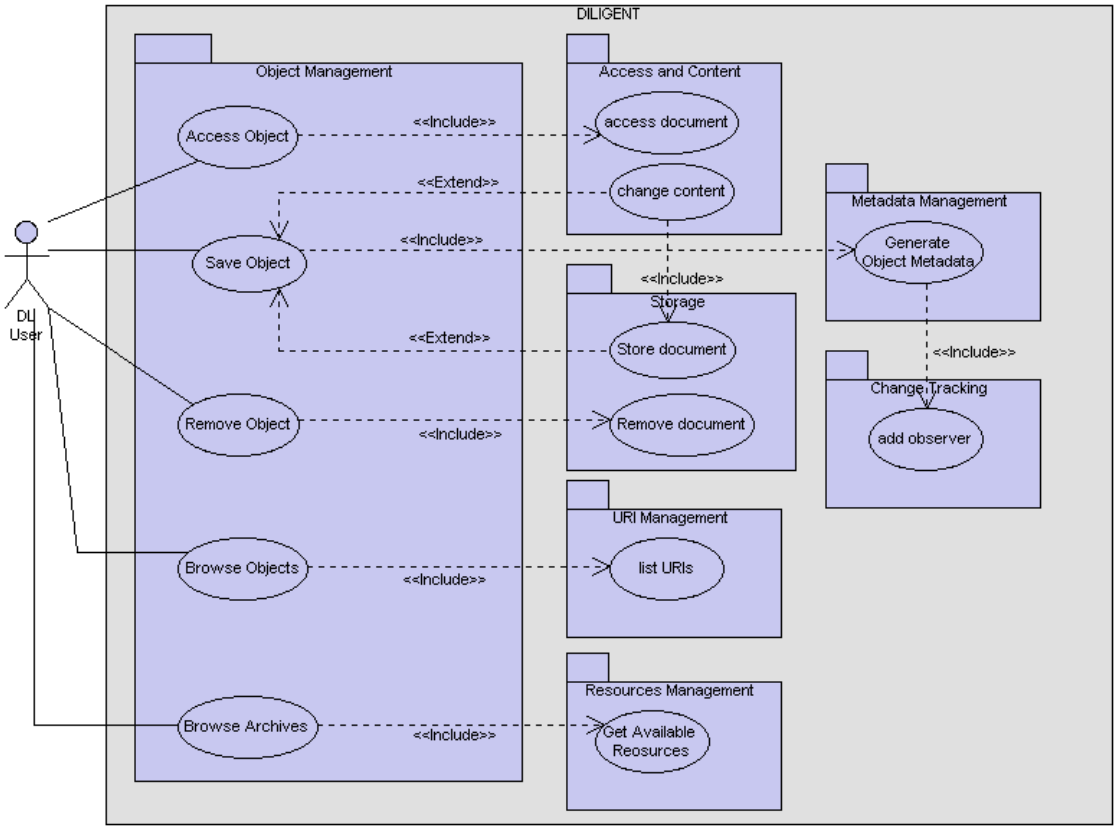

Figure 27: Objects Management (use case diagram)

5.2.1 Access Object

Description and Priority

This Use-Case comprises two basic functionalities (1) the access to an object's content by its description and (2) the visualization of that content. From a content management perspective, access to an object's content can be conducted by means of various search operations (see the respective Use-Cases). The actual fetching of the content data rests upon a unique URI, which translates to one or more physical location(s) of that object. In case that the object resides in a 3rd party data source with proprietary access patterns, an appropriate wrapping service must be employed to fetch that object. Furthermore, the content management service must decide from which node to fetch the object's content, if

there is a choice. Finally, if the object is being incorporated into a collection, it must be registered for notification upon change.

Requirements

This functionality encapsulates the requirements to (1) locate and (2) efficiently fetch an object's content. Optionally, (3) the "consuming entity" must be registered to be notified upon change.

Numbers

As this functionality will be invoked whenever some content must be read from storage, it will be very frequently used.

Constraints and Assumptions

The provided URI must be valid, i.e. map to a physical location within reach of the DILIGENT infrastructure. Furthermore, the hosting node must be cooperative, i.e. deliver the object in reasonable time. Content management must ensure the consistency of its catalogue (data dictionary), i.e. removal or relocation of an object must be detected by change notification mechanisms and propagated towards updated catalogue information. Diligent storage nodes will be equipped with low-level notification functionality which "fires" upon transactional commit of a change operation. Third party data sources will probably lack such "hooks" and need to be observed by polling/crawling techniques. By concept, there will be delays between the actual change of an object and its detection. This must be appropriately taken into account by the access object functionality, which might, thus, request an object, which does no longer exist in the 3rd party data source.

From the application scenario's perspective, appropriate strategies for 3rd party data source incorporation must be applied:

- Data may continue to exclusively reside in the 3rd party data source when (1) it cannot be extracted for legal or technical reasons or (2) the application scenario is fault-tolerant w.r.t. autonomous (and probably undetected/late-detected) changes of the data. The major advantage lies within management of the data by a 3rd party autonomous component, which is responsible for collecting, maintaining, and updating the data. The disadvantage is given by the incorporation of wrapper techniques, which have to deal with restricted access patterns, query transformation, and parsing techniques.
- Data can be extracted from external data sources and fed into DILIGENT storage nodes if (1) freshness of the data is non-critical, (2) distribution and replication should be controlled from within DILIGENT, and (3) efficiency of access, remove, and relocate operations is important.

Grid Exploitation

Grid exploitation comes into play whenever data needs to be stored, in particular if it is replicated on distributed nodes. That is, (1) an object's content either resides completely on a node/different nodes or (2) a voluminous object (e.g. a video stream) is partitioned into chunks, which are stored on different nodes. While in the first case the "right" node may be picked due to load balancing concerns, in the latter case, many nodes must be accessed in parallel.

Grid exploitation of data which resides in 3rd party data sources is restricted by (1) the number of DILIGENT nodes which "wrap" this data source and (2) the access patterns of the data source (e.g. number of concurrent accesses).

System Integration

As explained in "Constraints and Assumptions", "Access Object" is a functionality, which invokes numerous other functions like (1) wrapper services (including query transformation,

caching techniques, and result parsing), (2) data dictionary lookup, (3) load balancing, and (4) object composition from partitions.

Use Stories

- An object is partitioned into three chunks, which are stored on three DILIGENT storage nodes. When an access object request comes in, a lookup in the data dictionary returns the DILIGENT nodes hosting the chunks (which may, themselves, be duplicated on other nodes) and a composition rule to create the object's content from the chunks. The DILIGENT storage nodes receive a request to return the chunks. When a chunk arrives at the node performing the "Access Object" functionality, it will be cached until all chunks arrive. Finally, the complete object is composed out of the chunks.
- An object is stored on a 3rd party data source. When an access object request comes in, a lookup in the data dictionary returns the DILIGENT node(s) which host(s) the wrapping service(s) of this data source. The wrapper transforms the access request into a data source specific query (taking into account the limited access patterns and query syntax). The result (i.e. the object content) is received and extracted by a parser service.

Testing Issues

The most critical parts of this system lie within (1) reliability and efficiency of access operations in the light of replicated and distributed data and load balancing concerns; (2) change detection in 3rd party data sources; and (3) query transformation and result parsing in 3rd party data source access.

Related Non-Functional Requirements

In particular, 3rd party data source must be willing to be incorporated into a DILIGENT infrastructure. This fact is particularly expressed by (1) stable interfaces, (2) little access restrictions, and (3) acceptable QoS (response times, number of concurrent accesses, etc.).

5.2.2 Save Object

Description and Priority

According to the ARTE requirements document, the "Save Object" functionality permanently stores a previously retrieved object into the workspace. From our perspective, (1) one or more storage nodes must be allocated, (2) the object must be partitioned and/or duplicated onto these nodes, and (3) the data dictionary must be updated. In case, an object with an identical URI has been saved before, this functionality corresponds to an update operation where (1) storage nodes and (2) partitioning rules are fixed, (3) but update must be commenced on all involved nodes and (4) the notification service must be invoked upon transactional commit.

Requirements

As pointed out, this Use-Case harbours many functionalities of vital importance for content management. In particular, the distinction between storage of a new object (with node allocation and object partitioning) and update of an existing object must be taken into account. That is, an appropriate replication scheme must be fixed for purposes of the application scenario (query workload, number of concurrent accesses, etc.). In this way, update of and existing object may also trigger a revised replication (allocate new nodes, dispose existing nodes). Write operations in coupled 3rd party data sources are no objective of content management in general, and this functionality, more specifically.

Numbers

This functionality will be invoked very frequently. Each write access to an arbitrary object breaks down to this functionality.

Constraints and Assumptions

For the time being, we assume that the DL incorporates nodes, which are willing to store content. However, sufficient capacity and node uptime are not considered as vital preconditions but must be handled by the service. Furthermore, we assume we have access to a data dictionary (possible replicated) in order to conduct the operation. If it does not exist or is not accessible, the service cannot commence.

Grid Exploitation

Grid exploitation comes into play whenever data needs to be stored. Particular attention must be posed in case of replicated objects, which have to be concurrently saved or updated on different nodes. Likewise, change notification will likely "touch" many interested services, which may also reside on different nodes.

System Integration

Like the "Access Object" Use-Case, "Save Object" incorporates many fundamental content management functionalities like (1) node allocation, (2) object partitioning, (3) change notification, and (4) data replication.

Use Stories

A spacious object, say a video sequence, shall be appropriately stored. That is, the video streams shall be decomposed into a number of scenes where each is stored on a different node with a replicate on just another node. Twice the number as there are scenes must be allocated and requested for storing a object of a particular size. When this is done, the transportation of the content to that nodes and the actual storage must be commenced in a complex distributed transaction. When it is ready to commit, (1) the data dictionary is updated and (2) interested services are registered for change notification.

Testing Issues

The main problem of "Save Object" lies within establishing an appropriate replication/partitioning scheme, which must be tested from a performance point of view (w.r.t. to the typical query workload).

Related Non-Functional Requirements

Especially for complex objects, decomposition rules must (partially) be provided by the invoking service. This comes in handy when read requests do not actually access the object as a whole but only certain parts of it.

5.2.3 Remove Object

Description and Priority

A certain object is removed from the workspace upon user request.

Requirements

As an object may be replicated on different nodes and/or partitioned into chunks residing on different nodes, a removal mechanism must concurrently remove the object in all its partitions from all involved nodes. Moreover, registered services for change notification must be notified from object removal and be automatically unregistered.

Numbers

It is still to be determined, if object removal is a frequent operation. In fact, this heavily depends on the application scenario, at hand. In many cases, removal operations will rarely occur.

Constraints and Assumptions

In order to have to removal operation complete successfully, all storage nodes (as listed in the data dictionary) must be within reach (operational and on-line). If this can not be guaranteed, there might be a mechanism which postpones the actual physical removal of an object's replicate on a dedicated node and tags this object and "non-existent, to be deleted on node(s) {x, y, …}" in the data dictionary.

Grid Exploitation

Dedicated grid functionalities will only be exploited, if the object is replicated/partitioned onto different nodes that can concurrently conduct the delete operation.

System Integration

There will be a dedicated "Remove Object" functionality in the system, which is responsible for distributed delete. It also involves data dictionary lookup and change notification.

Use Stories

Suppose a video stream is partitioned onto different DILIGENT storage nodes and each chunk is, itself, replicated on a backup node. If a removal request comes in, all affected nodes as listed in the data dictionary receive a dedicated removal request concurrently. If all removals complete smoothly, the object's entry can safely be removed from the data dictionary. If there are non-available storage nodes, the object must be tagged for postponed delete and be physically deleted as soon as the node is up again. Important: there must be no object accesses, which bypass the data dictionary as central housekeeping instance.

Testing Issues

As the handling of non-available nodes is the major weakness of this service, testing must simulate such scenarios while keeping the transactional guarantees (in particular: conflict avoidance) in mind.

Related Non-Functional Requirements

Though the grid comes with a large extent of node autonomy, there must be mechanisms that DILIGENT storage nodes do not change their content, unless a "Save Object" functionality is invoked. Content management heavily relies on a central data dictionary that may become invalid/corrupted by "uncontrolled" data accesses.

5.2.4 Browse Objects

Description and Priority

The "Browse Objects" Use-Case harbors a functionality, which permits ARTE users to visualize an archive's content. A summary/header of each object is presented along with its metadata where the sort criterion is given by a user-selectable metadata element (e.g. author's name).

Requirements

"Browse Objects" is no core content management functionality as it actually invokes "Access Object" and also requires functionality of (1) search, (2) metadata management, and (3) visualization. In addition to "Access Object" a summary of the object must be generated for display (e.g. thumbnail, textual header, etc.).

Numbers

According to the requirements document, this is a frequently invoked functionality.

Constraints and Assumptions

As content management accesses objects by unique identifiers (URIs), but "Browse Objects" displays objects according to their membership in a certain archive, (1) the objects must be equipped with an archive membership tag and (2) there must be a search functionality which returns the URIs of all objects contained in a given archive.

Grid Exploitation

Very likely, an archive's content (i.e. the contained objects) is stored on separate storage nodes (or 3rd party data sources). In that sense, those objects can be accesses concurrently with "Access Object" requests being sent to the involved storage nodes in parallel.

System Integration

The "Browse Objects" Use-Case rests upon (1) content management, (2) search, (3) metadata management, and (4) visualization functionalities.

Use Stories

A user decides to browse the content of an image archive with the archive provided by the user. A search functionality returns (1) the URI of the archive (which is, itself, a DILIGENT object) and (2) returns the URIs of the contained images by the archive URI. At this point, content management comes into play and invokes the "Access Object" functionality, which can be done in a grid-aware fashion (i.e. in parallel, if objects are stored on separate nodes). Likewise, the summary (thumbnail) of the images will be computed in parallel and, finally, ranked for display according to their metadata.

Testing Issues

The main testing issues in this complex interaction of various DILIGENT services lie within (1) the correct invocation of those services and (2) the parallelization of the object access which must be in-line with the replication/partitioning scheme (according to data dictionary) and the current storage node's workloads.

Related Non-Functional Requirements

The generation of a summary representation for expressive, non-spacious visual representation of an object heavily depends on the media type. While DILIGENT may be equipped with transformation/extraction rules for common media types (text, images, video, etc.), complex user-defined objects will require dedicated rules to compute a "meaningful" summary representation.

5.2.5 Browse Archives

Description and Priority

The "Browse Archives" Use-Case is similar to the "Browse Objects" Use-Case but happens on a different storage hierarchy level. That is, all archives existing in a DL shall be visualized and ordered according to their description.

Requirements

See "Browse Objects".

Numbers

See "Browse Objects".

Constraints and Assumptions

See "Browse Objects".

Grid Exploitation

See "Browse Objects".

System Integration

See "Browse Objects".

Use Stories

See "Browse Objects".

Testing Issues

See "Browse Objects".

Related Non-Functional Requirements

See "Browse Objects".

5.3 Content Management: Collections Management

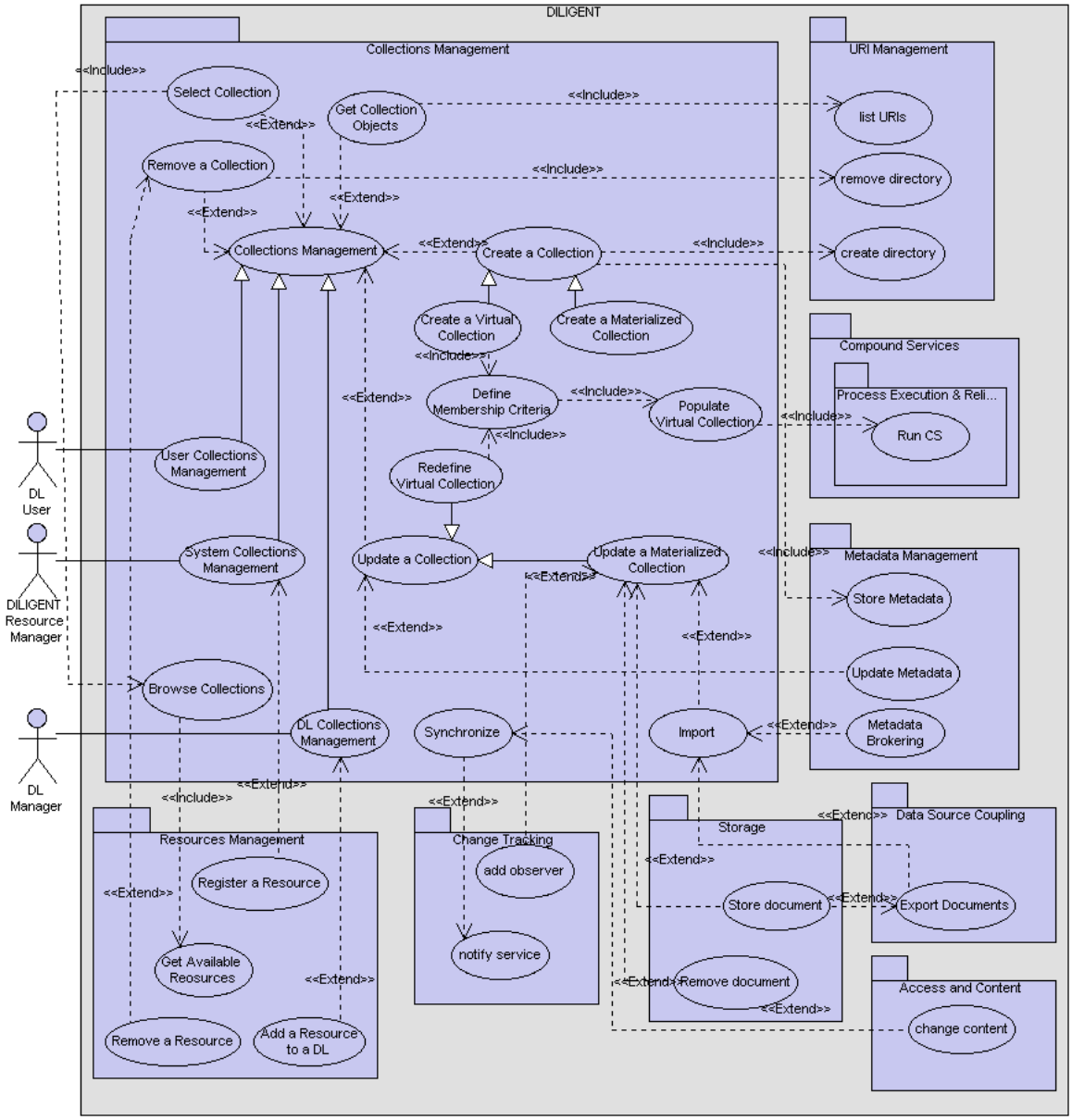

Figure 28: Collection Management (use case diagram)

5.3.1 Collection Management

Description and Priority

A collection is an entity, which comprises some content (i.e. DILIGENT objects). A detailed description is reported in Section 3.4. There are a number of "basic" collection functionalities (expressed by the "Create a Collection", "Remove a Collection", and "Update a Collection" Use-Cases, see there) to operate on this entity.

Requirements

From a content management point of view, a collection is just another (complex) object, which is identified and accessed by its URI. Therefore, any access/manipulation/removal operation implicitly invokes the respective content management functionalities ("Access Object", "Save Object", "Remove Object").

Numbers

For purposes of ARTE, a collection is a quite central entity to accommodate the content and assign services and users. Therefore, collection management will be invoked very frequently.

Constraints and Assumptions

In particular, the fitting of a specific operation to be conducted on the collection content will not be semantically checked. Therefore, users must carefully craft a collection and assign services to operate on it.

Grid Exploitation

See "Create a Collection", "Remove a Collection", "Update a Collection".

System Integration

In a sense, the content, services, and users assignment of a collection is its metadata, which has to be handled by metadata management. However a collection is a resource and thus the services and users assignment are fulfilled by manipulating resource policies, i.e. making use of Dynamic VO functionalities. In the other way around, content management is responsible for storing/accessing collections, which are abstracted as complex objects.

5.3.2 Create a Collection

Description and Priority

A named entity (collection) comprises a number of (user defined) objects (content). In case of virtual collection, the content is being imported into the collection by some membership criterion. In that sense, a virtual collection is more like a logical view than a physically materialized set of objects (i.e. a materialized collection).

Requirements

In case of v irtual collectiopns, the "Create a Collection" must meet a number of requirements, which are given by (1) automatic collection update upon "arrival" of new objects, which meet the membership criterion, and (2) automatic notification of permitted users whenever new objects (virtually) enter the collection.

Constraints and Assumptions

Besides manual specification of permitted users, further collection-wide restrictions may constrain the access of a specific user to a collection. That is, in general, only users, which exhibit certain properties, may be granted access.

Grid Exploitation

Automatic membership of newly arriving objects is an extension of the change notification mechanism is content management. That is, all objects are implicitly equipped with a change notification mechanism, which checks the membership criteria of all existing collections for inclusion of the new object. Grid exploitation comes into play when the checked objects and the checked collections reside on different storage nodes. In that case, they can be checked in parallel.

System Integration

"Create a Collection" invokes "Save Object" and "Change Notification".

Use Stories

See ARTE requirements document.

Testing Issues

In case of virtual collection, the most important part is given by the automatic membership of objects in a collection. That is, the implicit change notification upon object insertion, update, and removal in/from DILIGENT storage nodes and 3rd party data sources must be thoroughly tested.

5.3.3 Remove a Collection

Description and Priority

An existing collection is removed from storage and can be safely unregistered for notification upon object change.

Requirements

In terms of content management, a collection is a complex object that is (1) removed (by "Remove Object" invocation) and (2) unregistered for notification of arrival of new objects or changes in the contained objects.

Grid Exploitation

Grid exploitation comes into play, when the previously contained objects reside on different storage nodes. In this case, un-registration of change notification can be conducted in parallel.

System Integration

"Remove a Collection" makes use of the fundamental object removal and change notification mechanisms from content management.

5.3.4 Update a Collection

Description and Priority

Updating a collection involve the collection's content that may be expressed through the membership criterion.

Requirements

In case of virtual collection, the collection's content must be automatically adapted if the membership criterion is being changed. As the content is a virtual view onto physical objects, that change is transparent for content management. Access to a collection's object is then conducted by a search operation that takes the membership criterion as search predicate.

Constraints and Assumptions

In case of virtual collection, changing the membership criterion must be conducted such that a non-contradictive predicate is formulated which permits objects to become content of the collection. Furthermore, appropriate access structures (indexes) must exist to allow efficient access to the collection's content.

System Integration

"Update a Collection" requires functionalities from "Search" and "Content Management" (change notification, save object).

5.3.5 Define Membership Criteria

Description and Priority

A membership criterion is a logical predicate, which states the inclusion of DILIGENT objects in a collection. This predicate rests upon atoms which are (1) a basic metadata check (e.g. for name, identifier), (2) an archive membership, or (3) more complex metadata predicates (e.g. subject).

Requirements

The requirements document distinguishes between fixed and dynamic content, which does not make sense, from a functional point of view. In detail, we assume that no metadata tag is fixed but may be changed, and, thus also alter the membership of the object in a collection. Though this does not necessarily have to be true for all metadata, it is the most conservative assumption, which simplifies the handling of a membership criterion.

Constraints and Assumptions

We consider the membership criterion as an arbitrary complex Boolean predicate composed out of atoms, which are comparisons among the metadata tags. We do not permit fuzzy predicates, which return some scoring value, here.

System Integration

The major component of "Define Membership Criteria" is metadata management, as it provides the basic metadata tags, which are to be employed in the predicate. Furthermore, the search functionality must provide generic query processing where this membership criterion can be plugged into a selection clause.

5.3.6 Import

Description and Priority

A collection is a group of objects that are logically related. In order to populate a materialized collection, via this functionality, it is possible to gather documents or just to link documents maintained in a third party data source.

In the former modality when the importing begins, the collection management imports objects one by one by using the Export Documents of the Data Source Coupling area.

In the latter modality, the membership to data source has to be maintained and therefore the collection becomes a meta-object that is simply associated with the concrete instances of content objects that previously belong to the data source. A collection by itself doesn't contain any content, but metadata can be associated with it that can help in focussed searching. Often descriptions about collections are widely available which talk about the collection as a whole, and are supported by a single information provider. In the case the data source contains metadata objects in a proprietary schema the metadata has to be converted into the scheme the Collection creator decides for collection objects. For this purpose, DILIGENT metadata broker maintains a library of transformations that can convert

proprietary (XML) schemas into the other ones. This functionality is similar to the previous one except that there are additional steps involving the schema library and invocation of a particular transformation. The transformed schema is then stored and associated with the actual object in the storage.

This functionality can be also used when a new DL is created. W.r.t. metadata management, it is important to note that more than one DL may access the same collection. While modifying the collections objects maybe not be possible (due to copyright restrictions etc.), each individual DL might choose to add its own metadata like annotations to a content object or collection. Similarly, different DLs might choose different services or same services with different parameters to generate/update different sets of metadata about the same object. To accommodate these scenarios, one approach would be to maintain different metadata for different DLs. Therefore, when a DL is created, metadata management simply choose to copy the metadata for all collections that the DL includes.

5.4 Content Management: Workspaces Management

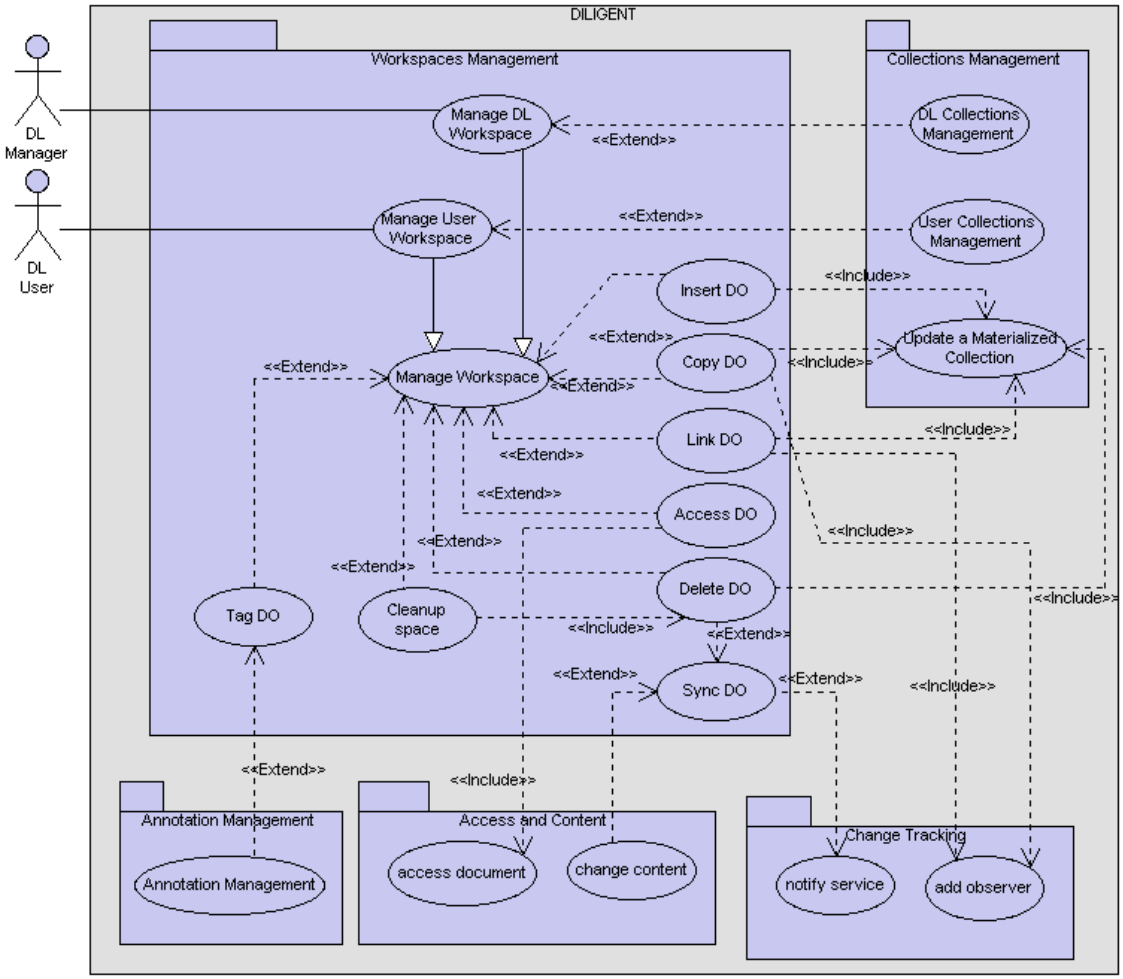

Figure 29: Workspace Management (use case diagram)

5.4.1 Manage Workspace

Description and Priority

The "Manage Workspace" Use-Case permits any user to perform certain operations like accessing/saving/removing objects, processing images, managing collections, and searching and retrieving objects. This requires a valid login of a member where each has its own workspace. Certain operations may be authorized whilst others are restricted. Clearly, this

does not exclusively comprise content management functionalities. However, content management comes into play (1) to recover from system failures, (2) to save retrieved objects, (3) to access a certain [stored] object, (4) to remove a certain [stored] object, and (5) to manage collections from within the workspace.

In particular, the following sub-functionalities belong to this group:

- Cleanup space: deletes all local digital objects and temporary space. Additionally deletes the container structures;
- Copy digital object (DO): duplicates a digital object to the local workspace if all authorization requirements are met and reports synchronization needs;
- Insert digital object: copies an external digital object to the local workspace;
- Link digital object: links a digital object to the workspace, allowing local naming and tagging and reports synchronization needs;
- Delete digital object: deletes a link or a physically stored digital object from workspace;
- Tag local object: inserts tags in a local object (some comments, a title etc, which are being defined by user profile schema). This tagging is supported by the Annotation functionality;
- Retrieve digital object: retrieves the actual digital object, through underlying storage mechanisms;
- Synchronize digital object: gathering notification about DO changes (e.g. deletion or updating) acts on the workspace accordingly in order to maintain the DO consistency.

Numbers

This functionality is supposed to be frequently required (several times a day), i.e. whenever a user interacts with DILIGENT.

Constraints and Assumptions

In terms of content management, access to and saving of objects is constrained by (1) available storage capacity, (2) compliance with access rights, and (3) existing access patterns to coupled external data sources, (4) security layer support for safeguarding privacy. Furthermore, the storage nodes must provide transactional support in order to enable recovery from system failures.

Grid Exploitation

Grid architectures come into play to serve two different purposes. One the one hand, data storage should be conducted on distributed grid nodes with transactional support to access the data. Although full independency of concurrent data accesses cannot be guaranteed in all cases, distributed transactions avoid conflicts while maintaining a better transaction throughput than on a single storage node. To enhance reliability and query processing times, data will be replicated on different nodes.

On the other hand, DILIGENT explicitly seeks to attach existing data sources. To do so, data source specific wrappers must be employed which offer a common interface to the DILIGENT infrastructure. These wrappers must deal with potentially restricted access patterns of the data sources and translate any request to a data source specific format, filter and transform the results. Each wrapper may potentially reside on a distinct grid node, which is responsible for accessing a particular data source. Since the coupled data sources may safely be regarded as mutually independent, such that they can be queried in parallel.

System Integration

The "Workspace Management" Use-Case requires functionalities to be provided by "Content Management" and "Search Service".

Use Stories

- Case 1: An end user wants to locally store a digital object that she/he has located for further use and processing, however does not want to depend on the future object availability. So she/he requests that a local copy of the object is being created to his personal storage area. The systems checks to allow this operation, with regards to object security, user quota and permissions and provides the local copy. The object can be locally renamed and referenced by its identifier.
- Case 2: An end user wants to search for images that mach an image that he holds on his personal computer. He submits his request; the UI uploads the digital object to her/his personal storage; the system extracts the similarity search features through DILIGENT feature extraction components; finally the system searches for digital objects that meet match criteria on the same sate of features, either by checking already extracted features or by on-the-fly extraction.

Testing Issues

The major testing issues lie within the reliability of accessing/storing objects. This involves the access to coupled (i.e. DILIGENT external) systems, replication issues, and transactional processing.

Related non-functional requirements

Underlying storage of user profiles exploits distributed storage management.

Supports both creating local copies of objects or links to other collection contained digital objects.

Secure access to personal and temporal data storage area.

Facilities to watch and limit personal storage to specific use and size.

5.4.2 Manage User Workspace

Description and Priority

The "Manage User Workspace" is a specialization of the "Manage Workspace" Use-Case. From an interface perspective, if offers exactly the same functionality as the "Manage Workspace" Use-Case. Potential differences (though not stated in the requirements document) might be (1) restricted access to certain objects, (2) restricted search operations, and (3) restricted collection management facilities which is closely related to (1). In general, (2) is not subject of the content management functionality and (1), (3) can be achieved by appropriate content security mechanisms, which is a dedicated functionality. Therefore, the "Manage User Workspace" should be dropped unless the user community provides more specific insights into the distinction to the "Manage Workspace" functionality.

Requirements

Specific restricted student workspace, also see "Manage Workspace".

Numbers

See "Manage Workspace".

Grid Exploitation

See "Manage Workspace".

System Integration

See "Manage Workspace".

Use Stories

See "Manage Workspace".

Testing Issues

See "Manage Workspace".

Related Non-Functional Requirements

See "Manage Workspace".

5.4.3 Manage a DL Workspace

Description and Priority

The "Manage a DL Workspace" Use-Case differs from "Manage Workspace" and "Manage Student Workspace" in that it permits additional related functionalities to be invoked. These include (1) workshop management (preparation), (2) exhibition catalogue creation, (3) course management, and (4) video processing.

Requirements

From a content management perspective, this Use-Case complements the "Manage Workspace" Use-Case in that it must provide all storage and change notification functionalities required to enable the additional operations (see there).

Numbers

See "Manage Workspace".

Constraints and Assumptions

A DL must exist and the user must be a DILIGENT member (i.e. must have the required credentials).

UML Diagrams

Grid Exploitation

See "Manage Workspace".

System Integration

See "Manage Workspace".

Use Stories

See "Manage Workspace".

Testing Issues

See "Manage Workspace".

Related Non-Functional Requirements

See "Manage Workspace".

5.5 Content Management: Storage

Description and Priority

Storage comprises all functionalities to (1) initially store, (2) finally remove, (3) partition, and (4) replicate documents. For the time being, the storage service is invoked to either store a document in one chunk (i.e. unpartitioned) in one replica or partition and/or replicate the document according to a user-provided partitioning and replication scheme. In

no case, the invoking service has to indicate the DILIGENT node(s) on which to store the document. For the time being, those nodes will be automatically determined from their current task and storage load.

This functional area comprises the following UCs:

- **Store Document** is the entry point for arbitrary storage requests, which includes (1) "plain" document storage (no partitioning, no replication), (2) replicated document storage (no partitioning, replication), (3) partitioned document storage (partitioning, no replication), and (4) storage in replicated partitions (replicated partitions). It picks the storage nodes according to their storage load (free capacity), task load (current availability). Later, invoking services may additionally influence that choice by providing "hints" including location of services accessing the document content.
- **Store Partitioned Document** introduces a plain partitioning scheme into document storage. That is, a document is decomposed into a certain number of disjoint chunks. In a plain scenario, the invoking service does not care about the number and size of partitions and leaves this choice to the "Store Partitioned Document" service, which distributes the partitions for certain optimization criteria (like balanced storage, etc.). Alternatively, invoking services may also provide "hints" like number and size of the partitions (also see "Store Document"). The actual document partitioning is conducted by means of the "Partition Document" service (see there).
- **Store Replicated Document** introduces replication for improved availability and performance to document storage. Invoking services will provide a number of replicas whereas "Store Replicated Document" decides on the physical location of those replicas. Actual replication is conducted by "Replicate Document" (see there).
- **Store Replicated Partitioned Document** is a mixture of replication and partitioning in a sense that a document is partitioned into chunks that are replicated on different storage nodes (a little bit like RAID). Please refer to "Store Partitioned Document" and "Store Replicated Document" for further comments on replication and partitioning.
- **Partition Document** will decomposes a given document into disjoint according to a plain partitioning scheme. Notice, that document partitioning may either be conducted on invocation by "Store Partitioned Document"/"Store Replicated Partitioned Document" or, alternatively, later partition a previously stored document which requires access to the data dictionary in order to add this partitioning scheme.
- **Replicate Document** replicates a document on a number of DILIGENT storage nodes. Again, it may either by invoked by "Store Replicated Document"/"Store Replicated Partitioned Document" or, alternatively, later replicate a previously store document which requires access to the data dictionary to add the replication information.
- **Remove Document** is a functionality that finally removes a document with all partitions and replicas. It also revokes the document URI, meaning the corresponding entry in the data dictionary will be deleted. If observing services exists, they will be notified and the observer will be terminated.
- **Cluster Document** physically clusters a document with another document, i.e. stores it at the same node for efficiency reasons.

Requirements

Storage answers the requirements of efficient storage and removal of documents including replication support and document partitioning.

Constraints and Assumptions

In order to have a document storage complete successfully, dedicated DILIGENT storage nodes must be within reach and willing to accommodate this document (enough storage capacity).

UML Diagrams

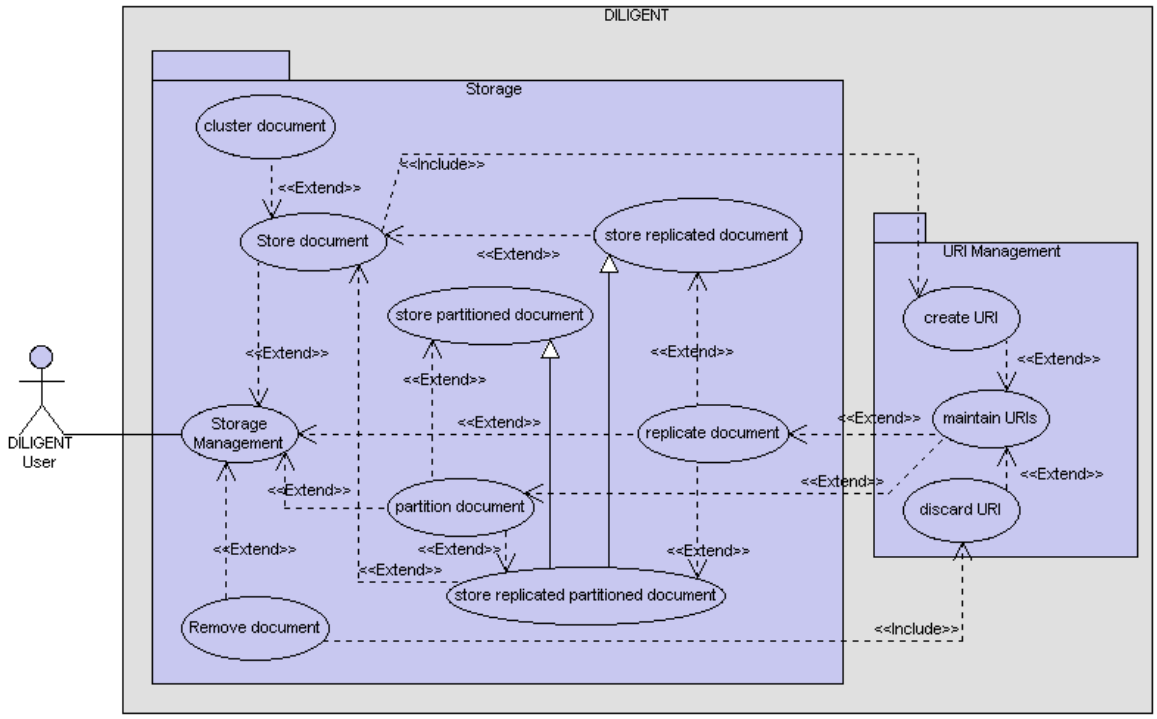

Figure 30: Content Management – Storage (use case diagram)

Grid Exploitation

Grid can efficiently be exploited as soon as it provides many dedicated storage nodes. Besides the parallel handling of concurrently incoming storage requests for different documents, storing document partitions and replicas is another example of potential grid exploitation.

System Integration

Storage exclusively comprises "core" DILIGENT services.

5.6 Content Management: Access and Content

Description and Priority

Access and Content names the pool of remaining functionalities in Content Management. Precisely, it accommodates functionalities to access and change a document's content.

This functional area comprises the following UCs:

- **Access Document** will return a document's content for a given URI. That comprises access to the data dictionary to figure out the physical location and replication/partitioning information. Potentially, it needs to visit several storage nodes and assembles the document's content according to the partitioning scheme.
- **Change Content** modifies the content of a (previously stored) document. It also needs to access the data dictionary to find out the physical location of a document and (if needed) adapt the partitioning scheme. It also represents the "official" triggering mechanism for services observing the document.

Requirements

Access and Content is on document access and change.

Constraints and Assumptions

Only documents with write access can be modified. This implicitly excludes any document from a coupled data source. Other (user or service specific) access rules can exist and must be obeyed by Access and Content.

UML Diagrams

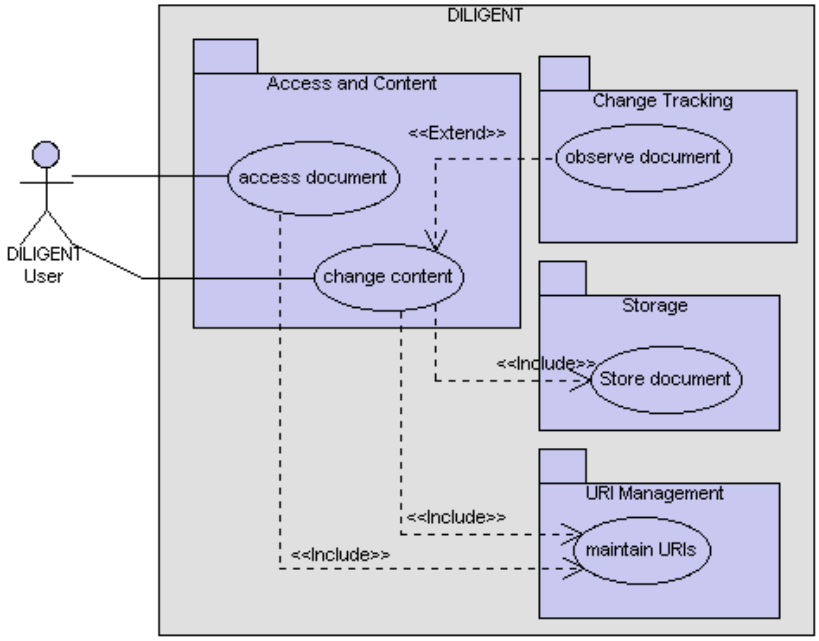

Figure 31: Access and Content (use case diagram)

Grid Exploitation

Both "Access Document" and "Change Content" can be parallelized in the presence of partitioning. In those cases, the affected storage nodes can be accessed in parallel.

System Integration

Access and Content exclusively comprises "core" DILIGENT services.

5.7 Content Management: URI Management

Description and Priority

URI Management is responsible for providing, maintaining, and revoking URIs that are logical document identifiers used throughout DILIGENT services (except for Content Management services). They are maintained in a data dictionary that is a global repository, yet not necessarily centrally stored. An entry in the data dictionary comprises the URI, the partitioning scheme (number, size and location of the partition), the replication scheme (number and location of all replicas), or a combined replicated partitions scheme (size of partitions, number and location of their replicas). Notice, that we employ a basic partitioning scheme that separates a document into disjoint chunks. The document can be re-constructed by concatenating all chunks.

This functional area comprises the following UCs:

- **Maintain URIs**: manages the global data dictionary and provides some basic URI management functionalities.

- **Create URI** is included as a partial functionality of "Maintain URIs". Upon request, it creates a new URI, for a document, allocates the physical storage space according to replication/partitioning scheme, and inserts this information into the data dictionary.
- **Discard URI** is also included as a partial functionality of "Maintain URIs" and will release a URI of a document by removing its entry from the data dictionary and marking all occupied storage space as available.
- **List URIs** is another partial functionality of "Maintain URIs" and lists all documents, maintained by this data dictionary.
- **Create Directory** creates a virtual document container as part of a hierarchical structure. Documents can be members of directories.
- **Remove Directory** removes an existing document container if it is empty, i.e. contains no references to documents or sub-directories.
- **Move Document** moves a document from one directory to another. Documents with no explicit assignment to a directory are assumed to reside within a virtual top-level directory.

Requirements

URI Management answers the requirement of a global naming repository that translates logical identifiers into physical locations enhanced by a replication and partitioning mechanisms to exploit the capabilities of the grid.

Constraints and Assumptions

The data dictionary will require a sophisticated naming and allocation scheme. It is up to the architectural design to either use pre-existing EGEE middleware for purely file-based storage or to come up with a new proposal that also covers database-supported storage. Possibly, existing EGEE technology will be supplemented to cover database storage, as well.

UML Diagrams

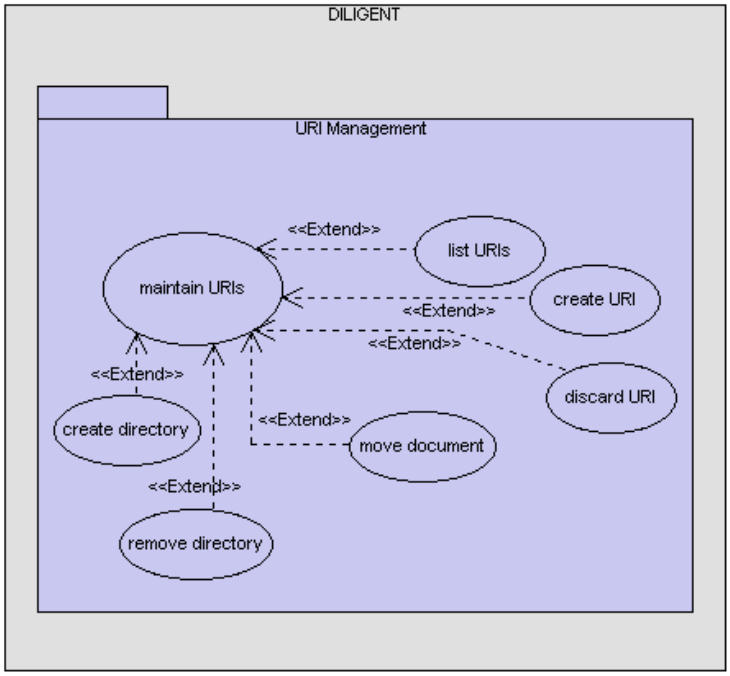

Figure 32: URI Management (use case diagram)

Grid Exploitation

Grid capabilities can be exploited for distributed data dictionary management to enhance transactional throughput through parallel "Create/Discard URI" requests.

System Integration

URI Management exclusively comprises "core" DILIGENT services.

5.8 Content Management: Data Source Coupling

Description and Priority

Data Source Coupling comprises all functionalities to couple a 3rd party data source (web site and the like) to DILIGENT Content Management. This primarily includes "wrapping" that data source to permit read-only access to the coupled data stock. It also comprises assignment of URIs to the documents stored in the data source, transformation of document formats, and handling of limited access patterns.

This functional area comprises the following UCs:

- **Register Data Source** couples a data source and the included documents to DILIGENT. It creates URIs for all documents and inserts them into the data dictionary.
- **Unregister Data Source** de-couples a data source and permanently removes all contained documents from DILIGENT Content Management. That is, though these documents will not by physically deleted they will be no more within reach of Content Management. The corresponding URIs must be discarded, i.e. removed from the data dictionary. If any document observer is running on a contained document, it will be stopped, and "listening" services will be notified.
- **List URIs in Data Source** returns a complete list of documents, which are contained in the coupled data source along with their DILIGENT URIs.
- **Export Documents** has been introduced to "download" all documents from a coupled data source and transfer it into DILIGENT storage nodes for enhanced availability, increased performance, and write accesss.

Requirements

Data Source Coupling attaches a data source to DILIGENT Content Management.

Constraints and Assumptions

As 3rd party data source may be equipped with a large variety of access protocols and data formats, dedicated wrappers must be implemented to access a data source. Though there has been some work on semi-automatic wrapper generation, it is not the focus of Content Management within DILIGENT. We will rather implement wrappers on a need to have basis.

UML Diagrams

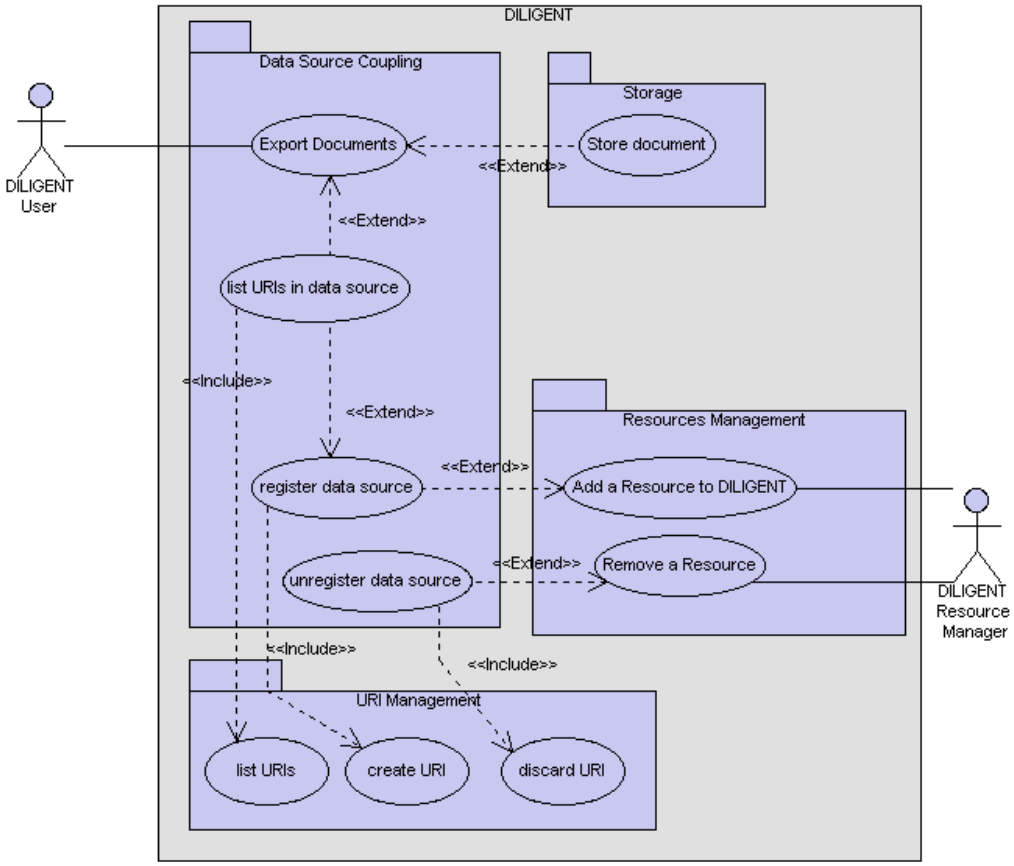

Figure 33: Data Coupling (use case diagram)

Grid Exploitation

Grid exploitation is severely restricted by the capabilities of the coupled data source. If it lives on many nodes, parallelization may be employed to speed up data access. In most cases, however, grid exploitation is not an issue for 3rd party data source coupling.

System Integration

Data Source Coupling exclusively comprises "core" DILIGENT services.

5.9 Content Management: Change Tracking

Description and Priority

Change Tracking is another "Content Management" functionality which is responsible to (1) keep track of modifications on data managed by DILIGENT and (2) inform interested services (and users) of observed changes. Depending on the capabilities of the storage nodes and any coupled 3rd party data source, change tracking may either by conducted by triggering mechanisms which automatically "fire" upon changes or polling techniques which check relevant document in short intervals. Interested services must register for each document to be observed and provide an interface for notification upon change. The data dictionary must also maintain a list of services observing each document (represented by an URI).

This functional area comprises the UCs:

- **Register Service for Document** will register a specified service for a document identified by an URI. It (1) accesses the data dictionary to register the service (included partial functionalities of "Maintain URIs") and (2) adds an observer for this document.

- **Add Observer** adds an observing mechanism for the specified document. It is the generalization of triggering-based and polling-based observers (see there). Obviously, an observer only needs to be set up for a document, if this document is not yet observed by another observer instance.
- **Start Document Polling** initiates a polling-based observation of a document (primarily intended for 3rd party data sources with limited capabilities).
- **Add Trigger for Document** sets up triggering mechanisms to keep track of changes in documents hosted by DILIGENT storage nodes that are not supposed to bypass the official way of document changes (see there). It is up to the architectural design to make the decision, if we permit other ways of changing a document. In this case, other trigger mechanisms must additionally be employed (like database triggers and the like).
- **Observe Document** performs the actual document observation (by polling or triggering). That is, it is either invoked by content changing functionalities (see there) or invokes document access functionalities (see there) in polling cycles. Upon change, it will notify the registered service and needs to access the data dictionary to do so. Upon document removal (special kind of document change), it stops the observation of that document and notifies all interested services of this terminal document change.
- **Notify Service** does the actual notification of a service that has registered for a (changed) document. Our diagram also contains a link to an actor that must be complemented or substituted by a link to a service. Notice, that a service that registers for document change tracking must provide a calling interface permitting the "Notify Service" functionality to actually perform the notification.
- **Unregister Service for Document** removes a service from a document's list of change-tracking services as maintained by the data dictionary. Existing observers can be shut down, if no other service is keeping track of this document.
- **Remove Observer** does the actual removal of an observing mechanism from a document. It is the generalization of revoking triggering-based or polling-based observer mechanisms. It ends the observation of a document and can only be invoked if no service remains to be notified upon changes on that document. Clearly, this requires another lookup in the data dictionary.
- **Remove Trigger for Document** revokes a triggering mechanism to observe a document. This functionality is primarily intended for dedicated DILIGENT storage nodes that are supposed to support triggering mechanisms. It may also be applied to any 3rd party data source, which provides similar techniques.
- **Stop Document Polling** ends a polling mechanism observing a document. Unlike "Remove Trigger for Document" it will not just notify a triggering mechanism of the underlying storage system but will explicitly terminate a cyclic access of the observed document. As we assume DILIGENT storage nodes to provide triggers, polling will predominantly be employed on coupled 3rd party data sources.

Requirements

Change Tracking answers the requirement of document observation and notification.

Constraints and Assumptions

Change tracking rests upon some basic functionalities, which we implicitly assume to be given. In detail, a triggering mechanism will be built into the "official" document changing functionalities of Content Management. This would actually suffice our needs, if we can guarantee that no DILIGENT service (or even external service) bypasses Content Management to alter the data on its own. Otherwise, the underlying storage system in

DILIGENT storage nodes must either provide triggers (which is true for most database systems) or permit a frequent polling of a document.

Furthermore, we will need to provide a standardized interface to services that register for document observation. This interface will permit the notification upon change in the observed document.

UML Diagrams

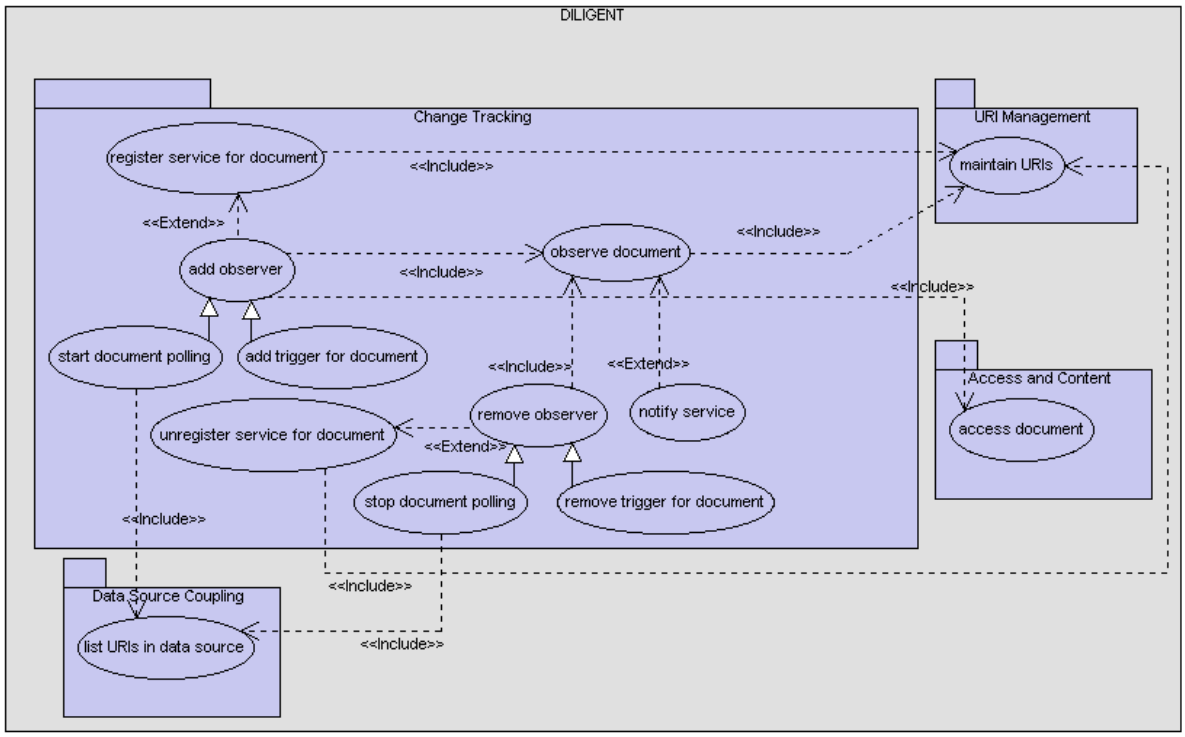

Figure 34: Change Tracking (use case diagram)

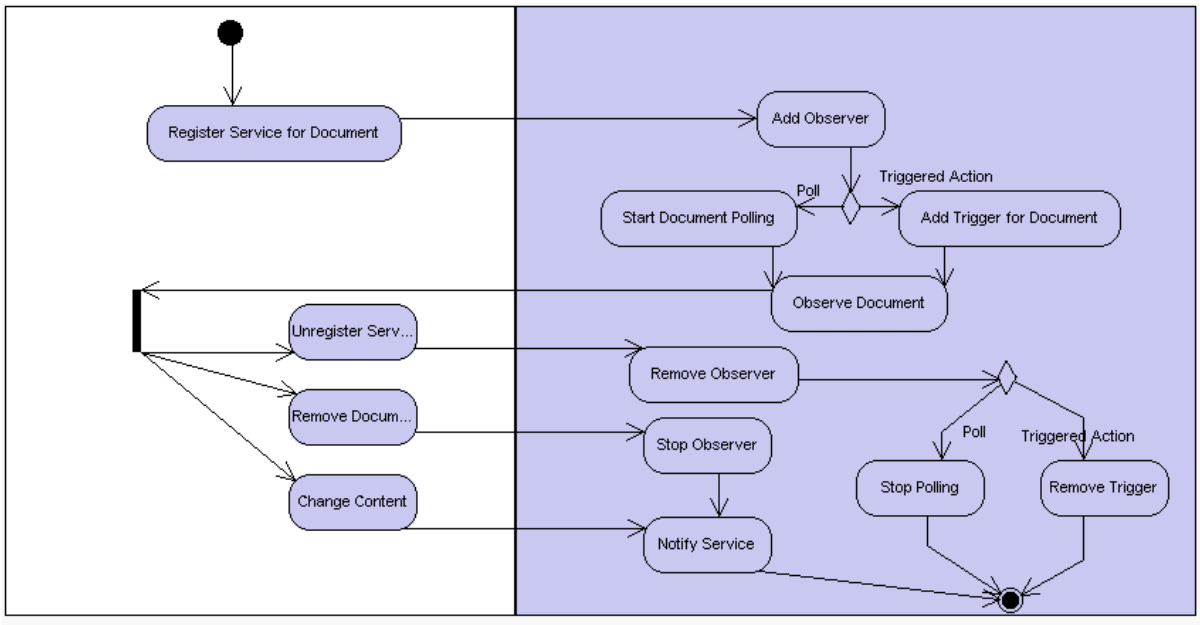

Figure 35: Change Tracking (activity diagram)

Grid Exploitation

In particular, the "Observe Document" UC might turn out to be computationally intense. That is, if polling needs to be employed, a document needs to be accessed in short intervals

to permit a timely detection of changes. Trigger-based observation might be computationally intense, as well, but is subject of (1) the underlying storage system or (2) the "Change Document" functionality (see there). As observation of a document in independent of observing another document, this task can be parallelized taking advance of the grid.

System Integration

Change Tracking exclusively comprises "core" DILIGENT services.

5.10 Content Management: Content Security

Description and Priority

The system allows the user to integrate security mechanism to protect the content of the provided media types. The user can decide of the type of security characteristics: between integrity, authenticity and traceability. The system provides the two security mechanisms: watermark and cryptographic solutions to cover the necessary security characteristics.

The integrity of the media is controlled by the integrity watermark. The authenticity of the media can be checked with an authentication watermark and the traceability with a fingerprint solution. The cryptography is an additional solution to provide a secure transmission and usage of the data.

The security mechanisms are offered as services to select standalone or combinations of the cryptographic and watermark solutions. The Content Security Management System manages the necessary processes or elementary services of the security service. With a necessary set of service parameters the processes are done to add the security mechanism to the media. To add the security mechanism in the right order a service map order the necessary service processes.

After the diligent user has selected the security mechanism the Content Security Management System starts the selected security mechanism after asking of the best parameter sets. A service structure map orders the necessary elementary services of the selected security mechanism to the right order. After ordering the elementary services are started and added to the DILIGENT media. For example an order of elementary services for a media can be decoding of the media, embedding of a watermark, encoding of the marked media and encrypting of the encoded media. The new encrypted media is send back to the DILIGENT user. The media decoding process decodes the media to get the necessary data for the watermarking process, for example in audio the frequency values of samples and in video and images the pixel values. The watermark embedding process adds a predefined watermarking scheme to the decoded media values. The encoding process encodes the marked media values back to a marked media and the encryption process encrypt the new marked media with a pre-selected key.

Requirements

The add security mechanism the user has to select the kind of mechanism. The system has to know the list of services to integrate the security mechanisms. To execute the services in the GRID system the Content Management System needs to list of GRID computers to distribute the services over the GRID system.

Numbers

The system will be required if the user starts requests to add the security mechanism

Constraints and Assumptions

The constraints exist between the security solutions and the type of media. Not every security solution is usable of every media type.

UML Diagrams

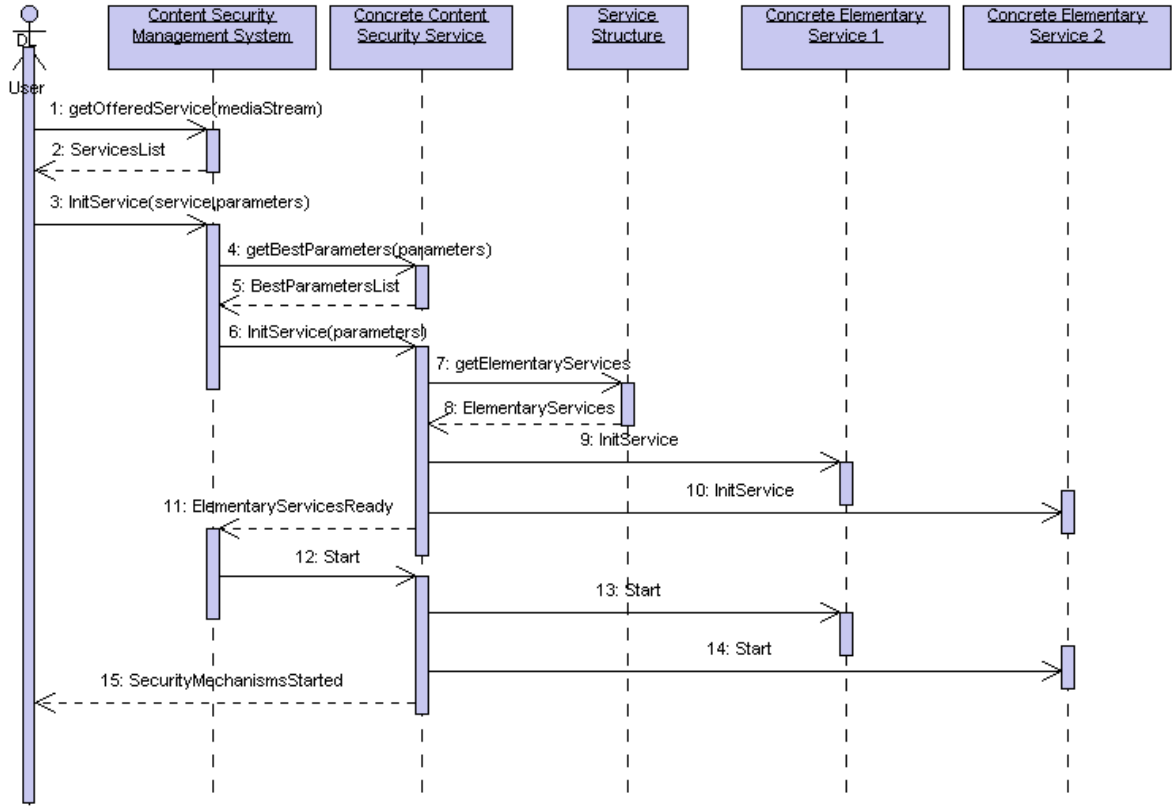

Figure 36: Content Security Management (sequence diagram)

5.11 Metadata Management

For metadata management in DILIGENT, we have identified the following functionalities from the user requirements of ARTE and ImpECt:

- Collection/Content related metadata
  - Basic DL metadata functionality
    - Insertion/deletion of content objects (and associated Metadata)
    - Update Metadata
    - Generation of Metadata
  - Collection Metadata Management
    - Importing collections
    - Transformation of Existing metadata
      - Definition of transformation rules
      - Application of defined rules
  - VDL- specific functionality
    - Integration of a collection (and its metadata) into a VDL
    - Integration of a subset of a collection (a view on a collection) and its metadata integration
- Service Metadata Management
  - Management of domain specific service taxonomy
  - Deployment/Un-deployment of Services (and related metadata)
  - Generation of Metadata for a DILIGENT Service

- Update Metadata

General notes about Metadata Management in DILIGENT

A typical digital library contains collections that are made available to the users via a mediation process. In this mediation process, the user expresses requirements, possibly to retrieve a subset of digital objects in a collection. The user might further wish to explore more than one collection, or use services to process the output of a object retrieval to generate new objects. The role of metadata can be in improving these processes, and more accurate.

Having developed initial ideas about what metadata needs to be stored, we can now better understand how the architecture needs to be. Understanding how EGEE currently handles metadata allows us to make this picture more concrete.

What Metadata can be used:

(i) **User Management**: Identification and Authorization data, Skills (themes, areas of expertise), Preferences, User context (Tasks, Roles, Relationships with other people)

(ii) **Service Management:** Semantic data (Ontological references for input output), Quality of service (Performance, efficiency, Computational resources available, Operation time, Information from service provider, User ratings, Availability)

(iii) **Object Management:** Indexing Data, Annotations, Annotation Schema, Source Schema (Syntax), Dublin Core (Who, where, etc.), Ontological Annotations (Semantics), Versions, Size, Cost

One important aspect is replication of metadata. Since DILIGENT is a distributed system which is based on an underlying distributed middleware, replication is an advantage as well as a necessity. Many replication schemes can be used, however, at this stage we use the replication strategies provided by the EGEE middleware.

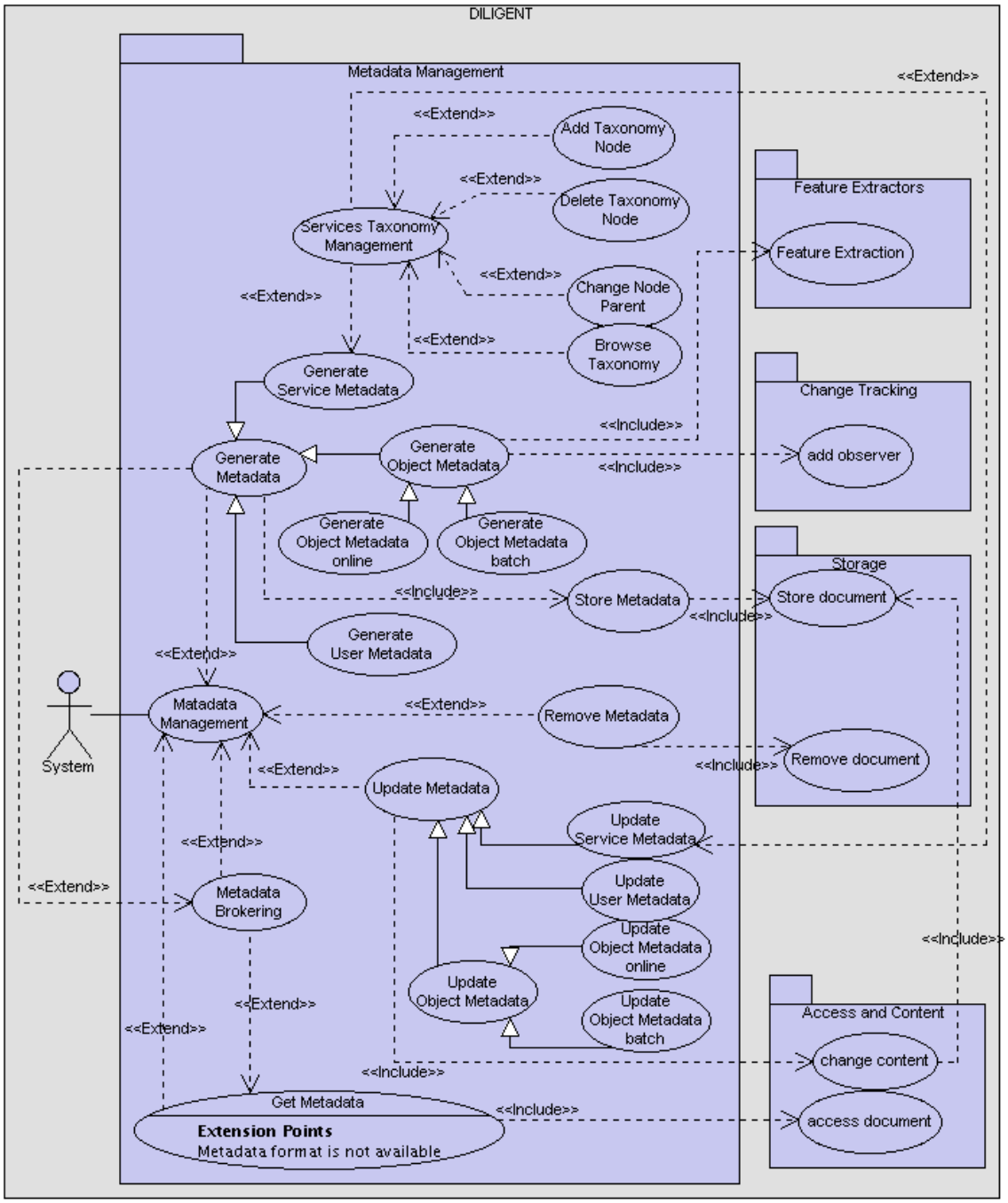

Figure 37: Metadata Management (use case diagram)

5.11.1 Generate Metadata

Description and Priority

Metadata generation is an important part of search and index functionalities provided by DILIGENT. This functionality is required when importing new objects or registering them. The imported objects may or may not have existing metadata. While the metadata management itself does not provide functionality to *generate* metadata, it provides the underlying infrastructure to update and consistently maintain derived metadata about objects. A typical example of this is feature extraction, which generates *metadata* about image objects processed. The generated metadata now has to be structured, managed and associated with the original content that it refers to. Since this is the overall task of the

metadata management, this functionality can be used to manage derived metadata in a fashion similar to descriptive metadata.

Following the same line of argument, similar functionality can be extended to user management and service management, to use the functionality of managing structured data for purposes of user profile management and service management. Service management also can benefit from some taxonomical arrangement of service offering (see Service Metadata Management)

Requirements

It should be possible to locate an imported object and the associated location in the metadata catalogue. Also, there should exist a service that can be invoked to generate metadata for this class of objects.

Numbers

This functionality is supposed to be infrequently required (few times a day), e.g. whenever new content is added to the system.

Constraints and Assumptions

The user invoking this functionality should have the requisite access rights, e.g. importing new resources and reading & writing to the content and metadata storage subsystem.

UML Diagrams

This use case is further sub divided into 3 use cases:

- Object Metadata Generation
- Service Metadata Generation
- User Metadata Generation

Grid Exploitation

Since this will be a frequently executed functionality, the GRID should be used to do the activities involved on multiple nodes.

Testing Issues

Batch import of a collection (e.g 10 pdf documents, 10 images, 10 text documents) and testing if correct metadata are generated.

For testing the throughput (number of resources served per minute), random files can be generated and time noted to generate metadata is noted.

5.11.2 Generate Object Metadata

Description and Priority

This functionality is used when new metadata about an object need to be generated. A DILIGENT object is an abstraction for any form of content object that the underlying content management can support. When a new object is added to the DILIGENT system, either as a part of a bigger collection or individually, the underlying content management component sends notification to the metadata management component, which has subscribed to this notification. The next step required is to look, or query for the metadata associated with this particular object. One approach to ensure that all objects have some minimal descriptive metadata available is to require metadata in a so-called Minimal Core metadata Schema for every object inserted. This minimal metadata will ensure that the searching and indexing services can rely upon a minimum schema to index for better searching. Additional metadata can be also be specified in a Additional Metadata Schema which will be a much larger schema catering to a much larger set of particular metadata attributes.

Object Metadata here could be Dublin core data (who, where, etc.), indexing information, object details like size, version etc. A new object being added is added to the Content storage and an entry is created in the metadata catalogue. The metadata is generated by a service, possibly different services for different kinds of objects, and one service providing the fixed interface:

generate-metadata(Object o, Type t, MetadataCatalogueLocation l)

The generation process could be online, or batch, depending on the object. In cases where processing time per unit are high (e.g. feature extraction in images), this process could be batched.

Numbers

This functionality is supposed to be frequently required (a few times a day), e.g. whenever a new object is added to the system.

Constraints and Assumptions

- Every object has a Type associated with it. This could be standard data types like Text, Image, BinaryFile, etc.
- For every Type of object, there exists a strategy (a set of services) to analyze and generate metadata.
- There is an agreed-upon schema for metadata for each type of object.
- If a new type of Object is used in the system, or a new metadata scheme is used, the services to do the necessary processing are also deployed.
- The inserted object provides some metadata in an XML format, at least in part, which conforms to a DILIGENT Core Metadata Schema.
- Every object added to the DILIGENT system has atleast some metadata associated with it. This metadata should ideally be in the DILIGENT Minimal Core Metadata Schema (which has to be concretely defined).

UML Diagrams

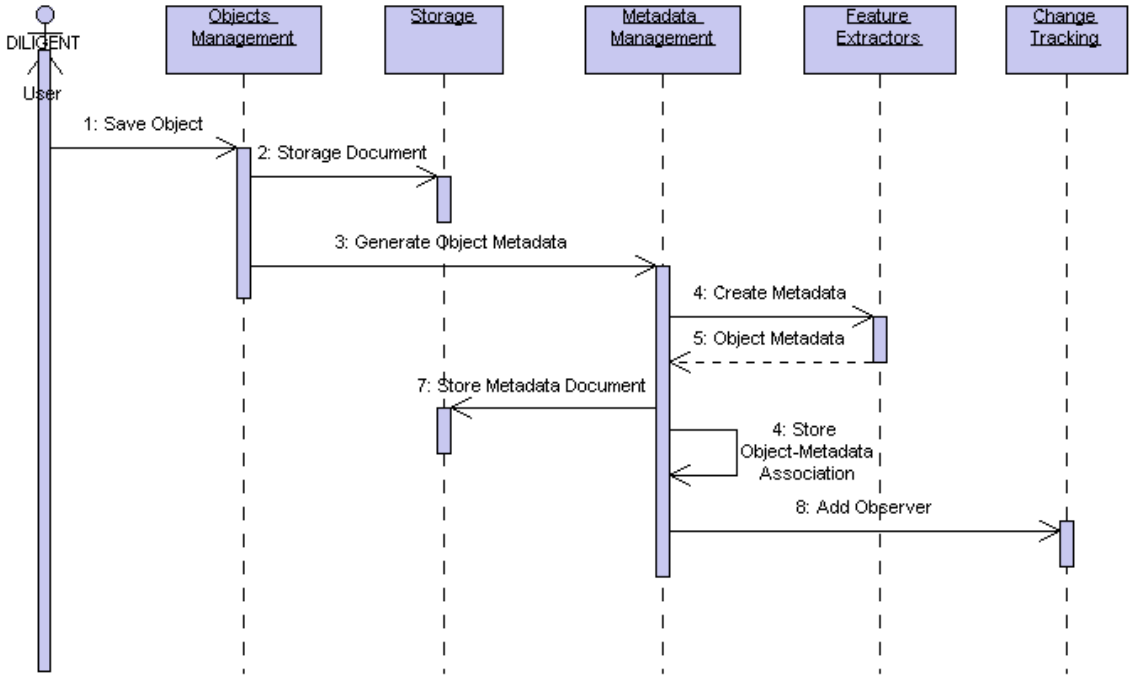

Figure 38: Generate Metadata for a new Object - online processing (sequence diagram)

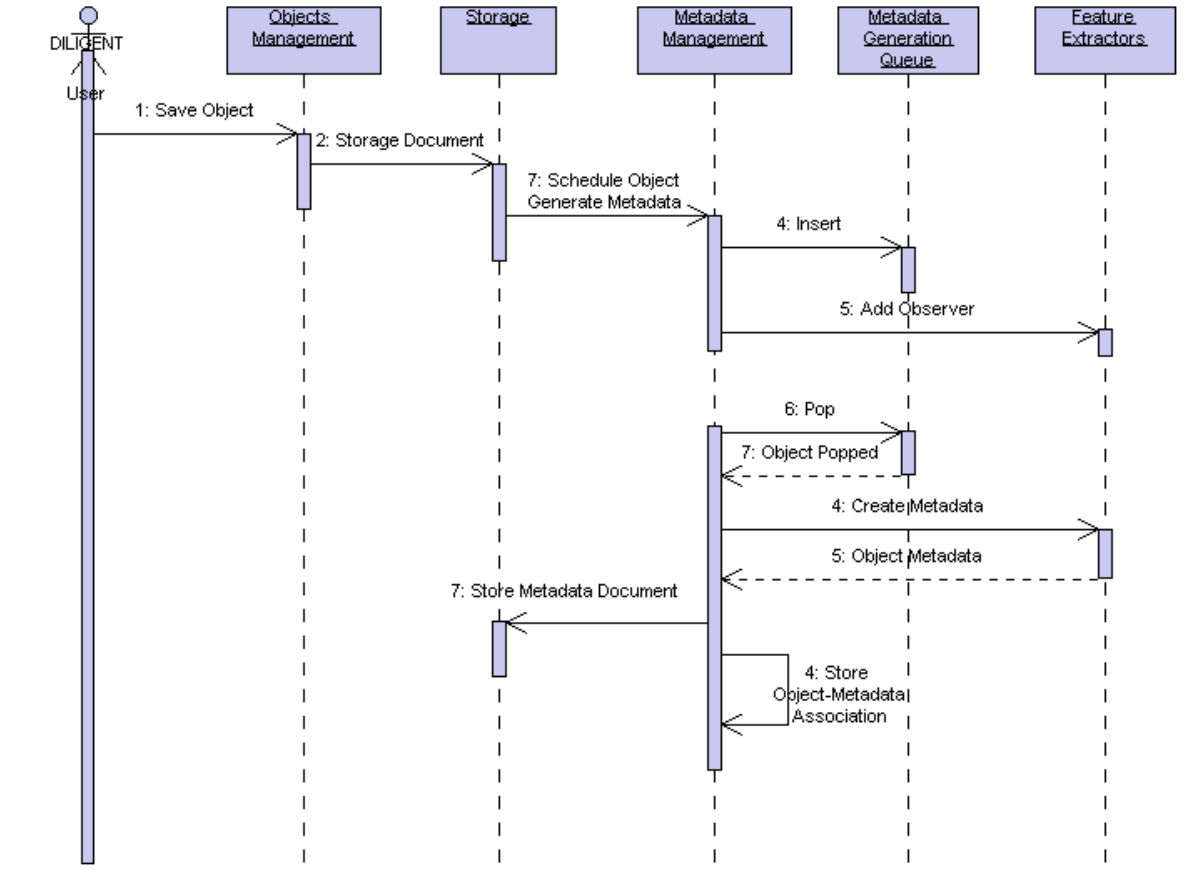

Figure 39: Generate Metadata for a new Object - batch processing (sequence diagram)

Grid Exploitation

For online processing, the grid node with a currently low usage can be called. However the network transfer overhead of the object being sent to this new location has to be balanced with the waiting time of closer nodes.

For batch processing, parallel processing of the batch is possible to reduce the overall time required.

System Integration

Metadata generation for objects integrates into the workflow of a new object being imported into the DILIGENT system.

Testing Issues

Batch import of a collection (e.g 10 pdf documents, 10 images, 10 text documents) and testing if correct metadata are generated.

For testing the throughput (number of resources served per minute), random files can be generated and time noted to generate metadata is noted.

5.11.3 Update Object Metadata

Description and Priority

This functionality requests update of metadata when the underlying content management sends an update notification. Typical scenarios of usage of this functionality include updating of data itself, a newer version replacing a previous one or re generation of metadata using a newer service or with different parameters. While its not planned at this point in time to allow the user to directly manipulate metadata, this could be another possible scenario.

At the end of updating process of metadata, checking consistency and syntax and type has to be performed and in case of error, the process has to be rolled back.

Requirements

See 'Generate Metadata'

Numbers

Not very frequently (a few times a week)

Constraints and Assumptions

See 'Generate Metadata'

UML Diagrams

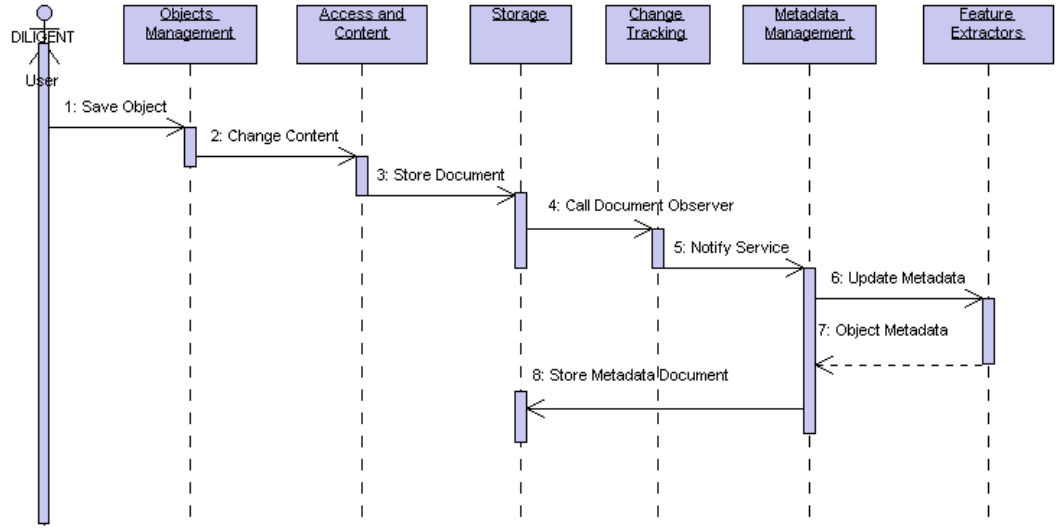

Figure 40: Update Object Metadata - online processing (sequence diagram)

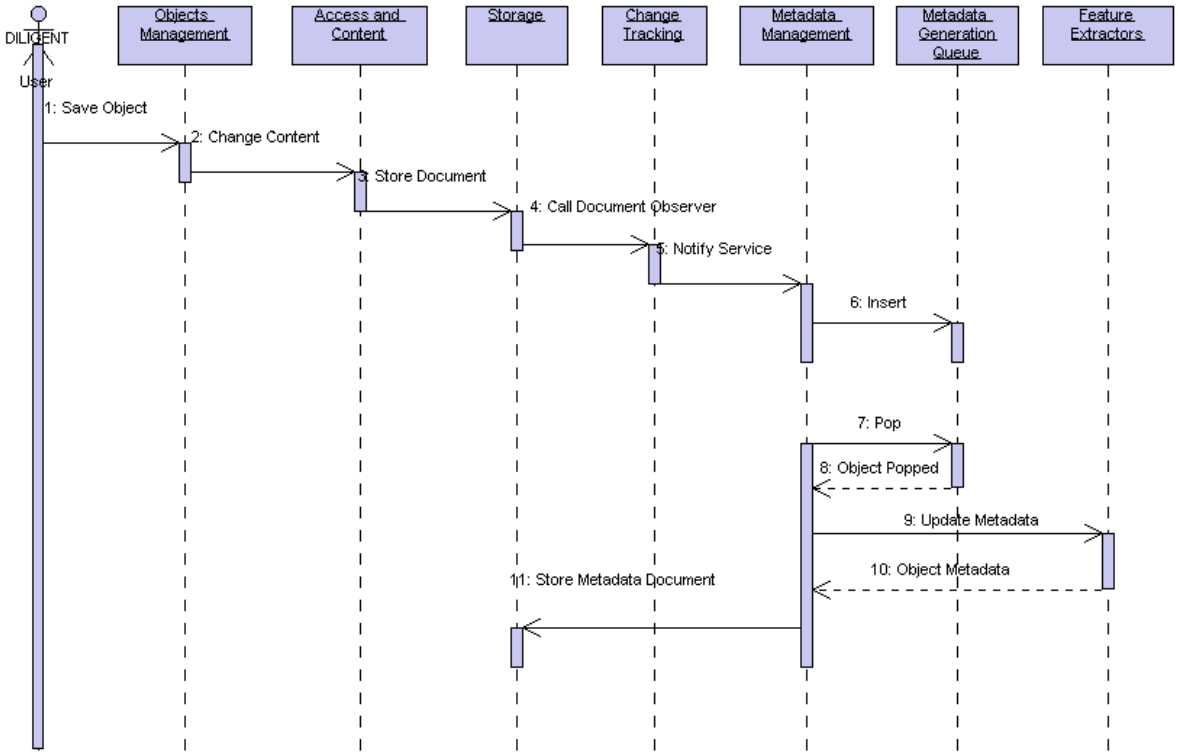

Figure 41: Update Object Metadata - batch processing sequence diagram)

Grid Exploitation

See 'Generate Metadata'

5.11.4 Generate Service Metadata

Description and Priority

Services in DILGENT form an important building block in the overall architecture. Different nodes in the DILIGENT offer services that are required for various digital library functionalities. For choosing the right service for a particular functionality, metadata about services could be used for brokering/recommendation. Also different nodes for faster processing in a parallel manner can offer the same services. This also means that the best service to choose depends on the individual 'context' of the service consumer (choosing nodes which are closer, node which are more trusted, or the most powerful etc). Metadata about *instances* of services can be useful in such a scenario.

Metadata about services can be of various types

1. Syntactic (input/output of a certain type): this is typically stored at the service registry
2. Semantic information
    a. Information about inputs or 'resources' used by a service (e.g. Service A acts on JPEG images)
    b. Information about the output and workflow of the services (e.g. Service B returns Boolean result of weather a face has been detected in input Image)
3. Deployment Metadata
    a. Service deployed at node a (geographic location, network speed, processing power)
4. Performance and tractability (expected runtime of a job, average latency or service response, average uptime, tractability of the service provider)

From this list, we can see how a user can be supported in a typical diligent scenario

1. "I want a service to manipulate images."
2. "I want a service that does feature detection."
3. "I want a service to detect features, but I want to choose a node which is closer to me."
4. "I want a service which will not be unavailable after 2 hours of processing ... I want a service which will complete this 8 hour batch job."

There are many additional types of metadata that could be used for service description. At this stage we point out the observation that a multi-dimensional service metadata description is required and the usage of this metadata is enhanced and automated service selection.

To order the service offerings by various parameters offers a first level of filtering. Associating services with nodes in the taxonomy related to the types of inputs and outputs they can handle.

The second step could be associating services with some process classification.

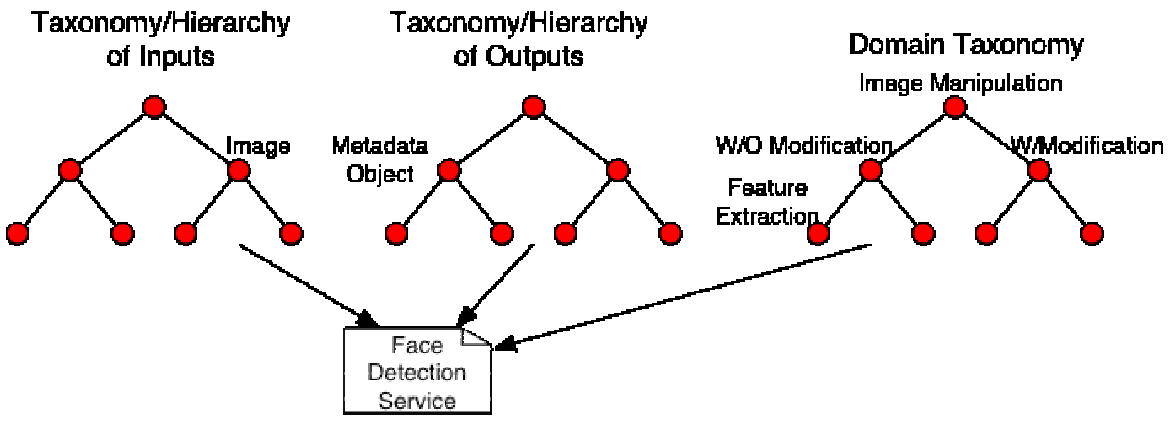

Figure 42: Metadata Services Taxonomy example

In the above figure, we show the typical taxonomical associations of a face detection service. This service takes an Image as in put and returns a metadata object as output. For this service, the input parameter is classified under 'image' category and the 'output' parameter is classified under 'Metadata'. Further, this service is classified under the feature extraction node in the image manipulation process tree.

This functionality is provided to management the service taxonomy. At this point of time, DILIGENT does not plan to include automatic generation of such taxonomies. Instead taxonomies created by domain experts could be used to model the processes in the domain. However, if services are available to generate such taxonomies, they could easily be interfaced with the functionality described here to automatically add new nodes.

Numbers

This functionality is supposed to be in-frequently required (a few times a week), e.g. whenever a new service is added to the system.

Constraints and Assumptions

For metadata management, there has to be service registry where the location of a service can be retrieved.

Also every service should be registered when it is added to the infrastructure. At this time, services should be classified for types of input and output parameters. Similar classifications can be done for workflows and processes implemented.

See also "Metadata Generation".

UML Diagrams

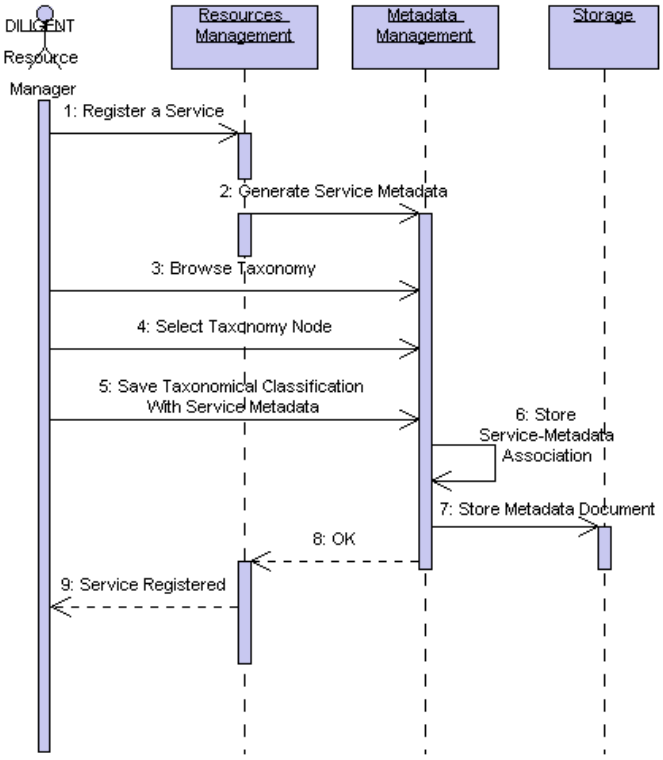

Figure 43: Generate Service Metadata (sequence diagram)

Grid Exploitation

Since the same service can exist in multiple locations, individual instances of services need to have their own associated metadata. This might also require maintaining multiple service–metadata repositories, or a metadata repository at every node extending the local service registry and feeding into the larger composite service registry.

System Integration

Services form an important part of the GRID infrastructure. Generating metadata is an important step of meta-data based filtering of services and choosing the one best suited to the requirement of the user in a given situation.

Use Stories

See "Metadata Generation".

Testing Issues

See "Metadata Generation".

Related Non-Functional Requirements

See "Metadata Generation".

5.11.5 Management of Service Taxonomy: Add Taxonomy Node

Description and Priority

This functionality simply adds a node to service taxonomy.

5.11.6 Management of Service Taxonomy: Delete Taxonomy Node

Description and Priority

This functionality deletes a node in a given service taxonomy.

5.11.7 Management of Service Taxonomy: Change Node Parent

Description and Priority

This functionality is used when the parent of a node has to be changed.

5.11.8 Remove Metadata

Description and Priority

This functionality is invoked when an object is deleted from the storage and requires deletion of metadata as well. When a DILIGENT object is deleted from the storage layer, the content management component sends a notification and associated metadata and the associations themselves can be removed from the repository. In particular, all replicated copies of this metadata have to be removed, and the indexing services have to be notified of this change as well.

5.11.9 User Metadata Generation

Description and Priority

Metadata about users will consist of static information like Name, date-of-birth, etc and dynamic information like skills and interests. Static information can be collected when the user registers to use the system, while for the dynamic data, a consistent process of user monitoring and analysis will have to be performed.

Like services, metadata about users will tend to be multi dimensional. Besides the challenges in identifying a sufficiently powerful representation for user metadata, we also need to keep in mind that user 'profiles' needn't be unique, i.e. the same user might have different values for the same type of metadata depending on the digital library. Since a user can have different interests and skills depending on the 'context' and the application, either different sets of user metadata (or profiles) need to be maintained, or a composite profile

Requirements

For metadata management of users, there has to be service registry where the location of a service can be retrieved.

Also every user should register before the user can access the infrastructure. At this time, classifications of users (administrator, DL creator, content manger etc) for access management can be implemented separately from metadata management.

Numbers

This functionality is supposed to be frequently required (several times a week), e.g. whenever new users are added to the system.

Constraints and Assumptions

See "Metadata Generation".

Grid Exploitation

The underlying Grid infrastructure maintains a centralized user database. Therefore the metadata repository for users could be centralized as well and linked to the main user database.

System Integration

User metadata generation is an important part of the user personalization and will be incorporated into the workflow of user registration. In addition, periodic updates to the user metadata by the will be done by personalization service.

Use Stories

See "Metadata Generation".

Testing Issues

See "Metadata Generation".

Related Non-Functional Requirements

See "Metadata Generation".

5.12 Annotation Management

This Section specifies a set of medium-priority functionalities for the integrated management of annotations for digital objects available on the DILIGENT platform, primarily multimedia content objects. The assigned priority reflects the key role of the identified functionalities in effectively supporting a localised and yet critical subset of DL activities based on remote, international, and invitational collaboration between qualified DL users. In the ARTE scenario, for example, annotation management supports Course Management and Workshop Management activities, particularly those associated with Exhibition Catalogue Management.

The following table lists the functionalities identified from the user requirements and associates each functionality (or set of functionalities) with an indication of the architectural layer which should be responsible for its provision:

Functionality	DILIGENT Layer
View Annotation	ASL[5]
Edit Annotation	ASL
Validate Annotation	DLL[6]
Translate Annotation	DLL
Post Annotation	DLL
Retrieve Annotation	DLL
List Annotation Stubs	DLL

Functional descriptions for each of the identified functionalities are reported in the next Subsections, while some generic observations follow immediately:

- **(on interpretation & scope)** Annotations are here intended as manually authored, subjective, context- and task-dependent forms of metadata about objects persistently stored on the platform; semi-automatically authored annotations are not currently among the user requirements but may be supported at a later stage. Furthermore, no early constraint is imposed on the semantics of annotated objects. Rather, the interpretation of 'objects' is assumed to be contextual to the annotation process, in that, besides multimedia content objects, services, annotations, and other forms of metadata may be themselves objects worth annotating. While the requirement for *meta-annotations* is not directly derivable from the use cases, it is considered desirable for the flexibility of the platform and it is not expected to excessively raise the complexity of the system.

[5] Application Specific Layer (see the DoW for details).
[6] Digital Library Layer (see the DoW for details)

- **(on structure)** Annotations must be structured in accordance with the schema and, where applicable, the controlled vocabularies associated with an annotation model, or *annotation ontology*. The ontology must at least include properties about:
  - the authoring of annotations (e.g. annotation ID, author ID, ingestion date, last modification date);
  - the binding of annotations to objects (object ID, anchor ID, collection ID);
  - the semantics of annotations (e.g. annotation type);
  - the content of annotations (e.g. annotation body, body type, annotation language);
- **(on representation)** Models and formats for persistent storage and networked exchange of annotations shall follow platform-wide guidelines and policies. At least for data exchange, but possibly also for persistence requirements, the use of XML and related management technologies is foreseen.
- **(on interoperability)** Reliance on annotation ontologies is expected to introduce interoperability issues within any heterogeneous platform. Of course, such issues shall be dealt with in alignment with more generic Metadata Management functionality, and in fact with platform-wide strategy and policies. Nonetheless, it is anticipated that some tension between, on the one hand, the flexibility required to support diverse *application-specific annotation ontologies* at the ASL and, on the other hand, the need to offer ontology-aware, cross-application Annotation Management functionalities at the DLL, will become manifest within the platform.

  At this stage, this tension may not be an immediate reflection of user requirements, and a single, well-crafted ontology may well satisfy both user communities. The problem, however, is expected to arise as soon as the platform opens its services to other user communities. For this reason, an early approach to interoperability is recommended.

  To solve the problem for the purpose of these specifications, no explicit reference to specific annotation ontologies is made in the following. Rather, ontology-aware functionality is interpreted as a set of functionalities which span across the ASL and the DLL of the platform. When interpreted at the ASL, the functionality should be understood as relying on application-specific ontologies. When interpreted at the DLL, the functionality should instead be understood as relying on a yet-to-be-defined *canonical annotation ontology*. Translate Annotation functionality is then required at the DLL to map between instances of application-specific ontologies and semantically equivalent instances of the canonical ontology. In particular, it is required that the canonical ontology be *extensible* so as to accommodate the properties of any application-specific ontology that are not mandatory in the canonical annotation ontology.
- **(on authorisation)** All annotation management functionalities are available to specifically authorised DL users. In the ARTE scenario, for example, authorisation of users is predicated on their authentication as ARTE members and is to be enforced in respect of the authorisation policies defined through Course Management and Workshop Management functionalities and, in turn, through Collection Management and User Management functionalities (all of which are specified elsewhere in the Use-case Model). In the context of Course Management activities, in particular, ARTE members must act in the specific role of Collaborators.
- **(on load)** Load on Annotation Management functionalities is expected to be intermittent within the system, and to intensify throughout the duration of related events, such as courses and workshops across ARTE DLs. For Course Management activities, in particular, the initial expectation is of a fairly contained low and a

limited number of users. Of course, load and number of users are directly proportional to the number of concurrent course and workshop events, an estimate of which has yet to be provided, if it may, from the user communities. In general, however, load and users of DLL functionalities are expected to be higher than those of ASL functionality, as the former concurrently support the activities of an unspecified number of DILIGENT DLs.

- **(on system integration)** At this stage, the focus is on Annotation Management functionalities which appear to be required across the platform at large. A possible mapping of those functionalities onto different services is nonetheless sketched. Following the approach to interoperability discussed in the previous point, functionalities are mapped onto capabilities of the Annotation Service in WP1.3 when they are interpreted at the DLL (and thus in terms of the canonical annotation ontology), and onto capabilities of application-specific extensions of that service when they are interpreted at the ASL (and thus in terns of annotation ontology specific to the application). Such service extensions, in particular, delegate ontology translation to ASL services that implement Translate Annotation functionality and then use the translated annotations to interface the basic Annotation Service. In any case, a precise specification of service capabilities is expected to emerge in later, design-oriented phases of the iterative specification process.

  Finally, no eager assumption is made on the exact nature of the relationship between Annotation Management functionality and the related functionalities of Content Management and Metadata Management. In particular, given the assumption that annotations are context- and task-dependent forms of metadata – and are thus subject to similar core management operations – it remains yet unclear where, and even if, boundaries between related functionalities and services should be drawn.
- **(on GRID exploitation)** As currently understood, none of the Annotation Management functionalities appear to raise processing or storage intensive requirements. Testing, however, may reveal high throughput for those functionalities that pertain to the DLL layer and thus may serve an unspecified number of DLs and/or an unspecified number of annotation-related activities within each DL. In any case, however, the reliance of Annotation Management functionalities on the GRID platform is justified on the basis of the good properties normally associated with widely and heterogeneously distributed platforms.

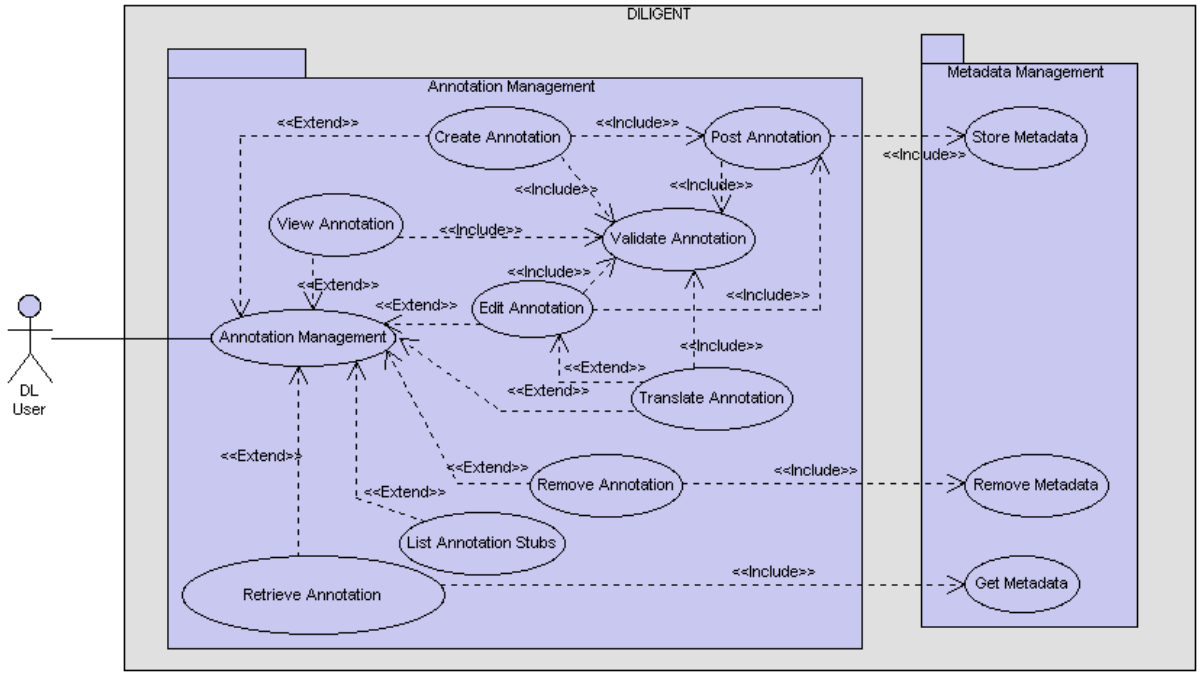

Figure 44: Annotation Management (use case diagram)

5.12.1 Edit Annotation

Description and Priority

Medium-priority, ASL functionality for the authoring of annotations.

Requirements

The functionality provides an editing environment (e.g. an input form) structured in accordance with the application-specific annotation ontology associated with the DL – or, if required at a later stage, with particular collection or collections within the DL – and it produces an instance of that ontology. Optionally, the functionality accepts an instance of the ontology and sets the defaults of the editing environment to the current values of the instance's properties. Specific requirements remain application-dependent and will not be discussed here.

Constraints and Assumptions

The functionality includes Validate Annotation functionality to ensure that input and output annotations are instances of the supported annotation ontology.

More generally, we identify a need for metadata quality assurance within metadata generation workflows and accept that such workflows are largely outside the scope of Annotation Management functionality.

Specific constraints and assumptions remain application-dependent and will not be discussed here.

UML Diagrams

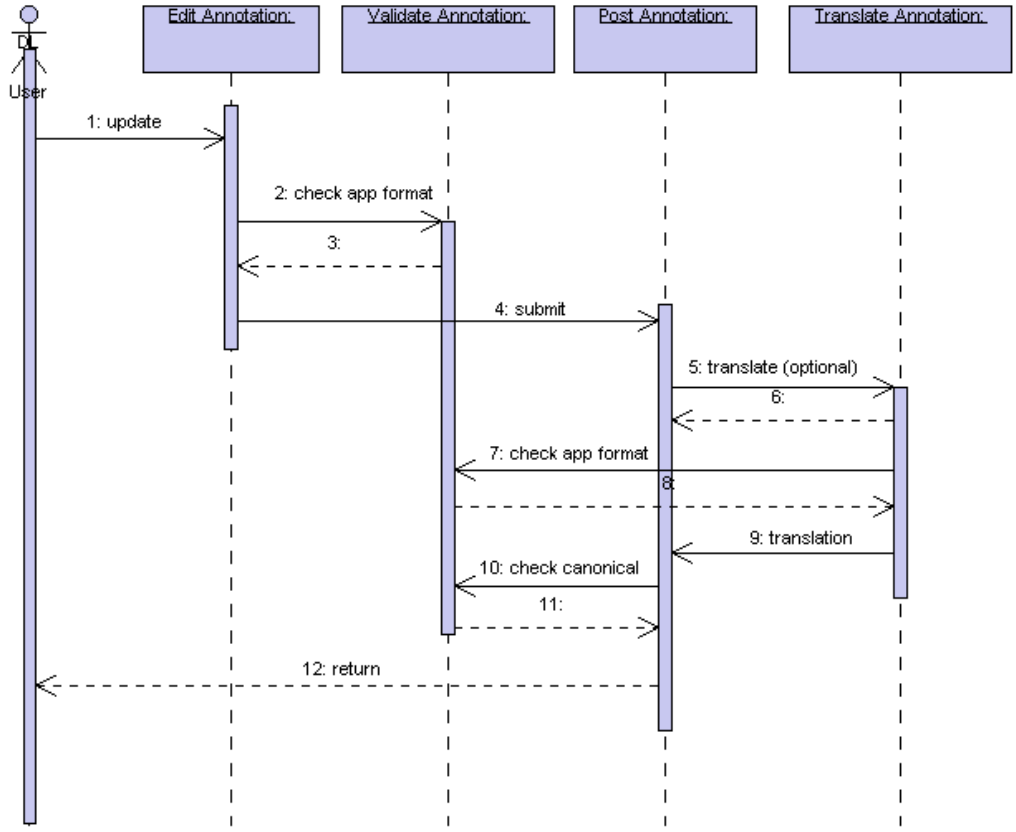

Figure 45: Edit Annotation (sequence diagram)

5.12.2 View Annotation

Description and Priority

Medium-priority, ASL functionality for the rendering of annotations.

Requirements

The functionality requires an instance of the application-specific ontology associated with the DL – or, if required at a later stage, with particular collection or collections within the DL – and it provides a visualisation environment structured in accordance with the ontology in which the values of all the properties of the instance are conveniently displayed. Specific requirements remain application-dependent and will not be discussed here.

Constraints and Assumptions

The functionality includes Validate Annotation functionality to ensure that the input annotation is an instance of the application-specific annotation ontology. Specific constraints and assumptions remain application-dependent and are not be discussed here.

5.12.3 Validate Annotation

Description and Priority

Medium-priority, DLL functionality for the validation of annotations.

Requirements

The functionality requires an annotation and validates it as an instance of a given ontology by matching the structure of the annotation against the schema associated with the ontology.

Constraints and Assumptions

The functionality results in an error condition if the annotation is not an instance of the supported ontology.

5.12.4 Translate Annotation

Description and Priority

Medium-priority, DLL functionality for the translation between application-specific annotation ontologies and the canonical annotation ontology.

Requirements

The functionality accepts an instance of a given application-specific ontology and returns a semantically equivalent instance of the canonical annotation ontology. Similarly, it accepts an instance of the canonical ontology and returns a semantically equivalent instance of the application-specific ontology.

Constraints and Assumptions

The functionality includes Validate Annotation functionality to validate the input annotation as an instance of the input ontology (canonical or application-specific) and thus propagates the errors that may result from that functionality.

5.12.5 Post Annotation

Description and Priority

Medium-priority, DLL functionality for the persistent storage of annotations within the platform.

Requirements

The functionality requires an instance of the supported ontology and persistently stores the annotation on the distributed DILIGENT platform.

Constraints and Assumptions

The functionality includes Validate Annotation functionality to ensure that the input annotation is an instance of the supported ontology (even when this validation is already provided by the Edit Annotation functionality).

At this stage, we do not distinguish between postings that result in the storage of new annotations or in the update of already existing annotation. Such distinction may instead be made on the basis of whether the value of the annotation ID is null or non-null. Accordingly, a non-null annotation ID must identify an annotation currently stored in the Platform.

The author ID contained in the annotation must identify a user with posting privileges and, under updates, one with update privileges. We assume that update privileges are granted exclusively to the author of the annotation. More flexible, group-based authorisation schemes may be envisaged at the cost of additional complexity within the system. For the time being, however, this is not identified as a requirement.

Finally, the functionality propagates the failures that may be reported by the Validate Annotation functionality.

5.12.6 Retrieve Annotation

Description and Priority

Medium-priority, DLL functionality for the retrieval of annotations previously stored within the latform.

Requirements

The functionality requires an annotation identifier and returns the instance of the supported annotation ontology identified by the identifier.

Constraints and Assumptions

The functionality results in an error condition if the input identifier does not identify an annotation currently in the persistent storage of the platform.

5.12.7 List Annotation Stubs

Description and Priority

Medium-priority, DLL functionality for the retrieval of stubs for all the annotations of a given multimedia object which have been previously stored within the platform.

Requirements

The functionality requires an object identifier and produces a list of the stubs of all the instances of the supported annotation ontology that annotate the object associated with the given identifier, if any such object exists in the persistent storage of the platform.

Constraints and Assumptions

Informally, an *annotation stub* is a yet-to-be-defined subset of the information contained in an instance of the supported ontology, which is used to: (i) provide users with an overview of the semantics of the annotation, and (ii) initiate and optimise application-level annotation management. The definition of a stub *must* include the identifier of the annotation and may include any other annotation property that is deemed necessary to support the two goals above. Due to (ii), in particular, the stub *must not* include the body of the annotation.

In particular, the functionality allows applications to inform their users of the existence and high-level semantics of all the annotations available for a given object before full information about individual annotations is opportunistically retrieved from the platform.

The functionality results in an empty stub list when the input object identifier is not recognised or when no annotations for the object are currently available in the platform.

5.12.8 Remove Annotation

Description and Priority

Medium-priority, DLL functionality for the removal of annotations previously stored within the Platform.

Requirements

The functionality requires an annotation identifier and a user identifier and produces an instance of the supported annotation ontology with the given identifier, if any exists in the persistent storage of the platform.

Constraints and Assumptions

The annotation ID must identify an annotation currently stored in the Platform.

The author ID contained in the annotation must identify a user with removal privileges. We assume that removal privileges are granted exclusively to the author of the annotation (see also Post Annotation functionality).

6 INDEX & SEARCH MANAGEMENT

6.1 Introduction

This functionality has been partitioned in four areas that group the functionalities of Index and Search management. These areas are:

- **Index Management** (see Section 6.2);
- **Content Source Description and Selection (CSDS) and Data Fusion** (see Section 6.3);
- **Feature Extraction** (see Section 6.4);
- **Personalization** (see Section 6.5);
- **Search** (see Section 6.6);

6.2 Index Management

Indexing serves to optimize search. In that respect, the indexing service is subordinate to the search service. Therefore it is mostly transparent to DILIGENT users, except for the performance boost and the management functionality invoked to control indexing: DILIGENT resource managers have full management control over indexing. (Other users, e.g. DL managers, may also control to some extent how indexing is applied to collections they own or manage.)

An index is associated with a collection of objects and enables efficient lookup of objects within the collection. The index is part of the metadata for the collection, and the association between the collection and indexes over it is maintained by the metadata service.

The index contains a set of records, which again consists of a set of fields according to an index description (schema). The field values are extracted[7] or derived from the objects in the indexed collection. This process is controlled by the metadata service. Depending on the data type, fields support various matching operations (e.g. simple comparison operations, wildcard matching, full text searches with various language-sensitive options). Fields can also be used for ranking (i.e. be used in a query time calculation of rank) or sorting of the result set. Finally, the result set consists of a set of records with fields either copied[8] from the index fields (e.g. object identifier) or derived from them (e.g. computed rank, summary of object with hit highlighting).

The basic index functionality is *lookup*: Given a *query*, a set of object identifiers matching the query (*result set*) is returned. Optionally, the query may specify ranking and/or sorting of the result set, and that only a limited number of results are to be returned. This functionality is invoked from the search service[9] when search (query optimization) locates an index which is applicable to the query search is evaluating.

The index service implementation receives content to be indexed from the content management and metadata services. Both the processing that produces the index field

[7] In a full-text index, the object content itself is one of the fields. It may be transformed in various ways depending on the matching options enabled, e.g. lemmatization (matching independently of inflexion forms of words) or character normalization, and for that reason the content must be copied into the index, although the object itself resides outside the index. Also, different context (such as titles, body text) can be handled by extracting content to different fields.

[8] Many types of fields cannot be returned as part of the result set, as they are not stored literally in the index.

[9] In principle, not from the search service itself, but components activated by the search service in order to carry out the query plan and running under control of the process management service.

values from the indexed object and the actual update and propagation of changes in the index structures may be time consuming. For that reason, it is not normally feasible to keep indexes perfectly consistent with the collections they are derived from. Thus, index update functionality must include mechanisms that permit some level of control of this inconsistency.[10] The basic framework for this is provided by the process management service.

The use case models from the scenario requirements touch on indexing in various ways, but mostly implicit:

- Indexing is the specific topic of the Index Management use case in the ARTE Scenario.
- Index Management extends Metadata Creation. (One would normally create an index over the new metadata.)
- All use cases involving search in some way also depend on and define requirements for index functionality. (Although search can be performed without an index, this is the exception, and in all the search cases it must be possible to use an underlying index.) In these use cases the user does not interact directly with the indexing subsystem, so they are not, strictly speaking, indexing use cases.
- Finally, there are the use cases for adding archives to the Diligent infrastructure or to a digital library. (Create/Redefine an ARTE DL, ARTE DILIGENT Management, Archives to be Added/Removed, Create/Reconfigure a DL.) Generally speaking, in order for an archive to be available for search, one or more indexes must be created when it is made available in DILIGENT or included in a DL.

We have handled this by factoring the functionalities covered by Index Management into separate extension use cases that extend both Index Management and the use cases managing archives.

6.2.1 Index Management

Description and Priority

This use case, taken from the ARTE scenario, is the only one specifically concerned with indexing. It is invoked from the other use cases whenever there is a need for an index to be created. The result of the use case is the creation of the index.

Functional Requirements

This use case is not very specific with respect to requirements, only that there is a requirement to be able to create an index to speed up search in order to support interactive queries. The use case, as described in the user requirements, only covers "one shot" generation of an index and does not include incremental index updates.

Numbers

There will be at least one index per archive included in the DILIGENT infrastructure, and it is probable that one or more new indexes will be created each time an archive is included in a DL (in order to cover the specific search functionality requested in that DL). The size of an index depends on the number and size of documents covered, but also very strongly on the features included in the specific index.

[10] Some partial consistency constraints may be fulfilled, such as: Changes are applied and become visible in the index in a well-defined, consistent order; new and old version of a changed object never appear together in the same result set, but at least one of them does appear. On the other hand, if there is a goal to maximize throughput or minimize indexing latency, it may be necessary to relax or altogether remove such constraints.

Constraints and Assumptions

See more specific use cases.

UML Diagrams

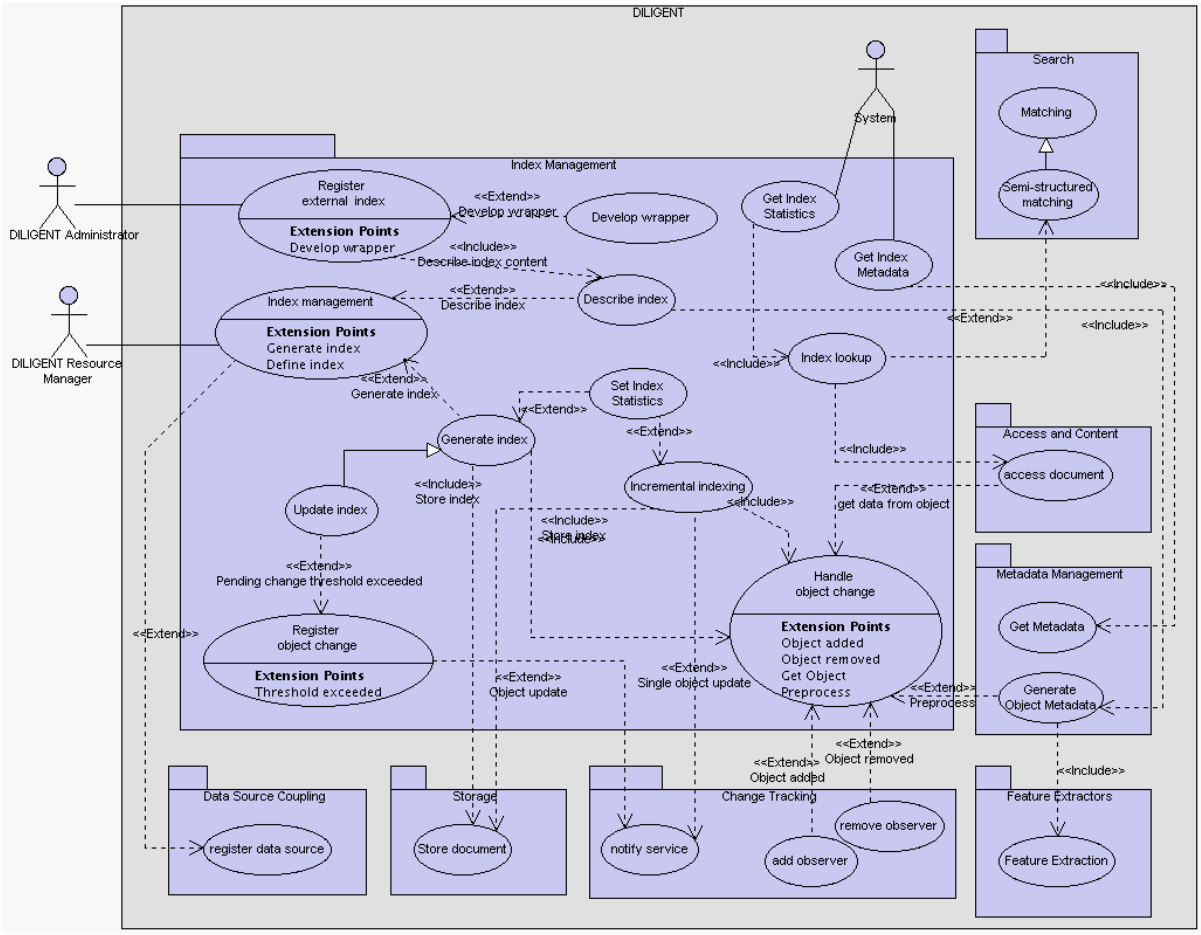

Figure 46: Index Management (use case diagram)

Grid Exploitation

See more specific use cases.

System Integration

See more specific use cases.

Use Stories

See more specific use cases.

Testing Issues

See more specific use cases.

Related Non-Functional Requirements

Most non-functional requirements actually arise in the specific use cases, see below. However there are some issues of a more "systemic" nature that don't belong to any one of these use cases.

Given the opportunistic nature of grid computing, consistency, durability and reliability requirements are challenging. The user requirements do not appear to include strong requirements in these directions, but that may partially be due to low awareness of the issue. In the absence of more specific requirements, we assume that medium level requirements apply:

- Indexes are considered as an optimization of search, and can in the worst case be regenerated from scratch based on the underlying data sources, the main limitation being the time and computational cost involved in index generation. Thus, no absolute requirement to index durability.
- Replicas and partitions of an index need not be perfectly synchronized; it is permissible (to some degree, to be defined) that objects are duplicated, missed or inconsistently ranked in result sets, in particular in large, frequently updated indices. That is, in the likely trade-off between index latency/freshness, lookup speed and availability on one side, consistency on the other side, consistency can be down-prioritized somewhat.

6.2.2 Describe Index

Description and Priority

This use case permits a DILIGENT user to describe an index (define a type or schema of and index) for an archive or other set of documents. The use case in itself is mandatory, but it covers all potential features of index descriptions, many of which are optional.

Functional Requirements

An index description specifies how the index can be used from the search subsystem, i.e. what search and ranking operations the index supports on what data. It determines how input documents must be preprocessed in order to index them (and what categories/types of document can be indexed at all). There are also various technical parameters that must be specified, e.g. index organization, performance tradeoffs, optimizations.

DILIGENT must be able to exploit existing (legacy) search/index engines. There should be one uniform DILIGENT language/model for index descriptions, which can be mapped to native index descriptions for the underlying engines. Federated search depends on this capability.

DILIGENT users must be able to define new indices to be created over existing or new document sets, but also to describe existing indices residing in external archives to be imported into the DILIGENT infrastructure, such that these indices can be made available for search in a uniform fashion.

Index descriptions may consist of both relatively high-level specifications (e.g. what user-visible features the index should support) and more technical issues of how the index is organized and mapped to the physical level. There should be a way for administrators to prepare description templates or rule sets that fill out the technical details, enabling regular users to define indices without need for specific technical expertise. In the simplest cases, index descriptions may be implicitly selected or generated with no need for user involvement.

Index descriptions provide the meta-information needed by the search service in order to perform search on a given index, and in particular the meta-information needed by the data fusion service to merge data retrieved via different indices.

The index description is separate from the set of documents (archive) it is applied to. (Applying the index description to a set of document to yield an index instance and handling updates to those documents is described in Create Index and Incremental Indexing.) However, the index description contains specifications that directly or indirectly impact creation or update of indexes, in particular whether incremental update is possible or what freshness is achievable.

Example categories of functional and non-functional characteristics of indexes that should be possible to specify:

- Schema/format of meta-information to be indexed.

- Fields/schema of index itself.
- Forward or reverse indexes.
- Sorting, ordering, normalization of data, supported search operators.
- Performance tradeoffs.
- Incremental update capabilities, freshness requirements.
- Index organization (data structures), in so far as this is not implied by other specifications.

(The list is tentative, and specific index features to be supported are to be defined, taking into account the particular requirements to multimedia search and geographical search.)

The functionality enabled through a particular index description is invoked through the other use cases: Generate index and Incremental indexing for generating or updating the index, Index lookup for all of the lookup functionality.

The index description is a particular type of document (object), probably expressed in an XML-based notation. Index descriptions may be stored in the users' workspaces or in archives.

Numbers

The task may be performed for each index, but it is the intention that several indices may share the same description or reuse an existing description with minimal modifications.

Constraints and Assumptions

The specific properties and functionalities the indexes should support must be derived from the required search functionality, feature extraction and metadata, defined in other use cases.

UML Diagrams

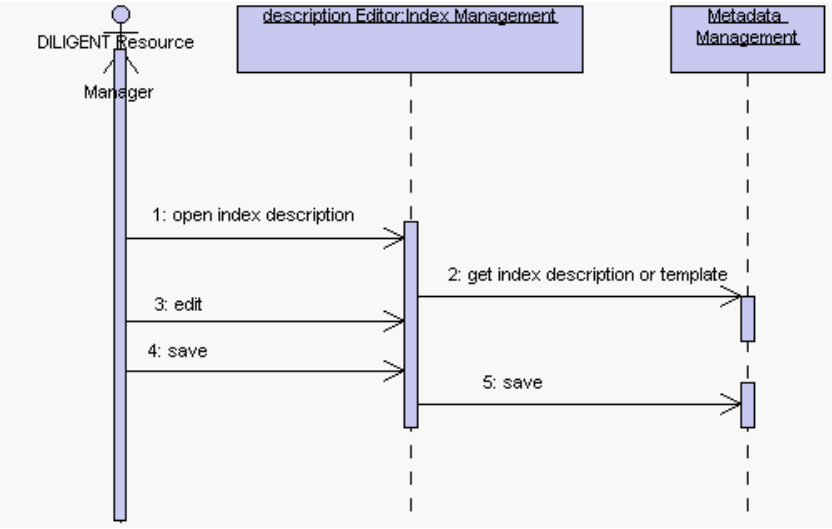

Figure 47: Describe index (sequence diagram)

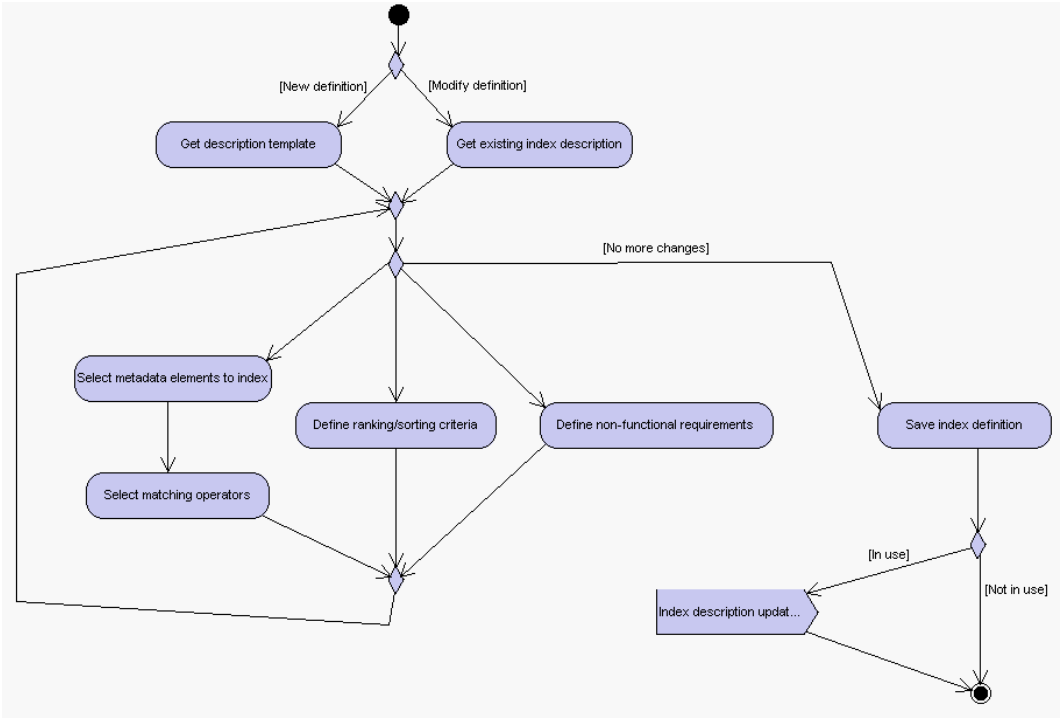

Figure 48: Describe index (activity diagram)

Grid Exploitation

None. This task itself is not computing or data intensive.

System Integration

Index descriptions can be stored in workspaces and DLs and in general handled like any other object.

The index description editor is a standalone tool, but there may be some integration with the metadata management, e.g. to extract metadata schemas and auto-generate corresponding index descriptions. The schema notations used in metadata management and index descriptions should coincide.

Related Non-Functional Requirements

Many non-functional requirements to the index, such as resource requirements, performance (space, time) tradeoffs, incremental updateability, are directly or indirectly influenced by management functionality defined here.

The index description format should be extensible, such that new index types, matching operations and other features and options can be handled in a plug-in fashion.

6.2.3 Generate Index

Description and Priority

The user applies an index description (see Describe Index) to a set of documents to create or update an index. The use case is mandatory, since it is the only way a new index can be created.

Requirements

The task is invoked at any time when an index is required, i.e. for any set of documents that can be indexed. (This includes at least archives to be imported into the DILIGENT infrastructure, see Constraints and Assumptions.)

For requirements to the index as such, see Describe Index and various use cases specifying search functionality.

The index description may have to be amended with archive-specific details, e.g. how the archive is to be accessed and updates are to be handled.

Specific features of an index, such as type and organization of index, supported metadata types, matching operators and other options may depend on plug-in components.

Numbers

See Index Management. This use case is invoked at least once per new index. If Update Index or Incremental Indexing is not supported, the use case will be invoked each time there is a need to update an index, see Update Index.

Constraints and Assumptions

We assume that indexing is to be applied to archives. (It may be useful or required to create indices over arbitrary collections or sets of documents defined in other ways, but we haven't identified any specific requirements in the scenario documents.)

UML Diagrams

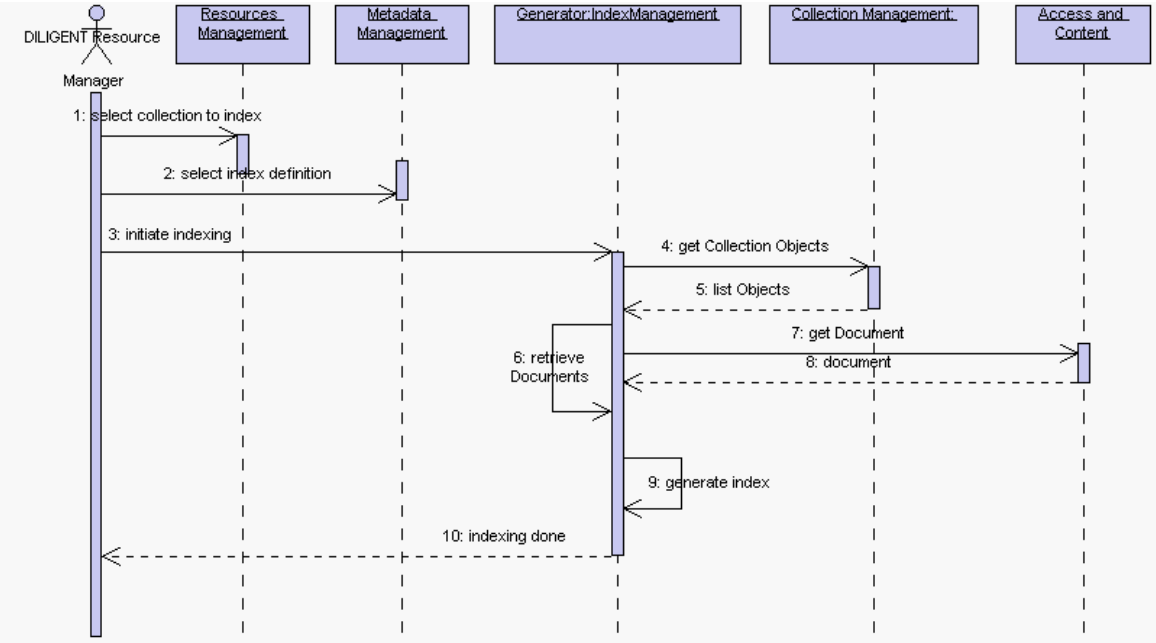

Figure 49: Generate index (sequence diagram)

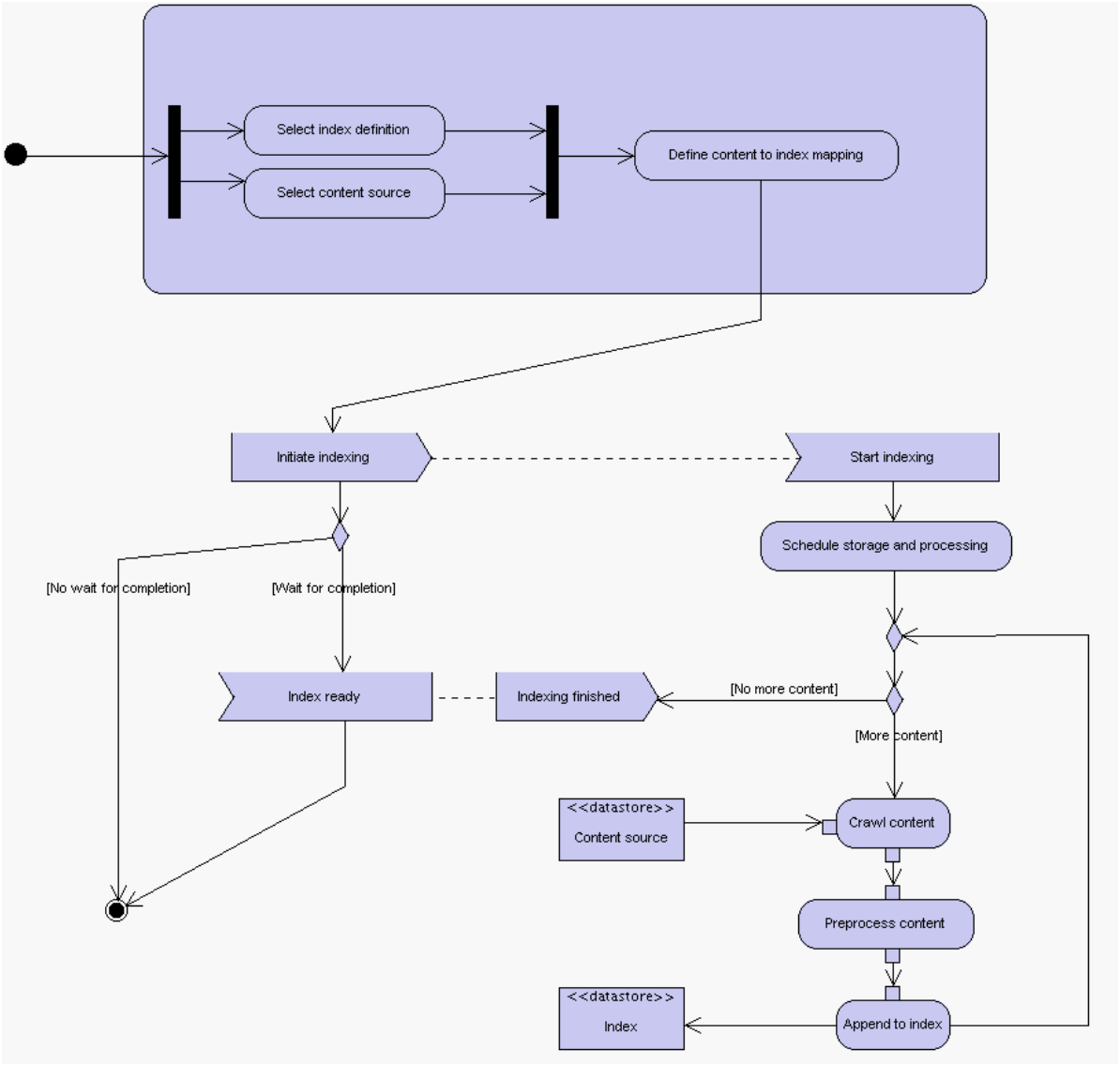

Figure 50: Generate index (activity diagram)

Grid Exploitation

Creation of an index is a data and processing intensive operation that in principle is well suited for grid exploitation. However, an index is typically voluminous, and creating an index typically requires reading all data in the collection to be indexed. While some of the processing (see Preprocessing Contents) is easily parallelizable, some types of indices are merged data structures with limited opportunities for concurrent generation. Existing search/index engines tend to be tightly coupled through the index data and don't expose the index in itself to the outside, only through the search interface. All those reasons reduce the opportunities for and benefits from grid deployment.

Short term:

- Wrap and publish existing search/index engines as grid services, but use native installation/deployment facilities to manage them. (I.e. no grid-based dynamic deployment of the engines or some of their components.)

Medium term:

- Grid-controlled deployment of existing search/index engines, but treating existing products as black boxes, not exposing internal interfaces to the grid. Objective: Deploy closely to/co-located with archives to be indexed.

System Integration

The task interacts with Content and Metadata Management to locate the content to index, determine what content is changed and schedule feature extraction and other preprocessing of content.

The storage service may be used to manage storage for the index itself.

Related Non-Functional Requirements

When updating an index, the work spent in re-indexing should be minimized. This in particular applies to the case where the index description is unchanged, but documents in the set have been modified or documents added to or removed from the set.

The index service should be extensible with respect to the types and features of indices to support, see Describe Index. The basic interface to the index service for generating indices accepts an index description, and should be able to handle extensions sensibly, either by dispatching the extension to the appropriate plug-in component or informing the user that the extension is recognized as such but is not supported.

6.2.4 Update Index

Description and Priority

After changes to the indexed set of object or the objects themselves, the index is updated to make it consistent with the new object set. This use case specializes Generate Index and is optional, since it is primarily an optimization.

Functional Requirements

Update Index is invoked periodically, when changes are known or suspected or when there is a need to ensure the index is consistent with the indexed set of objects.

Full regeneration of an index (as in Generate Index) may be too expensive or cause too long indexing latency, with index freshness suffering as a result. If the index organization permits incremental update, and it is possible to detect changes since the last time the index was regenerated or updated, the index update process may be optimized.

Numbers

The frequency with which this use case is invoked depends on the rate of update for the underlying archive and what freshness is required from the index. The amount of work for each invocation depends on how well re-indexing is optimized. In the optimal case, the work would be on the order of the size of change, but in practice one should expect a (fairly large) overhead per invocation, independently of size of change.

Constraints and Assumptions

Optimized re-indexing requires a means of identifying documents in the original and updated set and detecting changes (e.g. timestamps or version identifiers). This functionality ultimately depends on the archive system holding the set of documents to be indexed.

The index structure itself must support incremental updates. (Some index structures, in order to optimize for lookup speed, must be regenerated in batch.)

UML Diagrams

See Generate Index.

Grid Exploitation

See Generate Index.

System Integration

See Generate Index.

Related Non-Functional Requirements

The (old version of the) index should be available during index update. Time and resources spent re-indexing must be significantly better, compared to Generate Index.

6.2.5 Incremental Indexing

Description and Priority

Indexes designated as incremental are automatically kept consistent with the underlying set of documents as the set or the documents themselves change. This use case is optional.

Functional Requirements

The DILIGENT users should be able to define indices as incremental and have them automatically kept consistent with the underlying set of documents. An incremental index may receive updates in the form of document addition/deletion or modification (of fields in a) document and incorporate these changes independently of other documents.

Incremental indexing can be driven by the document archive itself sending change notifications. Alternately, a "crawler" must be configured to revisit each document in the archive periodically to identify changes. The crawler may depend on the state of the index (document IDs, timestamps or fingerprints) to determine changes in the archive.

Crawling and incremental indexing are relatively expensive operations, and DILIGENT users/administrators must be able to manage the activities to make appropriate tradeoffs between index freshness and resource consumption.

The stream of updates fed into an incremental index can also be fed in parallel to a component managing collections defined by a search (cf. FASTs Real Time Filter).

Numbers

See Update Index. Unlike Update Index, Incremental Indexing is continuously and automatically triggered as changes occur in the underlying storage system or are detected by the crawling process. However, the eagerness of the updating process may be configurable, such that some batching occurs.

Constraints and Assumptions

We assume that documents in a collection are relatively independent, such that there are no strict requirements for changes related to different documents to be applied in a particular order.

UML Diagrams

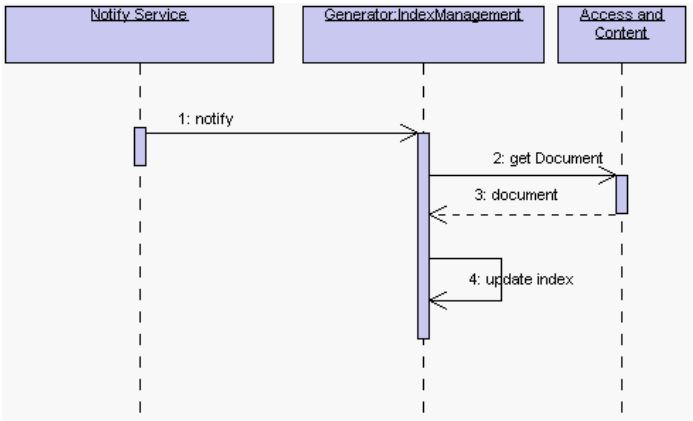

Figure 51: Incremental indexing (sequence diagram)

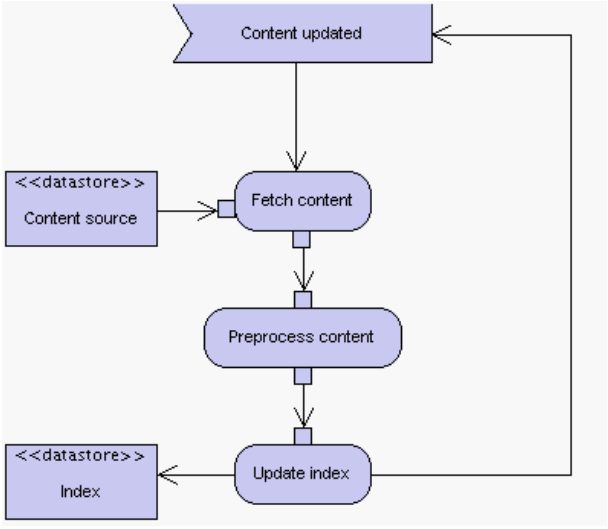

Figure 52: Incremental Indexing (activity diagram)

Grid Exploitation

With incremental indexing, there are additional benefits to deploying the agent (e.g. crawler) responsible for identifying and extracting changes at or close to the archive node. Otherwise, this case is similar to the Generate Index and Update Index cases.

System Integration

Incremental Indexing depends on triggering mechanisms in Content and Metadata Management.

The storage service may be used to manage storage for the index itself.

Related Non-Functional Requirements

The work spent re-indexing should be minimized. This obviously depends both on a suitable organization and implementation of the index itself and on the changes to be incorporated being precisely identified.

The availability and consistency of the index should not be adversely affected by re-indexing taking place.

The freshness of the index (i.e. the latency from a change occurs in the underlying documents until the change is visible through the search interface) is to be minimized, but depends on how much resources can be allocated to crawling and re-indexing.

6.2.6 Register Object Change

Description and Priority

This use case is invoked as an extension of the Notify Service use case in Change Tracking, whenever an object is changed that is indexed in a non-incremental index. Its purpose is to mediate between change tracking and the actual index update process. The index service registers change notifications and schedules batch updates of the index. The use case in itself is mandatory, although some of the variants of its functionality may be optional.

Functional Requirements

The basic required functionality is merely to mark an index as inconsistent, such that it will eventually be regenerated. The decision to regenerate an index could be manual, or triggered by some automatic strategy, for instance based on periodic checking, number of pending changes or the time index has been inconsistent.

For the indexer, the fallback case is to regenerate the entire index from scratch, i.e. by reading all objects currently in the indexed collection. Some types of indexes can be

regenerated more efficiently if more specific information about the changes is available; thus specific index types should be able to install their own notification handlers. In particular, updates affecting only some subsets of index fields may be particularly efficient.

Numbers

This use case is invoked each time an object indexed in a non-incremental index is changed, i.e. very frequently.

Grid Exploitation

None in particular. However, change tracking may touch on fundamental grid mechanisms.

System Integration

This use case is the integration point between the index service and the Change Tracking functionality in Content Management.

See also Handle Object Change below.

Related Non-Functional Requirements

See Handle Object Change below.

6.2.7 Handle Object Change

Description and Priority

This use case handles the processing performed when there is a change to an object that appears in an index, including adding or removing objects in an index. Metadata to be indexed is (re)generated and any other preprocessing that is required for the index fields is performed. Change tracking in the Content Management service is set up in order to get notifications of future changes to the object.

For incremental indexes, this use case is triggered by each change. For non-incremental indexes, the triggering is mediated by the Register Object Change use case and can be delayed, postponed or batched according to policies specific to a given index.

This use case is mandatory, but there are a lot of options (e.g. content- or index-type specific functionality) that are not explicitly modeled as extensions.

Numbers

This use case is invoked each time an object that appears in some index is changed and the index is regenerated or updated, i.e. very frequently.

Constraints and Assumptions

There are no specific assumptions as to the nature and precision of change information that available, but clearly this will influence the interfaces to Content Management and the ability to optimize index regeneration. In the simplest case, only object identifiers are available, along with an indication of whether the change is addition, update or deletion. Ideally, update notifications should indicate which fields/attributes are affected. It may not be possible to (efficiently) perform a differencing operation to determine a minimal set of changes to the object.

System Integration

It is not at this point clear whether it is the index service that is responsible for invoking the metadata service in order to re-derive metadata needed for index fields, or whether this is supposed to be handled more generically in a collaboration of the metadata and content management services without involving the index service. (In other words, the question is whether the index service is triggered directly by changes in the base objects or indirectly by "downstream" changes in derived metadata.)

In the latter case, a generic version of this use case would more appropriately belong to the metadata service. The generic use case is extended according to the nature of the specific metadata in question, i.e. with indexing functionality for the particular cases of indexed metadata or the index itself being considered as metadata.

6.2.8 Index Lookup

Description and Priority

The search subsystem invokes the index service to look up information in indexes, as determined by the query optimizer. This use case is mandatory.

Functional Requirements

The lookup functionality (lookup interface) available for a given index is defined by the index description. Performance and other non-functional requirements may also be determined or influenced by the index description, e.g. through its selection of index organization.

The basic functionality of Index Lookup is to perform a match operation on one or more fields in the selected index and return (references to) the matching objects. The match operators and options available depend on the type of index and the field being matched, and the basic interface to Index Lookup should be neutral with respect to that.

The search service must be able to query the index description governing a particular index.

The result set may be very large, and it should be an option to retrieve only a part of the result set, by specifying a limit to the number of results or by reading results sequentially until satisfied. The results are returned in rank order. The index may support different ranking criteria, and the search service must be able to select among them as well as supply parameters (e.g. weights) for the rank computation.

In order to support data fusion, i.e. merging data from different indexes with proper ranking, it should be possible to return relevant statistics about the result set (e.g. term occurrence statistics) and per-document ranking information (at least the computed result rank value, but input/intermediate values in the rank computation may be useful) to enable data fusion to compute a normalized rank across different indexes.

The result set may contain additional information about the hit occurrences, e.g. positions, context, match precision, depending on what is supported by the type of index and nature of the data. This information is intended for further processing on the side of the search service, such as matching or ranking operations not directly supported by the index, to support data fusion or to generate "teasers" (summaries of search results for presenting to human users).

Numbers

Search is assumed to be a very frequent operation.

UML Diagrams

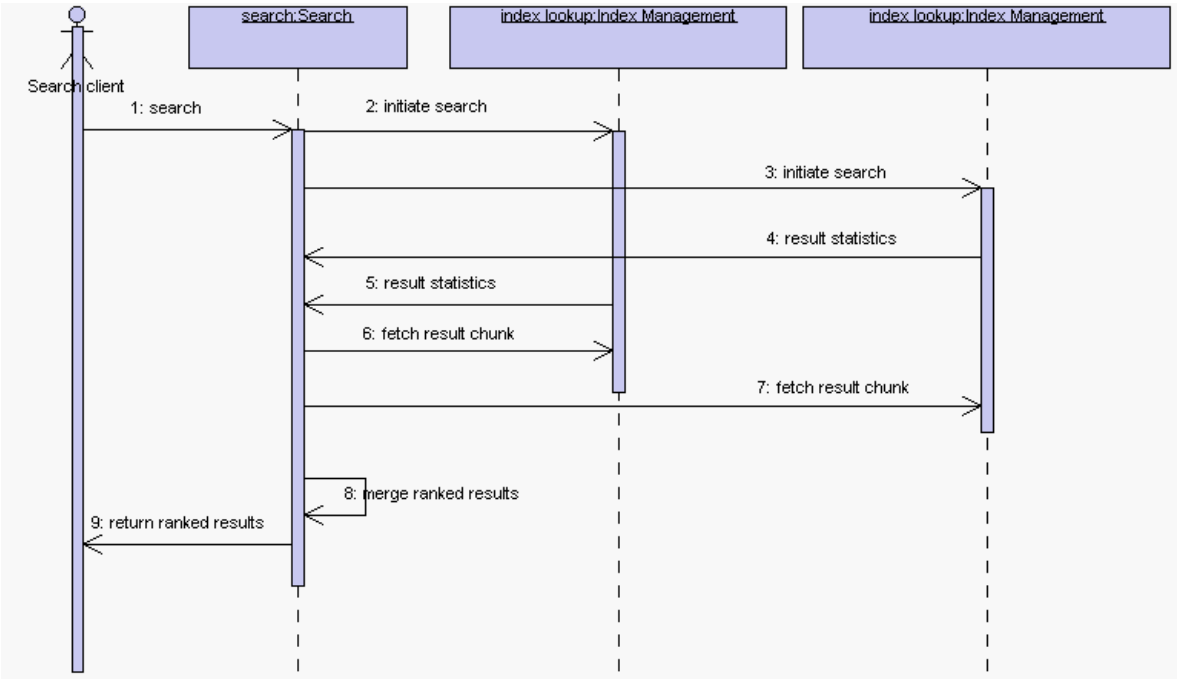

Figure 53 Index lookup (sequence diagram)

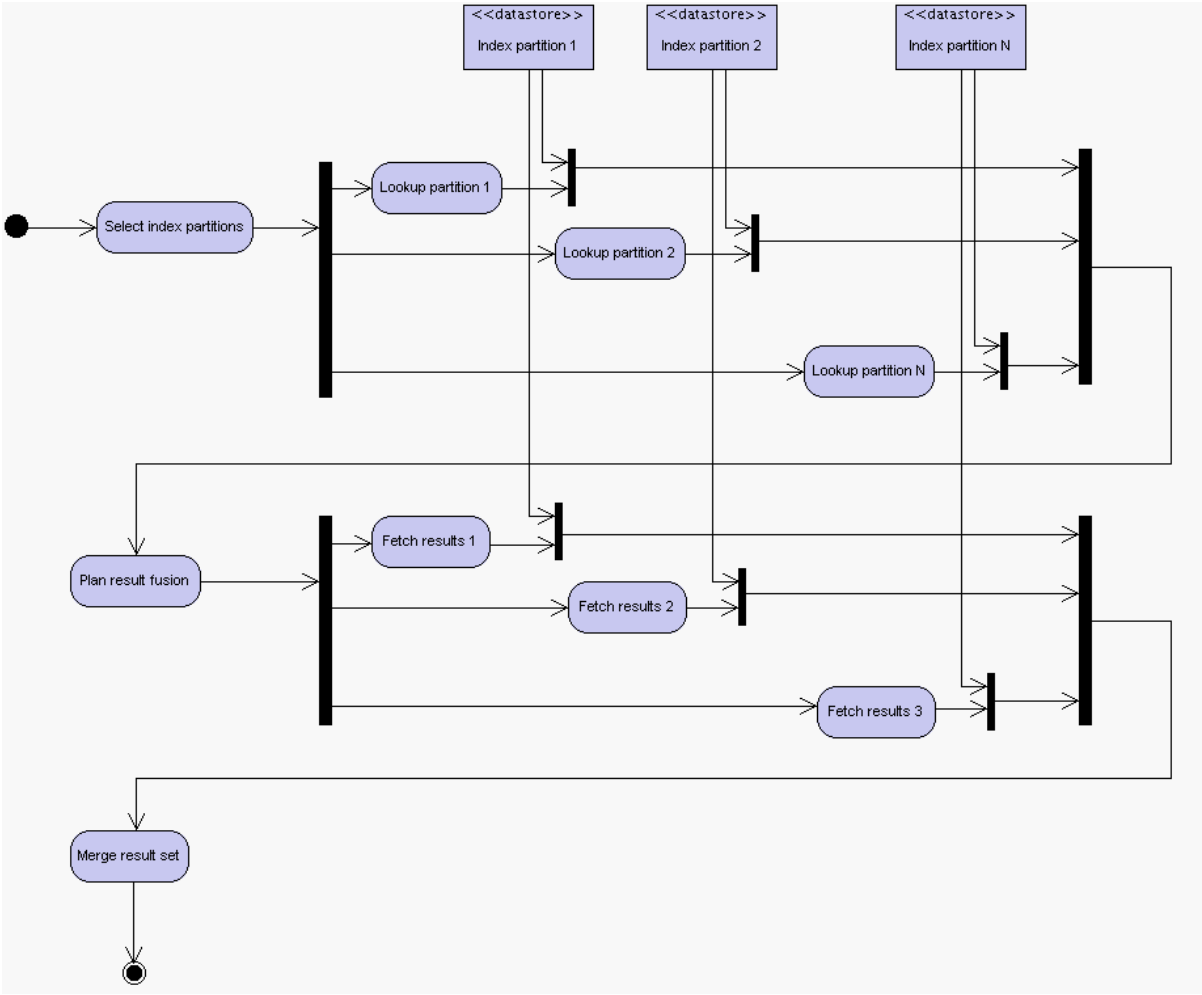

Figure 54 Index lookup (activity diagram)

Grid Exploitation

The grid primarily acts to locate the index service containing the requested index. The grid may also be a vehicle for replicating the index. The index service is a "grid service", in the sense that it can be located and invoked through standard grid mechanisms. However, on the short and medium term, we expect most implementations to be based on legacy, non-grid software, wrapped with grid-compliant interfaces, but statically installed on available nodes. (Filtering and ranking components might be dynamically deployed on the node where the actual index is located in order to increase precision, reducing the size of the result set; it is not clear whether such components should be considered as part of the index, search or perhaps data fusion service.)

Related Non-Functional Requirements

See Index Management.

6.2.9 Wrap External Index

Description and Priority

Indices outside DILIGENT are made available inside DILIGENT via a wrapper. This use case in itself is mandatory, but the characteristics (interfaces etc.) and actual set of external indices to support with wrappers is to be defined.

Functional Requirements

In order to be usable within DILIGENT, an index must provide the expected DILIGENT interfaces. For indices that actually are implemented outside DILIGENT, the index is represented inside DILIGENT by a wrapper implementation that implements the required interfaces and adapts and forwards service request between internal DILIGENT components and external components.

The index features and functionality provided by the external index is described through an index description. The wrapper should behave like a native DILIGENT index described by the same index description. There may be some restrictions on what index description features may be used in/supported by wrappers in general, but such restrictions should be minimized. (On the other hand, one particular wrapper implementation will typically cover a very restricted set of cases.)

There should be a library of wrappers available covering the common cases, such that new indices can be integrated without programming, merely by selecting and configuring a wrapper from the library. However, the set of external indices is open-ended, and it must be possible to create new, custom wrappers, see Develop Wrapper.

Note that there are two basic ways in which an external content source can be integrated with DILIGENT:

- An external index over the content source is made available inside DILIGENT. (The case in question here.)
- The content source is crawled and an index is built inside DILIGENT. "Generate Index" and other use cases within this package cover this case.

In both cases, the content source itself must be wrapped and appear as a collection/archive inside DILIGENT. This is outside the scope of the index service and is not discussed here (although the wrapper approach is similar). From the point of view of other DILIGENT services, the two cases should be indistinguishable (except for the index features exposed, as described by the index descriptions).

Numbers

Comparable to "Generate Index".

Grid Exploitation

Not relevant. (Some specific wrappers might have grid-related issues, however.)

System Integration

This is one of the major ways in which interfaces between DILIGENT and external services are defined and implemented.

6.2.10 Develop Wrapper

Description and Priority

By using a DILIGENT software development kit (SDK), custom wrappers can be developed. In this way new content sources and indices can be supported, or new/custom mappings between DILIGENT and external functionality.

This use case is mandatory. (It is the way we are going to populate the wrapper library initially.)

Functional Requirements

Wrappers should not unduly restrain the functionality and index features exposed: As long as there is a useful mapping between the functionality and features supported by the external source and those that DILIGENT can exploit, it should be possible to implement that in a wrapper.

Numbers

Relatively infrequent, at least when a suitable library of wrappers has been built up over time.

Constraints and Assumptions

The SDK should support common programming languages and programming models, since there may be a requirement or at least strongly desirable to use existing components, libraries, toolkits to implement the access the external content source.

Grid Exploitation

Not relevant.

Related Non-Functional Requirements

Most regular software engineering issues apply.

Wrapper APIs should be well-defined and stable, such that existing wrappers can be used with new releases of the DILIGENT platform with no or minimal change.

6.3 Content Source Description and Selection (CSDS) and Data Fusion

In their most general form, CSDS and Data Fusion are functionalities associated with the provision of services, which rely in real-time on a distribution of data and control[11]. Specifically:

- Data Fusion is real-time functionality concerned with the synthesis of service responses from the partial outputs produced at individual loci of distribution, or *nodes*;

[11] Here and in the following, the term 'real-time' and its opposite 'batch' are informally intended with respect to service end-users (human or otherwise).

- CSDS is functionality concerned with the optimisation of the number of nodes – and thus network interactions – which are required to satisfy individual service requests. In particular:
    - Content Source Description is off-line functionality which produces information about the data available at each node, and accordingly is a specialised form of indexing functionality[12];
    - Content Source Selection is on-line functionality that uses the output of Content Source Description in order to distinguish nodes on the basis of their likelihood of `relevance'[13] to the satisfaction of individual service requests.

From this general perspective, CSDS and Data Fusion functionalities are largely orthogonal to the semantics and distributed architecture of the services they support. In the literature, however, they are most commonly associated with *content-based*, *application-level search services* – primarily full-text but also multimedia document search – and under the assumption of client-server and, more recently, peer-to-peer architectures (Distributed Information Retrieval, DIR). As shown below, it is largely in this sense and with such scope that we shall interpret them within the Digital Library Layer (DDL) of the DILIGENT platform.

Note: Functionality related to CSDS and Data Fusion may also be identified within the Collective Layer (DCL) – and, further away from applications, within the underlying GRID middleware – in the form of brokering and matchmaking functionality. Here, however, we shall assume a more restricted scope that is closer to applications and their high-level semantics (e.g. information content), rather than to computational resources and their effective sharing within a distributed infrastructure[14]. In any case, we make no assumption at this stage as to the specific distributed architecture of the supported services (client-server or peer-to-peer), which may vary from service to service across the DILIGENT platform.

As DLL functionalities, CSDS and Data Fusion do not explicitly emerge in the user requirements of D2.1.1 and D2.1.2. Rather, they may be identified as supporting both content-based and structure-based Search functionality, which interfaces directly with end-users in a number of uses cases, namely:

- Search and Retrieve Objects (ARTE)/ Explore Objects (ImpECt)
- Search Objects by Full-Text (ARTE)/ Search for Objects (ImpECt)
- Search Objects by Video (ARTE)
- Search Video Frames by Keywords (ARTE)
- Search Objects by Tone (ARTE)
- Search Objects by Metadata (ARTE)
- Search Objects by Image (ARTE)
- Search Part-of Objects by Image (ARTE)
- Search Archives (ARTE)
- Search Archives by Image (ARTE)

[12] Indeed, there may be some dependencies/overlap between Index Definition and Content Source Description still to identify and resolve.

[13] Notice that we assume here the most generic interpretation of the term 'relevance' and do not immediately relate it to the information need underlying a user query (rather, we relate it only to request satisfaction). Of course, a narrower interpretation is harmless as we will soon restrict our scope to search services.

[14] Of course, we do not exclude at this stage that implementation of such functionalities may directly rely on related DCL functionality.

- Search Archives by Metadata (ARTE)

Of course, detailed analysis of these use cases concerns the functional specification of Search functionality and thus escapes the immediate scope of these specifications. Here, however, we observe that such functionality is given *high priority* across both user communities; in the ARTE scenario of D2.1.1, for example, search functionality supports most management activities, from DL Management and Workspace Management to Workshop Management and Collection Management. Specifically, full-text search, search by image, and structured search are perceived as *mandatory* or *highly desirable* functionality, while more sophisticated forms of search, such as search by tone and search by video, are described as *desirable*. In particular, we notice that Search functionality spans different types of media – from plain text to images, videos, and audio – as well as approaches – from structured to semistructured and unstructured content. We thus inherit *a generic requirement for media- and structure-dependent forms of source description and selection, as well as data fusion*.

Note: We notice that all forms of search are associated with auxiliary functionality which allows users to indicate and thus limit the content sources against which searching should be distributed. While this may be used towards performance optimisation – and thus as a manual and user-driven complement to Content Source Selection functionality – it is more directly intended, and typically used, to increase precision of query results on the basis of static knowledge of the sources' content. As such, we relate this functionality to the user interface, locate it within ASL layer of the platform, and leave it outside the scope of these functional specifications; its only effect is to subset the input of Content Source Selection functionality.

Besides requirements, we also inherit an operative context from Search functionality and base on it a number of assumptions. First, we assume the definition of a *canonical query language* for DILIGENT – presumably one capable of mixing *structured and unstructured searches* and with *operator support for multimedia input* – and expect all queries to be cast in the form of expressions of such language. Similarly, we assume agreement on a *canonical format for query results* and expect support for Data Fusion functionality to come in the form of proposed extensions for such format.

Of course, we assume that the execution of user queries *may* be distributed across a finite number of nodes – presumably via collaborating instances of the Search Service – and thus that mechanisms (e.g. protocols) exist within the platform for dispatching queries to nodes and gathering partial results prior to their fusion. We also assume that suitable query engines are available at each node that participate of query execution, where each such engine may have been deployed as part of the DILIGENT platform or as the result of some ad-hoc wrapping of pre-existing search services. In the latter case, we assume – but not concern ourselves with – the existence of mechanisms for (partial) query translation (e.g. as carried out by 'proxy' nodes in the MIND project). In fact, we consider local search engines as black boxes and thus abstract over the details of their indexing and retrieval strategies.

Finally, we capture the interactions between CSDS, Data Fusion, and Search functionality in following use-case diagram.

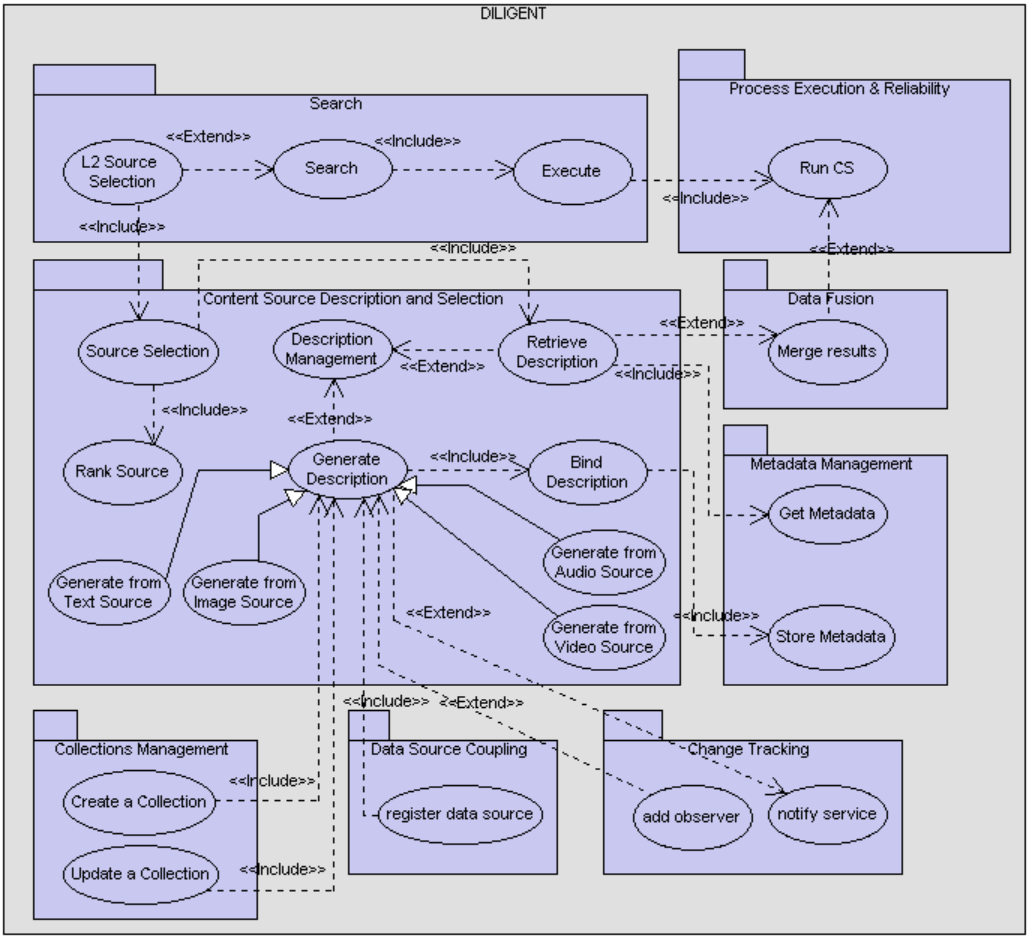

Figure 55: CSDS and Data Fusion (use case diagram)

6.3.1 Content Source Description

Description and Priority

Set of high-priority DLL functionalities for the generation and management of content descriptions for searchable data sources. The assigned priority reflects the key role of Content Source Description in supporting the scalability of Search functionality, which has been in turn identified as high-priority functionality across both user communities.

Functional Requirements

At the very least, the functionalities must allow the *generation*, *binding*, and *retrieval* of content descriptions for searchable data sources. The binding of a description refers to its explicit and persistent association with the content source from which it has been generated. The generation of a description must trigger notification mechanisms available within the platform (presumably as Content Management and Metadata Management functionality), and these mechanisms must then be used to maintain bindings up-to-date within an acceptable time lag – via re-generations or, preferably, incremental updates of currently bound descriptions.

Descriptions must be generated from content sources of different media and format types. Text must be supported while images, video, and audio – as represented through a small range of popular formats yet to be identified – should be supported through media-sensitive descriptions mechanisms. Specific kinds of structured and semistructured content sources, such as relational or XML databases, must also be supported through structure-sensitive description mechanisms (e.g. fielded descriptions). As description models vary across and within media, we require the platform to offer basic descriptions for each of the supported

media and then accommodate application-specific descriptions within the platform through inheritance mechanisms.

For the time being, we require that each content source be bound to a single description; retrieval of descriptions is then predicated only on the successful identification of the corresponding sources. As a longer-term goal, however, we plan to remove this constraint and allow sources to be described in different ways for different purposes. Of course, this requires descriptions to be explicitly identified (either opaquely or by a controlled vocabulary of types) and the identifier used as a retrieval parameter. It also raises a requirement for functionality, which caters for the retrieval of *all* the descriptions available within the platform at a given moment in time for a given source.

Numbers

Load on Content Source Description functionality is expected to be relatively contained and concentrated primarily at DL creation time. As the functionality is shared across DL applications, testing is needed to reveal and quantify possible peaks of usage. On the other hand, descriptions for given sources may outlive the DL for which they were initially created and thus may be reused over time (possibly subject to updates, of course).

Constraints and Assumptions

We assume that the generation and binding of descriptions occurs as soon as sources are included within a DILIGENT DL and thus that the functionalities operate in batch mode with respect to the execution of queries which they ultimately support (see also System Integration section).

Clearly, descriptions may be retrieved only after they have been bound to the sources from which they were generated. Bindings themselves must be tested for conformance to the description format, and an error should be generated when the test fails.

At this stage, we make no assumption as to the degree of persistence or distribution model of descriptions within the platform.

Regardless of the media type, we assume all source descriptions to be structured and mostly textual. In particular, we assume them to be represented in some canonical *description format* for the DILIGENT platform. We expect such format to provide a number of fields – whether compulsory or not, media-dependent or media-independent – which prove necessary to support content source selection algorithms (e.g. local ranking algorithm ID, stop word list flag, date of last change, support for fielded search flag, etc). Finally, the description format should be extensible, so as to accommodate application-specific extensions, which we expect to mirror/complement the functional extensions discussed above[15].

Note that description of text-based sources is well known and offers good properties. Vocabularies annotated with frequency of occurrence data, or 'unigram language models', offer favourable growth rates and their automated synthesis using query-based sampling has been proved effective. Description formats for text have also been proposed (e.g. see the STARTS protocol at Stanford). Description of non-textual unstructured sources is instead a much less explored issue.

UML Diagrams

See Figure 55: CSDS and Data Fusion. Further functionality decomposition may occur at later stages (design) but should be considered as 'internal' or 'micro-functionality' from the system perspective (no direct interaction with other platform components).

[15] At this stage, however, we are not directly concerned with approaches to the interoperability problem, which we expect instead to be dealt with uniformly within the platform at large.

GRID Exploitation

As currently understood, Content Source Description functionality may raise intensive processing or storage requirements, especially in the case of generation of non-textual unstructured source (e.g. images, video, audio). For its batch nature with respect to the query execution processes it ultimately supports, the functionality is an ideal candidate for GRID exploitation.

System Integration

We expect the functionality to be invoked exclusively by Search functionality, in particular after the L1 Source Selection functionality.

Related Non-Functional Requirements

From a qualitative perspective, descriptions must effectively support Content Source Selection functionality and, in turn, the different forms of Search functionality which are made available on the platform. Specifically, *effective description* is to be measured against requirements of convenient storage, flexible evolution, and mostly effective source selection (see Content Source Selection for a definition of 'effective selection').

6.3.2 Content Source Selection

Description and Priority

High-priority DLL functionality for the selection of searchable content sources based on the likelihood that they will prove effective in satisfying the information need underlying a given user query. The assigned priority reflects the key role of Content Source Selection in supporting the scalability of Search functionality, which has been in turn identified as high-priority functionality across both user communities.

Functional Requirements

Given a query *q* and a set *SD* of content source descriptions, the functionality must identify a subset *S'* of the sources described in *SD* across which the execution of *q* would be effectively distributed. As a first requirement, the selection must be effective in promoting the *scalability* of the different forms of Search functionality that are available on the DILIGENT platform[16].

We suggest a measure of e*ffective selection* in terms of the following:

- the ratio *k/n* between the number *k* of sources in *S'* and the number *n* of sources described in *SD*. For given source descriptions and values of (ii), smaller ratios *k/n* indicate better optimisation of network use and smaller information overload for users;
- the ratio *R(k)/R(n)* between the *effectiveness of retrieval* when the query is distributed across the sources in *S'* and the effectiveness of retrieval when the query is distributed across all the sources described in *SD*. For given source descriptions and values of *k*, smaller ratios *R(k)/R(n)* indicate that given optimisations can be achieved at less significant losses of retrieval effectiveness;

and we notice that any such measure is always:

- relative the query *q* and the quality and freshness of the descriptions in *SD* and,
- (orthogonally to queries and source descriptions) ultimately dependent on the type, semantics, and distribution of the content of the sources described in *SD*.

[16] Besides scalability, other decision-theoretic parameters related to the effectiveness of service provision may be identified at a later stage.

For these reasons, we identify a requirement for supporting multiple selection algorithms within the DILIGENT platform and for allowing those algorithms to be parametrically configured, possibly on a per-query basis. As with Content Source Description functionality, and indeed in parallel with it, we require the platform to offer basic algorithms for each of the supported media and also to be able to accommodate application-specific extensions across and within media types through inheritance mechanisms.

Numbers

The load on Content Source Selection functionality is exactly that estimated for Search functionality, upon which this functionality depends.

Constraints and Assumptions

We assume that the selection of content sources occurs prior to the dispatching of queries to nodes that participate of their execution, and thus that Content Source Selection functionality operates in real-time mode with respect to such execution.

The query *q* must be expressed in the canonical query language and the descriptions in *SD* include all the information required by this functionality. Failing these conditions, the functionality must produce an error. There is also an implicit assumption that the search service that exposes the content sources described in *SD* is capable of executing *q*, even though this is not directly tested or enforced by this functionality.

We assume that the sources in *S'* are selected against a *ranking* of the sources described in *SD*, where such ranking is based on the likelihood that such sources will prove convenient to the satisfaction of the information need underlying *q*. After such ranking has been derived, the sources in *S'* may be the first *k* sources, or all the sources ranked below a given threshold value, or else all those which satisfy some other cost metric to be defined. The value of *k*, the threshold value, or the cost metric adopted for the selection should be configurable, possibly on a per-query basis.

We make no early assumptions as to the exact definition of 'effectiveness of retrieval'. It may be defined exclusively in terms of relevance of query results or complemented by other decision theoretic, application-level parameters (e.g. financial cost). Whatever the specifics of the decision theoretic framework adopted, the exact notion of relevance employed therein should be defined in the context of Search functionality and, mostly importantly, taking into account the local search engines which participate of query execution. As such, these specifics are outside the scope of Content Source Selection functionality.

As mentioned above, source rankings may be obtained in a number of ways and different are also the metrics which may be used to measure their effectiveness at a given cut-off point. For full-text retrieval, as an example, dedicated applications of the Baysian Inference Network model (based on suitable variations of classic *tf.idf* formulas and known as the CORI algorithm) have been proved – using and a metric largely based on document recall values – to achieve acceptable results when selecting as little as 10% of a thousand sources. Given the expected size of DILIGENT DLs, we believe that such results give sufficient confidence – as far as full-text search is concerned – of the possibility of granting effective source selections.

UML Diagrams

See Figure 55: CSDS and Data Fusion. Further functionality decomposition may occur at later stages (design) but should be considered as 'internal' or 'micro-functionality' from the system perspective (no direct interaction with other platform components).

GRID Exploitation

The functionality may require complex algorithms of multi-criteria decision making which have not been explored for heterogeneous and mixed-media sources. On the one hand, this

complexity suggests good potential for GRID exploitation. On the other hand, it introduces further real-time performance overheads and this may well discourage its wide distribution for performance reason.

System Integration

We expect the functionality to be invoked exclusively by the Search service.

Related Non-Functional Requirements

For their particular relevance in this context, non-functional requirements are reported in the Requirements Section.

6.3.3 Data Fusion

Description and Priority

High-priority DLL functionality for the integration of partial results produced during the distributed execution of user queries across a number of searchable content sources. The assigned priority reflects the key role of Data Fusion in supporting the distribution of Search functionality, which has been in turn identified as high-priority functionality across both user communities.

Functional Requirements

Given a query *q*, a list *SD* of content source descriptions which preserves the ordering of the corresponding sources derived by Content Source Selection functionality, and a set *PR* of partial result rankings, the functionality must produce a single *merged ranking R* of the rankings in *PR*, where each ranking in *PR* is a finite list of results obtained by executing *q* at exactly one of the content sources described in *SD* and ordered by relevance to *q* accordingly to the notion of relevance defined at that source.

We require effectiveness of merged rankings, where an *effective ranking* is one that deviates as little as possible from the raking which would have been produced by a single query engine running *q* against the union of the content sources described in *SD* and under one indexing algorithm, one retrieval algorithm, and one notion of relevance[17]. For this purpose, we require fusion algorithms based on some notion of *normalisation* of the scores of partial documents as independently computed at each of the content sources described in *SD*. In particular, we require that score normalisation is such to: (i) favour results from sources which ranked high during source selection processes, but also (ii) enable high-scoring results from sources which ranked low during those processes.

Once again, we notice that effectiveness of merged rankings is dependent on the information made available to the merging algorithm and one such algorithm depends in turn on the type, semantics, and distribution of content across sources. For these reasons, we identify a requirement for supporting multiple data fusion algorithms within the DILIGENT platform and for allowing those algorithms to be parametrically configured, possibly on a per-query basis. As with Content Source Description and Content Source Selection functionalities, and indeed in sync with them, we require the platform to offer basic fusion algorithms for each of the supported media and also to be able to accommodate application-specific extensions across and within media types through inheritance mechanisms.

Numbers

The load on Data Fusion functionality is exactly that estimated for Search functionality, upon which this functionality depends.

[17] Obviously, even if all sources used the same retrieval engine there would still be a need for result fusion, but this would be trivial.

Constraints and Assumptions

We assume that fusion of partial results is a necessary condition to the satisfaction of a user query, and thus that Data Fusion functionality operates in real-time mode with respect to such execution. We predicate this assumption on the belief that, because of distribution and content selection processes, *all* queries should produce ranked results, even those which admits only exact answers at each participating content source. We note that, in this latter case, a basic fusion algorithm might simply merge results in the same order produced by Content Source Selection functionality. It is to support this and other non-trivial merging algorithms that we require the descriptions in *SD* to preserve such ordering. Of course, the ordering may also be ignored (e.g. when exact results must be sorted with respect to specific field values).

The query *q* must be expressed in the canonical query language and the descriptions in *SD* must include all the information required by this functionality. In particular, we expect that – for purposes of effective normalisation – this functionality will rely on *global statistics* for each of the sources described in *SD*, such as the number of documents in each source, the number of distinct indexing features within each source (e.g. terms), the number of documents with a given feature, the part of the query executed at a given query, etc. For the same reasons, we require that the rankings in *PR* contain lists of results represented according to an extension of the canonical query result format which provides all the input required by the merging algorithm. This may include – and must include in all non-trivial merging scenarios – the score locally computed for the result at each content source, as well as the frequency with which a given indexing feature occurs within the result. Failing any of these conditions, the functionality must produce an error.

UML Diagrams

See Figure 55: CSDS and Data Fusion. Further functionality decomposition may occur at later stages (design) but should be considered as 'internal' or 'micro-functionality' from the system perspective (no direct interaction with other platform components).

GRID Exploitation

See Content Source Selection functionality.

System Integration

Conceptually, Merge Results is invoked in the context of Search functionality. In practice, however, searches will be executed as processes managed by Process Execution & Reliability functionality, and thus the invocation of Merge Results will be part of such process execution.

Related Non-Functional Requirements

For their particular relevance in this context, non-functional requirements are reported in the Requirements Section.

6.4 Feature Extraction

Feature Extraction is the process in which an initial measurement patterns or some subsequences of measurement patterns are transformed to a new pattern feature for simplifying analysis processes. This functional area will provide functionality for:

- Image analysis: generation of metadata pertaining to the image content;
- Video analysis: generation of metadata pertaining to the video content;
- Audio analysis: generation of metadata pertaining to the audio content;
- Text analysis: generation of metadata pertaining to the text content;
- Cross media analysis: generation of metadata pertaining to different related media;

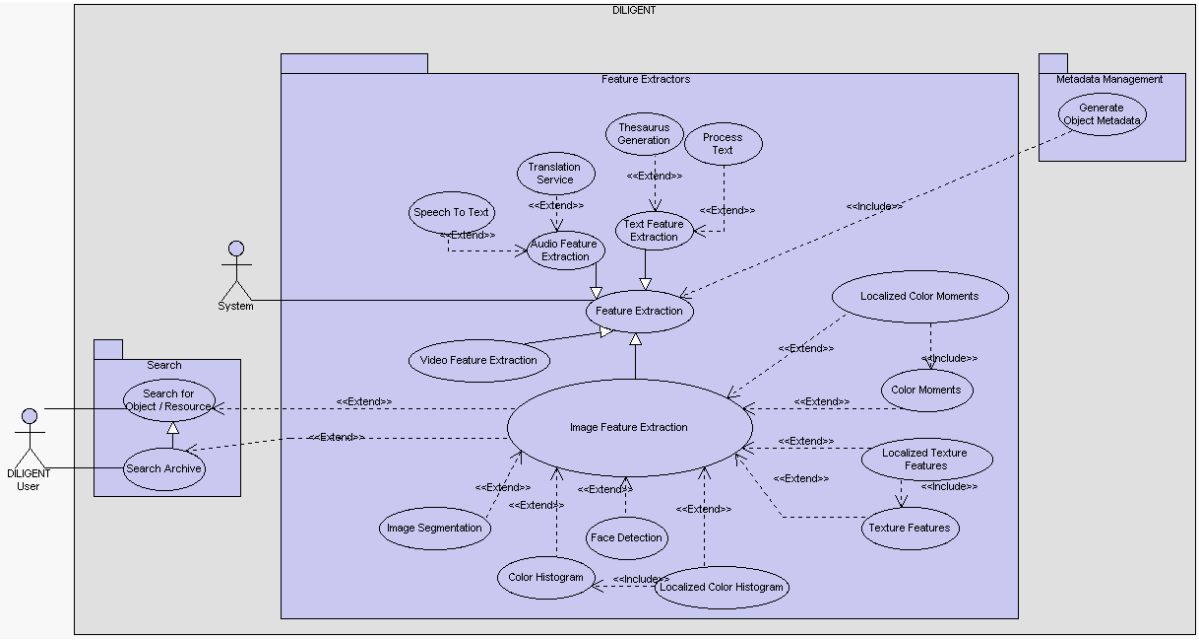

Figure 56: Feature Extraction (use case diagram)

6.4.1 Speech to Text

Description and Priority

This functionality will transform audio streams or audio tracks of video sequences into a written text in a given language.

Requirements

The service shall extract speech from vocal audio streams.

Constraints and Assumptions

- A speech recognition software must be freely available and offer a functional interface to couple it as a third party component.
- The given audio stream must contain speech in a supported language.
- All search functionalities taking advantage of this functionality must be aware, that speech recognition rests upon algorithms that do not accurately extract text in case of noisy audio, dialects or badly trained software.

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

Since speech recognition resides within a 3rd party component that processes complete audio files, "Speech Recognition" offers little potential for grid exploitation.

System Integration

"Speech Recognition" is a kind of "Audio Feature Extraction" (see there).

Use Stories

See ARTE requirements document.

6.4.2 Translation Service

Description and Priority

This functionality supplements the "Speech to Text" UC by an additional translation mechanism, which converts the extracted text into a text in another language.

Functional Requirements

"Translation Service" acts as a combination of (1) speech recognition in a given audio track, which extracts text in the native spoken language, and (2) translation of the given text into another language (probably the language of the user).

Constraints and Assumptions

Depending on the audio quality and the "training" of the speech recognizing software, speech recognition works fairly well. In difference, translation software still lacks the capabilities to correctly translate complex sentences. In terms of implementing this service, a translation software must be freely available and offer a functional interface to invoke it from within a DILIGENT component.

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

See "Speech to Text" service.

System Integration

"Translation Service" is a kind of "Audio Feature Extraction" (see there).

Use Stories

See ARTE requirements document.

6.4.3 Process Text

Description and Priority

"Process Text" is a plain text processing operation. The ARTE requirements document states that "Process Text" primarily does word stemming which is a precondition for or part of more complex text processing operations, like "Thesaurus Generation" (see there).

Requirements

"Process Text" will return a list of word roots of all words existing in a given textual document.

Constraints and Assumptions

A 3rd party word stemming component will be required in order to conduct the actual extraction of word roots.

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

"Process Text" comprises two phases, which (1) separate a given text into a list of separate words and (2) extract their word roots. Stage (1) can not take advantage of the grid, but stage (2) can be easily parallelized. Precisely, word stemming of a particular word is independent of the remaining words. That is, each participating grid node receives a list of words and extracts their word roots.

System Integration

"Process Text" is a kind of "Text Feature Extraction" (see there).

Use Stories

See ARTE requirements document.

6.4.4 Thesaurus Generation

Description and Priority

"Thesaurus Generation" generates a data structure for a given text, which establishes manifold associations among the contained words, including word roots, synonyms, similar words, etc.

Functional Requirements

"Thesaurus Generation" processes a multilingual text into a data structure which associates meta information with the words.

Constraints and Assumptions

A 3rd thesaurus generator component must be freely available and offer a functional interface.

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

Since the actual thesaurus generation is being conducted by a third-party component, grid exploitation is restricted to the word stemming which is conducted by the (included) "Process Text" UC (see there).

System Integration

"Thesaurus Generation" is a kind of "Text Feature Extraction" (see there).

Use Stories

See ARTE requirements document.

6.4.5 Feature Extraction

Description and Priority

The "Feature Extraction" Use-Case is the most general "umbrella" feature extraction component, which is a generalization of any media-specific feature extractor. However, the invocation of all feature extractors is conducted in the same manner, i.e. a media object (e.g. an image) is submitted along with a selection of the feature extraction algorithms (e.g. colour histogram) and its parameters (e.g. dimensionality of feature vector, colour space, language of speech). The feature extraction component picks the right media-specific feature extractors, which are likely to be "wrappers" for proprietary legacy components.

Functional Requirements

The "Feature Extraction" Use-Case will extract numeric or textual features from the content of an arbitrary media object. In detail, it will receive the media object, pick the right feature extractor and pass the media object to it (probably somewhere on the grid), collect the results (i.e. the actual features) and either return them to the calling query processing component or to the "Content and Metadata Management" component.

Numbers

This Use-Case will be one of the most frequently invoked functionality. That is, it is a fundamental building block of any search operation that involves some sort of content-based multimedia retrieval.

Constraints and Assumptions

The usefulness of a certain feature extractor to answer a particular similarity query heavily depends on the application domain. Currently, the ISIS prototype incorporates a number of low and medium level features that will be ported to the DILIGENT architecture. As we receive more formal requirements from the ARTE community, more adequate features may be included, depending on the state of the art in image, text, and audio processing.

UML Diagrams

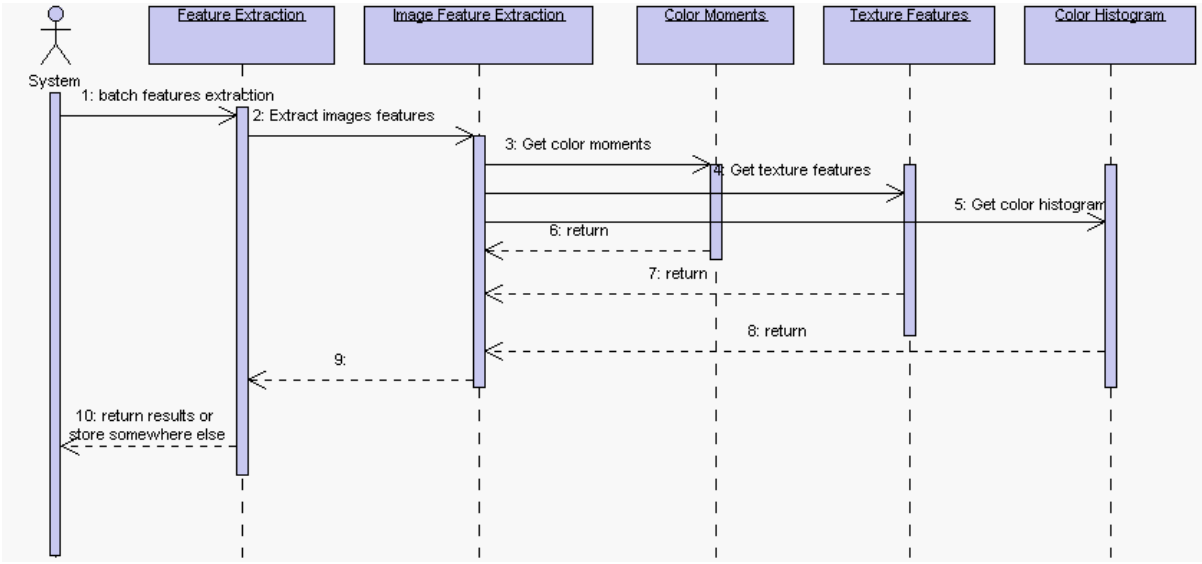

Figure 57: Feature Extraction (sequence diagram)

Grid Exploitation

In terms of batch processing of existing multimedia collections, grid capabilities can be exploited in a meaningful way. In particular, (1) there are no real time requirements, (2) feature extractors are, in general both CPU and I/O intensive operations (images are voluminous objects to be retrieved from storage, feature extraction may involve expensive operations like Fourier transformation etc.), and (3) feature extraction can be parallelized in an orthogonal way where (a) each object and (b) each feature may be processed on a different grid node.

System Integration

There will be a dedicated functionality for feature extraction, which is being invoked from various (1) search and (2) content and metadata management services.

Testing Issues

Testing must mainly check the incorporation of DILIGENT-external legacy feature extractor components and their interaction with the DILIGENT architecture.

Related Non-Functional Requirements

The feature extraction functionality in DILIGENT depends on (1) the existence, (2) license-free availability of legacy feature extractors which are present as (3) small components that (4) can be wrapped and controlled from external software.

6.4.6 Image Feature Extraction, Text Feature Extraction

Description and Priority

The "Image Feature Extraction" and "Text Feature Extraction" Use-Cases model media-specific feature extraction functionalities. They are specializations of the "Feature Extraction" functionality for handling images and text. Beyond the "Feature Extraction" functionality it provides mechanisms for image/text handling that are common in all *specific* image/text feature extractors (e.g. image normalization, transformation into joint file formats, common colour model, plain text extraction from other formats like MS Word, PDF, HTML, etc.). Specific image/text feature extractors (like "Colour Histogram", "Thesaurus Generation") are coupled via an extend-relationship *and* a generalize relationship). In other words, they are more specific functionslities picked depending on the kind of invocation of the "Image Feature Extraction" and "Text Feature Extraction" Use-Cases.

Requirements

These Use-Cases are responsible for feature extraction on all kind of image/text objects to answer all content-based image/text retrieval requests.

Numbers

These functionalities will be invoked on the query image(s)/text(s) whenever a "Search X by Image"/"Search X by Text"-style functionality is carried out.

Constraints and Assumptions

The actual usefulness of these functionalities for purposes stated in the ARTE requirements document depends on (1) the coupled specific feature extractors and (2) precise similarity measures, which must be provided by the user community.

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

See "Feature Extraction" Use-Case.

System Integration

The image/text feature extraction functionality is either invoked by the more general feature extraction functionality which basically picks the right feature extractor based on the media type or will directly by invoked by all "Search X by Image"/"Search X by Text" Use-Cases.

Testing Issues

See "Feature Extraction" Use-Case.

Related Non-Functional Requirements

See "Feature Extraction" Use-Case.

6.4.7 Audio Feature Extraction, Video Feature Extraction

The "Audio/Video Feature Extraction" Use-Cases have been introduced into our functional specification as placeholders for future additions of (desirable) Use-Cases from the ARTE requirements document.

6.4.8 Image Segmentation

Description and Priority

The "Image Segmentation" Use-Case has been introduced to reflect the user requirements of the "Process Image" and "Resolve Image into Parts" requirements. Both are part of the "Object Management" package and still present in our use-case diagram. They couple with

the "Image Segmentation" Use-Case via an *include*-relationship. The reason they are still to be found in our diagram is twofold. On the one hand, this shall facilitate understanding our revised diagram. On the other hand, the ARTE requirements document does not provide a clear distinction between the "Process Image" and the "Resolve Image into Parts" Use-Cases. We assume that "Process Image" is not restricted to image segmentation but provides other image operations (and might, thus, be exploited by the "Image Feature Extraction" Use-Case).

Requirements

See "Process Image" and "Resolve Image into Parts" Use-Cases.

Numbers

See "Process Image" and "Resolve Image into Parts" Use-Cases.

Constraints and Assumptions

See "Process Image" and "Resolve Image into Parts" Use-Cases.

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

See "Process Image" and "Resolve Image into Parts" Use-Cases.

System Integration

See "Process Image" and "Resolve Image into Parts" Use-Cases.

Use Stories

See "Process Image" and "Resolve Image into Parts" Use-Cases.

Testing Issues

See "Process Image" and "Resolve Image into Parts" Use-Cases.

Related Non-Functional Requirements

See "Process Image" and "Resolve Image into Parts" Use-Cases.

6.4.9 Colour Histogram, Colour Moments, Texture Features

Description and Priority

The "Colour Histogram", "Colour Moments", and "Texture Features" functionalities represent low-level image feature extraction algorithms, which are coupled to the "Image Feature Extraction" Use-Case via *extends*-relationships. That is, each may optionally belong to the description of that Use-Case.

Functional Requirements

These Use-Cases accomplish the basic requirements for content-based image retrieval, which matches images according to the overall colour distribution and texture properties. Each image that is passed to any of those Use-Cases will be analysed and processed into a high-dimensional vector of numeric features.

Numbers

See "Image Feature Extraction" Use-Case.

Constraints and Assumptions

See "Image Feature Extraction" Use-Case

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

See "Image Feature Extraction" Use-Case

System Integration

See "Image Feature Extraction" Use-Case

Use Stories

See "Image Feature Extraction" Use-Case.

Testing Issues

See "Image Feature Extraction" Use-Case.

Related Non-Functional Requirements

See "Image Feature Extraction" Use-Case.

6.4.10 Localized Colour Histogram, Localized Colour Moments, Localized Texture Features

Description and Priority

The "Localized Colour Histogram", "Localized Colour Moments", and "Localized Texture Features" functionalities represent low-level image feature extraction algorithms, which are coupled to the "Image Feature Extraction" Use-Case via *extends*-relationships. That is, each may optionally belong to the description of that Use-Case. Besides, they refer to the non-localized Use-Cases via *include*-relationships, which expresses that they apply the same functionality on partitions (defined with a fixed grid) of the image.

Functional Requirements

These Use-Cases accomplish the basic requirements for content-based image retrieval, which matches images according to the localized colour distribution and localized texture properties. Each image that is passed to any of those Use-Cases will be analysed and processed into a high-dimensional vector of numeric features. In contrast to the non-localized feature extractors, image matching on a localized features takes some geometric alignment information into account and, thus, exhibits better retrieval effectiveness.

Numbers

See "Image Feature Extraction" Use-Case.

Constraints and Assumptions

See "Image Feature Extraction" Use-Case

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

See "Image Feature Extraction" Use-Case

System Integration

See "Image Feature Extraction" Use-Case

Use Stories

See "Image Feature Extraction" Use-Case.

Testing Issues

See "Image Feature Extraction" Use-Case.

Related Non-Functional Requirements

See "Image Feature Extraction" Use-Case.

6.4.11 Face Detection

Description and Priority

The "Face Detection" Use-Case represents a medium-level feature extractor, which already conducts a simple semantic analysis of the given image. That is, faces in front shots will be detected. As the ARTE requirements does not state any mandatory Use-Case which takes advantage of face detection, this Use-Case should just be labelled desirable.

Requirements

This Use-Case is useful for any image depicting human faces which is highly useful in certain CBIR queries (potentially mixed with Boolean predicates on keywords), which seek for a certain person in a given image set.

Numbers

See "Image Feature Extraction" Use-Case.

Constraints and Assumptions

See "Image Feature Extraction" Use-Case

UML Diagrams

See Figure 56: Feature Extraction.

Grid Exploitation

See "Image Feature Extraction" Use-Case

System Integration

See "Image Feature Extraction" Use-Case

Use Stories

At the DILIGENT Rome meeting a demonstration of the ISIS prototype was given which included a complex similarity query searching for a certain person's images when an example image along with the person's name were provided. By including face detection into the query, images whose keywords match but that depict no person, could be excluded.

Testing Issues

See "Image Feature Extraction" Use-Case.

Related Non-Functional Requirements

See "Image Feature Extraction" Use-Case.

6.5 Personalization

Personalization refers to the ability of the system to export a behaviour that matches the personal needs and preferences of a user, i.e. virtually different for each and every user that accesses its functionality.

Usually, personalization is handled using profiles, i.e. "records" that contain information useful for adapting the system behaviour on a per-user basis. These profiles can be created and maintained by various means, usually on user / group actions, and are consumed when behaviour is to be exposed to the user.

Adaptation might refer to various topics:

- Conformance to the capabilities of the user client system (not always considered as personalization),
- Presentation (layout, colours etc) based on user preferences,
- Functionality configured by explicit user preferences
- Implicit (usually slight) deviation from standard operation by monitoring user activities
- Etc.

Topics that relate to personalization and fall in the DILIGENT scope are:

- Creation and maintenance of profiles
  - Automated (intelligent)
  - Manual
- Management of profiles
- Personalization of information retrieval
- Personalization of system interaction (presentation layer)
- Personal (and temporal) storage areas

Creation and maintenance of profiles for the DILIGENT will be manual, however the ability for future extensions that will support automated intelligent profile creation and updating will be there. On the other hand, the profiles will be available to be consumed by various services, however the personalization of information retrieval through will be the main objective of the DILIGENT platform.

Finally, although not part of the Index & Search group of services, personal (and temporal) storage is a desired functionality in order to allow users to create owned placeholders for Digital Objects.

Personalization of interaction falls in the application layer and is to be handled mostly by the portal engine.

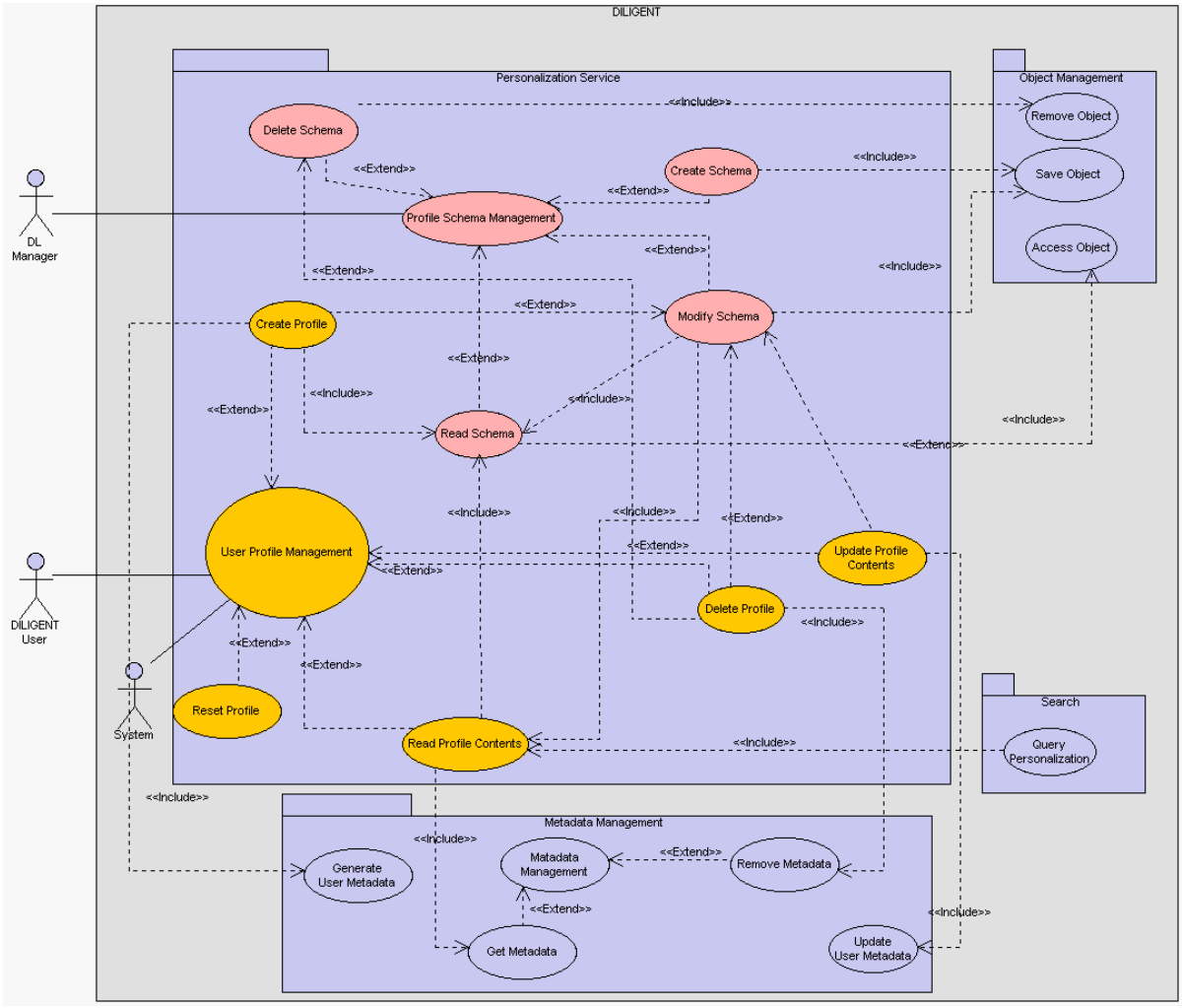

Figure 58: Personalization Management (use case diagram)

6.5.1 Profile Schema Management

This functionality gives access to the group of functionality that deal with the management of the container of the personal data related to the user's profile. Specifically, these operations create, modify and apply the structure of the profile container. Issues such as the way this information should be stored and exploited are the objects of this set of use cases. So this group includes the following functionalities:

- Create Schema
- Delete Schema
- Read Schema
- Modify Schema

Constraints and Assumptions

The service relies on storage services provided by the substrate. Each schema will represent the personal individuality of the system being configured under it and for simplicity it will most probably applied that an instantiation of DILIGENT as a VDL will be ruled by a single schema. In addition to that, simple transformations of schemas will be supported such as to add/delete fields or add/drop behaviour monitoring data tables etc

Regulations for managing and accessing personal data should be fully respected and supported by this functionality.

Use Stories

Only a high privileged user should be able to manipulate the schema of the data container. This means that only an "administrator" can trigger these actions. When the creation operation is triggered the system takes over to create a new schema, based on supplied parameters and after that it is ready to store data and metadata regarding the user's personal preferences.

When the modification operation is executed, the system changes the structure of the data container and when the "delete" operation is takes place the system saves the changes of the modified schemas.

The "read" operation is performed when the personalisation service accesses the user profiles for accessing stored information, on behalf of the user. Another potential user of the read operation is the Annotation Service. This might lead the search service to limit search in DL personal sub-domains by adding specific keywords to the query.

Grid Exploitation

Managing user's data has no significant needs for grid technology, other than the provided by the underlying storage service. Through these lower DLL services, the personalisation service will exploit transparent distributed replicated storage provided by the Grid substrate.

Related non-functional requirements

Underlying storage of user profiles exploits distributed storage management.

This service and the structures it manages should guarantee that personal data will be handled with respect to the laws and rules for safeguarding relevant regulations (privacy, security etc).

Testing Issues

There are currently no special suggestions for service specific testing.

6.5.2 Create Schema

Description

Creates the schema (e.g. fields, options etc) of the profiles to be used throughout a VDL[18].

The creation of the schema is an important process that must be executed in order to create a new schema for a specific user. Creation takes place when a well-authorized user (mainly the administrator) decides that some information plays a significant role in the process of characterizing a user and wants to take into account this information in this specific process.

Priority

This functionality is important for the administration of the personal data container and the provision of the Personalization Service. However it could be replaced by the provision of fixed profiles, which renders it as "conditional" in the context of Personalization.

The overall functionality is optional for the DILIGENT scope, due to the "optional" nature of personalization.

Requirements

- Support for custom user profiles.
- Respect regulations for personal information management.

[18] Multiple profiles will be considered at design stage.

Numbers

Only the administrators of the VDL are the ones to access this operation, which is to be infrequently used.

System Integration

This functionality is to be provided as part of the Personalisation Service of the Index & Search group of Services.

It accesses functionality provided by Storage and Metadata group of services (mainly related to Object Management) in order to handle its data.

6.5.3 Delete Schema

Description

The second functionality of this group is "profile deletion", which drops a schema of profiles. Using this, one can delete a schema that is no longer needed. Normally used by DL Administrator[19].

Priority

This functionality should only exist if the "Create" functionality exists.

The overall functionality is optional for the DILIGENT scope, due to the "optional" nature of personalization.

Requirements

- Support for custom user profiles.
- Respect regulations for personal information management.

Numbers

Only the administrators of the VDL are the ones to access this operation, which is to be infrequently used.

System Integration

This functionality is to be provided as part of the Personalisation Service of the Index & Search group of Services.

It accesses functionality provided by Storage and Metadata group of services (mainly related to Object Management but Metadata management involvement can be considered as well) in order to handle its data.

6.5.4 Read Schema

Description

This functionality is the one that reads schema. This is the one that is mainly exploited after the creation of a DL. With this functionality the system (or the user) can read the appropriate schema that needs so as to access the corresponding information stored in that schema. Within the context of Profile Management, the "Modify" functionality exploits the "Read".

Priority

This functionality is important for the exploitation of the personal data container and the provision of the Personalization Service. However it should only exist if the "create profile" functionality is included in order to tackle with custom profile schemas.

[19] Usefull if multiple profiles schemas are allowed on a single VDL.

The overall functionality is optional for the DILIGENT scope, due to the "optional" nature of personalization.

Requirements

- Support for custom user profiles.
- Respect regulations for personal information management.

Numbers

All DILIGENT users do indirectly access the schema retrieval functionality since it has to be consulted on every access to personalization data, in order to figure out the nature of the user specific personalization information held in the system repositories.

System Integration

This functionality is to be provided as part of the Personalisation Service of the Index & Search group of Services. It accesses functionality provided by Storage and Metadata group of services (mainly related to Object Management) in order to handle its data.

It mostly provides its services to Search, and in particularly the "Personalize Query" functionality in order to take any actions towards fulfilling a user's request meeting his/hers expectancies.

It is also being accessed by anyone who interested in managing a user profile, i.e. the personalization facilities of the portal.

6.5.5 Modify Schema

Description

The last functionality refers to the modification of an already existing schema, in order to better represent the new needs of the DL.

The modification of a profile schema is not just a sequence of read -> delete -> create. Actually a special operation of migrating user profile contents to the new schema must be applied. However in the DILIGENT scope this migration facility is entirely optional and if provided it will be kept to a minimal level.

Priority

The overall functionality level is "optional".

Requirements

- Support for custom user profiles.
- Respect regulations for personal information management.

Numbers

Only the administrators of the VDL are the ones to access this operation, which is to be infrequently used.

System Integration

This functionality is to be provided as part of the Personalisation Service of the Index & Search group of Services. It accesses functionality provided by Storage and Metadata group of services (mainly related to Object Management but Metadata management involvement can be considered as well) in order to handle its data.

6.5.6 User Profile Management

The User Profile Management case is subdivided into the following main cases.

- Profile creation case, when a new user is being added to the DILIGENT platform.
- Deletion of an existing user.

- Modification/Update of a profile is allowed (instead of Drop and Recreate)
- Resetting of a profile (initial values)
- Accessing the personal information store contents, which will be the most popular member of this set of functionality.

Requirements

Respect regulations for personal information management.

- Allow creation of user descriptions and subsequent management of it
- Allow tracking of user activities
- Support personal content stores
- Support personalised information retrieval
- Support personalised information presentation

Grid Exploitation

Managing user profiles has no significant needs for grid technology exploitation, other than the facilities already provided by the underlying storage services. Exploitation of user behavioural information or complex processing of profiles could be computationally demanding however these aspects will not be handled within DILIGENT.

Constraints and Assumptions

The here presented case, assumes that the user has already registered with the DILIGENT platform and that he has already been given a unique ID or some other kind of distinctive name.

Through an authentication and authorization facilities the user has already gained access in the system.

Additionally, there should be a storage resource to keep the personalized data. The process of managing and storing the profiles should be considered with security issues and the need to protect any sensitive data. These mechanisms already exist and should be exploited accordingly.

This use cases assume the existence of a profile schema management mechanism.

UML Diagrams

The functionalities are included in the general personalization service diagram (Figure 58: Personalization Management).

Use Stories

The invocation of the functionality related to profile management is triggered either by the administrator or the system itself. When a new user interacts with the system, a request for a new profile should be implied if the system policy permits the user to access the DL. In contrast only an explicit request for a deletion should result in the actual deletion of a user profile. The addition request could be served by an automated procedure or it could wait the administrator to take action. On the other hand the invocation of the deletion service is triggered either by the administrator or the user himself.

Read invocation is to be performed by various actors: Query Personalisation, Results Presentation, Personal Virtual Collection Management (optional?) etc

End users will mainly trigger modification of their profile, since it is expected that they will be able to interact with their profiles through application provided user interface in which they can determine their personal needs. Any user has full control over his profile so he can invoke any of the previous operations. In addition to that, system will (optionally) invoke the service to store data concerning user's behaviour (submitted queries, preferred results etc)

Testing Issues

There are currently no special suggestions for service specific testing.

Related non-functional requirements

Underlying storage of user profiles exploits distributed storage management.

This service and the structures it manages should guarantee that personal data will be handled with respect to the laws and rules for safeguarding relevant regulations (privacy, security etc).

6.5.7 Create Profile

Description

This functionality relates to the creation of a profile container, when a new user is being added to the DILIGENT platform instance.

After the invocation of this "creation" case, the system is enabled to monitor and process information regarding the personal preferences of that specific user. Apparently, this operation requires high privileges in order to take place. This means that only the administrator can trigger it.

Priority

The priority of the whole User Profile Management case is "conditional" within the DILIGENT platform. Within the "Personalization" scope, this use case is of "essential" level, which reflects the importance of tracking user needs and replying to user queries with information that is regarded as the most interesting to him.

Numbers

DILIGENT administrators might be able to use this functionality however it is expected that it will be infrequently used.

System Integration

This functionality is part of the Personalisation Service of the Index & Search group. Currently it is a standalone functionality related to user management. It accesses functionality provided by Storage and Metadata group of services (mainly related to Metadata management) in order to handle its data. However at design stage the case of directly using Object Management functionality will be considered.

6.5.8 Delete Profile

Description

This functionality relates to the deletion of a profile container, when a new user is being dropped from a DILIGENT platform instance.

Regarding the "deletion" case, apart from the user he, only the administrator has the authority to invoke it. When the operation is executed the specific personalized information for this user no longer exists in the DILIGENT platform.

Priority

The priority of the whole User Profile Management case is "conditional" within the DILIGENT platform. Within the "Personalization" scope, this use case is of "conditional" level, which reflects the relevant need of cleanup.

Numbers

All DILIGENT users (especially administrators) might be able to use this functionality however it is expected that it will be infrequently used.

System Integration

This functionality is part of the Personalisation Service of the Index & Search group. Currently it is a standalone functionality related to user management. It accesses functionality provided by Storage and Metadata group of services (mainly related to Metadata management) in order to handle its data. At design stage the case of directly using Object Management functionality will be considered.

6.5.9 Update Profile Contents

Description

In the updating / modifying case, information of the user's personal preferences should be gathered. This can be achieved through a process that could involve interaction with the user (e.g. filling in a form). Inside the scope of this the user is able to create filters that reflect own preferences. This includes preferences regarding ranking, searching categories etc. An alternative or additional means to do this is to passively monitor user's activities inside the system. Both of these are application-domain specific cases.

Apart from "general" descriptions and preferences, a user's profile might be capable of storing activity information, which could be exploited by an advanced Query Personalisation component. Storing such information will be optionally triggered by the system as a result to a user query submission or result-ser management. However these options will probably not be implemented as part of the DILIGENT personalisation related functionality.

Priority

The priority of the User Profile Management case is "conditional" within the DILIGENT platform. Within the "Personalization" scope, this case is of "essential" level, which reflect the importance of tracking user needs and replying to user queries with information that is regarded as the most interesting to him.

Storing user activity information is ranked as optional within the DILIGENT scope.

Numbers

All DILIGENT users might be able to use this functionality however it is expected that it will be infrequently used.

System Integration

This functionality is part of the Personalisation Service of the Index & Search group. Currently it is a standalone functionality related to user management and specific parts of the application layer. It accesses functionality provided by Storage and Metadata group of services (mainly related to Metadata management) in order to handle its data. At design stage the case of directly using Object Management functionality will be considered.

6.5.10 Reset Profile

Description

Through the resetting use case, the entire data of the specific user are set to default values that correspond to a neutral profile. The need for this could arise in extreme cases such as system failure or data loss.

Numbers

All DILIGENT users might be able to use this functionality however it is expected that it will be infrequently used.

System Integration

This functionality is part of the Personalisation Service of the Index & Search group. Currently it is a standalone functionality related to user management. It accesses

functionality provided by Storage and Metadata group of services (mainly related to Metadata management) in order to handle its data. However at design stage the case of directly using Object Management functionality will be considered.

6.5.11 Read Profile Contents

Description

Accessing the personal information store contents, which will be the most popular member of this set of functionality.

As already stated, optionally the service might be designed to be capable of storing user behaviour information, however accessing and using such information in a sophisticated manner will not be applied within DILIGENT. This specific area is highly active in modern research in Information Systems however DILIGENT application scenarios did not show need for such advanced facilities.

Priority

The priority of the User Profile Management case is "conditional" within the DILIGENT platform. Within the "Personalization" scope this case is of "essential" level, which reflect the importance of tracking user needs and replying to user queries with information that is regarded as the most interesting to him.

Processing this information through Computational Intelligence algorithms is optional and is not being handled as part of the personalisation service.

Numbers

All DILIGENT users will indirectly access the profile retrieval functionality since it has to be consulted on every access to personalization data.

System Integration

This functionality is part of the Personalisation Service of the Index & Search group. It is being used by the application domain components (e.g. the portal engine, specific portlets etc) and the query personalization component that needs this information to personalize the search procedure. It accesses functionality provided by Storage and Metadata group of services (mainly related to Metadata management) in order to handle its data. However at design stage the case of directly using Object Management functionality will be considered.

6.6 Search

The "search" functionality is one of the most fundamental for a Digital Library Management system and it is the main "front" end of the DLL layer to the application layer, along with the content-related group of services.

Many different use-cases refer to the search functionality for different types of search. However it is our intention to create an extendible open yet "all-in-one" search engine that will uniformly cover all types of searches, be it for content or resources, similarity based, field matching based, etc.

Under this perspective the functionality to be supplied is a single one, the "Search".

6.6.1 Functionality Breakdown

Despite this, the Search functionality is a single function consumed by various system components and the end users (mainly through the portal), when seen under different perspectives one can define the a set "partial functionalities". These will not be further analyzed here, and for maintaining compatibility with the rest of the document they will be presented in a formal way in the next section.

6.6.1.1 Search for Documents Using Criteria

This is the case when a user wants to locate a document using typical criteria. This is a typical case of structured or semi-structured (depending on XML schema) search. These criteria depend on the metadata supported by the VDL setup in question and typically could include:

- Creation date
- Author
- Type of content
- Classification items
- Quality attributes
- Etc.

The system will provide typical logical operators for handling the content.

This type of search is mandatory for the system.

6.6.1.2 Search for Documents Using Textual Description

This is the case when the user wants to locate a usually text-containing document by supplying some words that he/she expects to be located in the document. This is a typical unstructured type of search usually found under the term of "full text search", however the specific approach to be implemented may vary significantly with regards to features.

This type of search could be possibly combined with the structured criteria-based search.

6.6.1.3 Search for Similar Documents

Another use case includes the search for similar documents when one or more prototypes are being supplied. Search by similarity is yet another type of unstructured search, where a set of documents are being processed and then the values of specific attributes are being combined to form a matching index usually in a slightly more arbitrary manner than in the full text search case.

6.6.1.4 Search for Archives by Content

This case refers to the ability to search for archives that contain documents that match a set of supplied documents. In the most cases this is a highly unstructured type of search and is based on supply of documents (e.g. images) and the already described search by similarity. An altermnative structured search for Archives is handled in the section for "Search for Resources / Objects".

6.6.1.5 Search for Documents referring to Geographical Area

The search for data using geographical information is a quite complex use case to deal with. Typically it could be refined to be a simple criteria-based structured form of search, by obtaining a set of coordinates and performing a set of comparisons based on geometrical operators. This would clearly be the case of a Geographical Information Management System and such functionality could be integrated to DILIGENT. However users are not expected to provide generally geocoded information to the system. Nor can the system deal with geocoding arbitrary information, because this would be a far-beyond-scope feature extraction topic. So spatial search will be handled as a mixture of full text search, when referring to geographical areas by name, and pure spatial search. Other woprkarounds could possible be defined in a per-application-case fashion.

6.6.1.6 Search for Resource / VDL Object

Both end users and the system require to be supplied the facility have an internal search capability, in order to locate items in use. It is expected that this will be a solely structured criteria-based search, to be performed on object / resource metadata. Typical searches to be performed are:

- Users
- Supported Services
- Computing Elements
- Storage Elements
- Archives
- Etc

6.6.2 Search Functional Definition

Description

The role of the Search functionality is to process a user provided query and provide back adequate information to locate and describe the Digital Library Objects that meet the user query constraints. The functionality described here offers an aggregation of all search capabilities tracked by the user requirements documents as well as the experience of the DILIGENT platform architects.

By the term Digital Library Objects we intend to avoid limiting search to Documents hosted or tracked by the Digital Library Management System (i.e. DILIGENT). Search can be performed on Users, Services and other types of resources using many of the facilities provided in typical search for documents.

This broadening of vision, allows search to be exploited by many more use cases other than the expected portal search portlet.

On the other hand many different types of search are being supported. Due to lack of fully satisfactory terms we will introduce the terms below, which might be revised at later stages of the project:

- Semi-Structured Search, deals with search on matching semi-structured data. A typical type of search that fits this category is search in XML based on xpath and other relevant technologies.
    - Fully structured search (or simple structured search) is a sub-category of semi-structured search and is based on field matching via typical comparison and containment operators.
        - Spatial search is a sub-category of structured search[20], which offers operators to calculate overlapping of geometrical shapes.
- Unstructured search, which is usually based on the calculation of an index that indicates the degree which two different samples of documents match one another[21].
    - Similarity search is the most common type of search to be performed under this category. "Similarity search" is usually[22] a "two-step" procedure, which invokes the feature-based search after a feature extraction performed on a supplied prototype (or a set of them).

[20] Spatial search can be build under the Semi-structured search, however for the DILIGENT platform, spatial search will be based on well-defined information matching.
[21] Spatial search can be also built under this concept, however this falls out of DILIGENT scope.
[22] Based on the DILIGENT use cases and existing technologies.

- Full-text search is a rather well defined class of similarity search. However full-text search can greatly benefit from structured data existence, under proprietary implementations.

At this point we have to clarify that

- The various types of search are usually not mutually exclusive.
- The unstructured search algorithms can make use of structured data existence[23].

Search has to meet user expectancies in terms of quality and performance. User requirements showed that users expect to submit a query that might be evaluated over various collections of digital content and provide back the results with minimal delay, clearly proposing the most suitable ones either by estimation or by user provided rules. Additionally users expect that they will receive the best suiting content with minimal search parameter submission, sometimes through searching quite heterogeneous data sources, in terms of content and structure.

It is also clear that the search facility has to be quite extendible in terms of services and facilities it consumes. The diversity of applications to be potentially supported by an open Virtual Library Management System inserts the requirement that many parts of the search facility are open to extension / customization, in order to meet each application-domain-specific need. A typical example is that an image content managing VDL should provide rich functionality to process images while one dealing with Sound or Video objects should present different capabilities.

Apart from the above, background knowledge on query processing, especially in a complex dynamic environment such as the one to support DILIGENT, indicates that extreme mis-utilization of resources might occur if not careful planning precedes the actual query processing.

Although the "search service" offers just one end-user accessible entry point ("function"), the "Search" it is internally broken down into many more components[24]. This approach is roughly presented in bullets bellow, which reflects the internally needed functionality:

- Query parsing: Get the structured query and analyse its syntax and operators and internally represent it in a machine friendly form. This form is most likely to be a tree-like structure expressed in XML (quad query).
- Broker L1, is a front-end broker to select the search engine which is to carry out the search (optional). Design stage might show that search should not be 1st level invoked service but rather a composite workflow hosted by the Process Management Service (optional)
- Source Selection (L1), is a pre-selection of resources to be sought by a specific search, based on query supplied constraints, which might be an output of personalization or user actions. It is clear that this is not based on the intelligent facilities provided by CSDS service.
- Content Source Selection (L2): Interface with the Content Selection Service to select the appropriate system to receive the query. Content sources might be collections or external DLMSes (optional facility).
- DLMS Adapters: transform the quad query to the actual language understood by a specific DLMS, if query is submitted out of DILIGENT control space. (Functionality is optional)

[23] A most obvious case is the Full-text search which can make use of special tags that denote information such as title, author, subject etc

[24] Not all components are related to functionality thus some of them are not being presented to the diagrams included in this functionality.

- Query Optimiser: process query and provide and optimised execution plan based on: operators and search space properties, resource state (optional), data collection attributes and volumetrics, system configuration, QoS constraints (optional) etc. This facility is complementary to the Process Optimisation however it bases its operation on different classes of information.
- Broker L2: Submit the work to actual DILIGENT execution services i.e. Process Management Service (functionality of brokering is optional at this level).
- Executive: carry out the details of search, i.e. index reads, ranking, invocation of feature extraction and transformation components etc, indirectly invoked by the Process Management Services as part of the Query Process Workflow. This group of functionality (Execute) is
- Result formulator: prepare output and return it to end-user via a standard protocol. Extra metadata might be added to the resultset in order to be reused for further processing.
- Linguistics & Semantics Processing: refers to a group of facilities that deal with normalization of text queries and semantics base search. This use case bases its functionality on support provided by underlying metadata layer services and ontologies[25] (or taxonomies). The facility is optional to the DILIGENT platform, and it is likely that it will mostly left blank. However placeholder for such class of processing will be supplied[26].

Priority

Search it is one of the two most important DL functions. The other one is the digital object storage management.

This justifies that the priority of the Search functionality is considered of "essential" level within the whole DILIGENT Platform. However some of the internal facilities to be proposed or provided have lower ranking that might reach even level "optional".

Requirements

The Search service is the front-end of the DILIGENT platform to the end user application. Its functionality is directly derived from user requirements. However not both user scenarios posed the same requirements on the service operation. Aggregating both of them gives out a list of requirements enumerated below:

- Provide results sorted by specific fields of the referenced digital object (sorting)
- Multiple search capabilities (see Description section)
- Reasonably performing ranking function.
- Provide content description information back to end-user
- Support complex content processing
- Utilise provided and generated digital object features to perform search.

Numbers

Being one of the two most important DL functions, the search functionality is likely to be accessed by each and every user accessing the DILIGENT.

Additionally the cardinality of the operation implementation instances in the DILIGENT platform is 1-n. Absence of the operation leaves the DL almost unusable.

---

[25] If these components are implemented in the DILIGENT scope, these structures will be manually populated.

[26] The "parsing" hosted placeholder is an obvious junction point, however exploitation of this facility during "Matching" is also expected.

Constraints and Assumptions

Search will build on top of a series of facilities of the DLL and underlying layers. The facilities have to provide enough support for the service to execute. For example, lack of a process management component would render the actual execution of a Search Query Workflow impossible.

A basic assumption however is that an extendible structured query language (not to be confused with SQL) will be adopted. This does not need to be user friendly. An XML-related "protocol" will most probably become the basis of the language.

On this presentation we assume that the plugging of components, and content sources are not part of a hypothetical Search Administration facility. Under this point of view this is non-existent. They are rather implemented as part of the lower Resources and VOs Management packages.

Another assumption is that Personalisation Service is not integrated to the search; rather it acts on its own. This opens a wider range of capabilities to link to external sources. The "injection" of personalisation information is done prior to the actual search, by adding terms to the end user query, an operation performed by the application domain specific application. However a different approach could also be as valid as this one.

In order for a successful search to be performed, the system should exploit and cope with:

- Information related to configuration, resources availability and capabilities etc (Operations supported, ranking approach, operators, resource existence, capabilities and utilisation, content sources etc)
- Indices and relevant facilities
- Digital object management
- Invocation of processing components addressed by the query.
- etc

Additionally a standard (probably XML/SOAP based) must be adopted to return result sets to end-user.

UML Diagrams

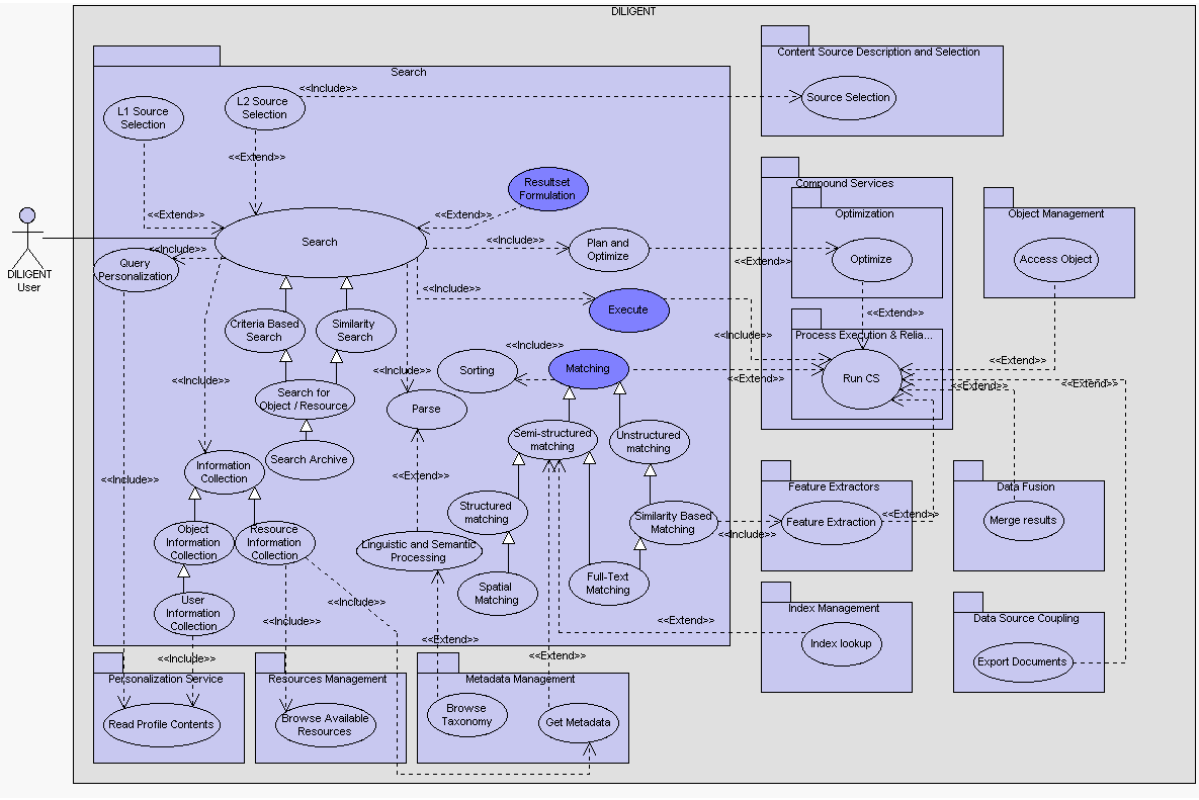

Figure 59: Overall Search Use Case

In Figure 59, we present the overall functionality and logical organization of the search engine and its connection to various external services. On this diagram the dark boxes represent blocks of functionality that will have the responsibility of creating resultsets that will ultimately form the outcome of the operation.

The sub-functionality of Query Optimisation, which is part of the Process Optimisation, is depicted in Figure 69: Optimize Process Use Case.

Grid Exploitation

The search function processing and storage requirements mainly depend on the complexity and size of the supplied query and underlying Digital Content collections. Other factors include the complexity of operators and the availability of resources etc

Both storage and computing power supplied by the Grid are most likely to be well exploited under the DILIGENT platform search operation.

Finally, the optimisation component will be aware of resources and state by consuming Grid supplied information services through DILIGENT CL Services.

System Integration

Search functionality mainly belongs to the Index and Search group of services, however being one of the top layers of the DILIGENT platform, heavily depends on and influences logically lower level services. Under this observation use cases of search include many references to external components:

- Accesses metadata through functionality provided by the Metadata Management.
- Accesses digital objects through functionality provided by Object Management .
- Consumes taxonomies provided by the Metadata Management services.
- Processes documents through the Feature Extraction services.

- Merges resultsets through the Data Fusion components.
- Runs search under the Compound Services functionality (Run CS).
- Locates documents using the index services.
- Selects content sources through CSDS service.
- Personalizes queries and results through the Personalization service
- Retrieves information on system capabilities, configuration and state using DILIGENT collective layer based information sources, via Metadata Management.

Search is consumed by:

- End user through a portal or any other non-DILIGENT provided front end in order to retrieve a list of DL hosted objects.
- Other system services in order to locate items (objects / resources) managed by the DILIGENT platform for internal use.

Use Stories

The main case of invocation of the search functionality is directly triggered by end-user actions. However, alternatively the service could be triggered by external systems that query a DILIGENT hosted Digital Library for content.

The functionality of Search is consumed through an application domain specific query submission front-end. In the DILIGENT project case, the corresponding portals will provide this. Nevertheless, expert users or other systems will be able to access the search facility directly submitting queries using the DILIGENT structured query language, which is most likely to be one of the well known "standard" query languages, adapted or extended with facilities to capture the DILIGENT set of operators and requirements.

The service receives an end-user query, pre-processes it, selects the content sources that are most suitable for fulfilling user expectancies and retrieves their contents based on a workflow of operation which might range from simple selection to highly sophisticated Computational Intelligence related processing.

Testing Issues

The overall performance of search facility must be tested and evaluated in terms of

- Quality, i.e. the distance of the provided search outcome when compared to a hypothetical best matching result set.
- Efficiency, i.e. resource utilisation by the service itself compared to a hypothetical optimal one
- Correctness, i.e. safeguarding the semantics of the supplied search.

Especially in non exact-match functionality, the degree of matching the end-user expectancies (ranking) must be tested and evaluated.

Related non-functional requirements

High priority non-functional requirements include:

- Efficient resource utilisation
- Fast response
- Utilise dynamically linked content processing components such as feature extraction components and transformations.
- Adoption of a "standard" query language to submit queries, extended as appropriate to cover DILIGENT and DLMS specific needs.

Second priority level requirements (Wishes) include

- Dynamic exploitation of existing resource information.

- Support of highly interactive application scenarios (minimal response time over a complex dynamic environment)

6.6.3 Query Personalization

Description

Query Personalisation is the action of injecting information in the query in order to provide end-user with personalised results. Actually this functionality is the main "user" of personal profile. Its operation consists of processing a quad query prior to submitting it to execution. In an advanced form the component to provide the functionality understands specific terms of the query and injects more filter or ranking criteria or adds "hints" and formatting information. The rest of the procedure will be carried out by the remaining search components. A more highly advanced personalisation service could also operate on lower levels in higher detail however this would have a significant added value for the DILIGENT platform.

A side effect of the functionality would be the addition of a record to a log of user submitted queries containing the query or a summary of it.

Priority

This service is ranked as "conditional" in the DILIGENT context.

However the priority of the service is "essential" within the overall personalization service. Lack of it leads to failure to personalise the search however personalisation can be still applied to other aspects of the system, such as personal collections, presentation preferences etc.

Requirements

Respect regulations for personal information management.

- Support personalised information retrieval
- Support personalised information presentation

Numbers

All DILIGENT users will indirectly access the query personalization functionality since it has to be consulted on every search to personalization data. Administrators will access the rest of the functionality.

Constraints and Assumptions

This functionality assumes the existence of user profile management mechanism.

Query language syntax should provide sufficient means to receive personalization information.

Service exposes vulnerabilities to "personal data".

UML Diagrams

The functionality is included in the general personalization service diagram (Figure 58: Personalization Management).

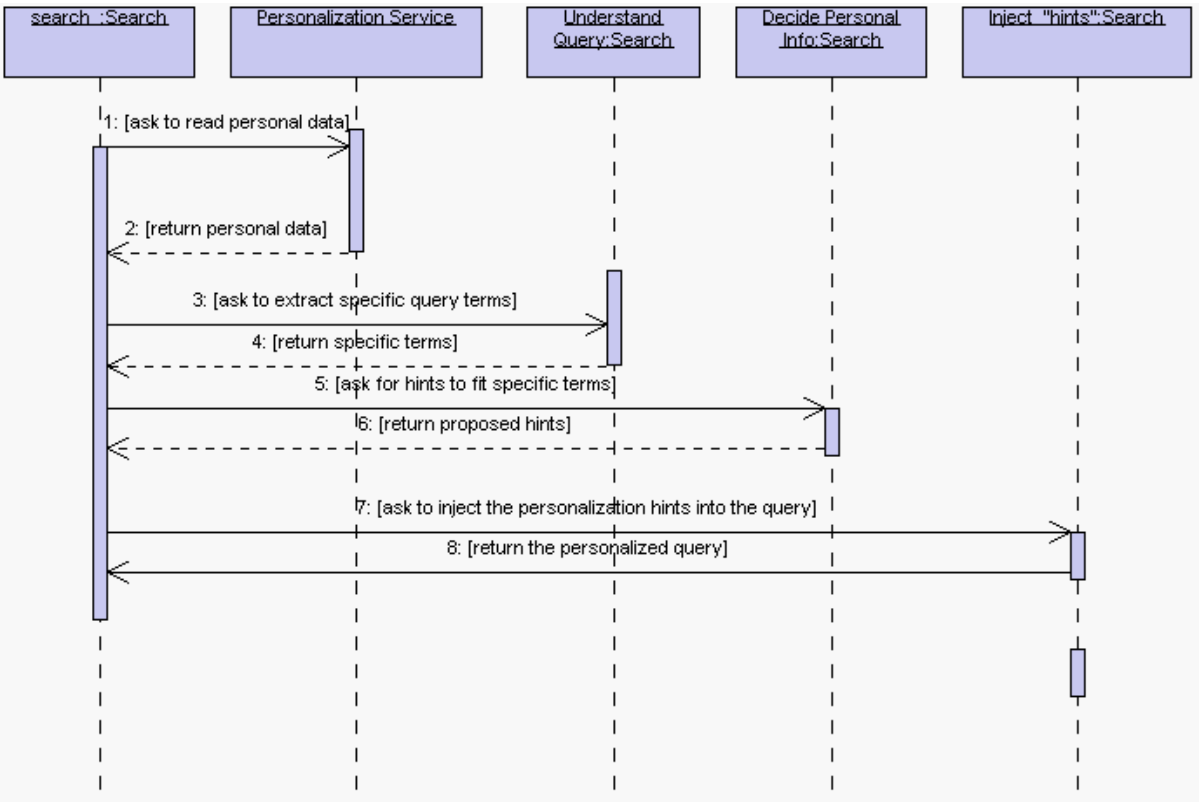

Figure 60: Query personalization (sequence diagram)

Grid Exploitation

Personalizing a query does not expose significant grid exploitation needs.

System Integration

This functionality is part of the Personalisation Service of the Index & Search group

Use Stories

The invocation of this functionality is triggered just before the query processing. Actually the user profile could also be attached to the query for future processing by enhanced personalisation components; however the initial personalisation implementation will not handle that.

Testing Issues

There is a need to validate the query semantics preservation after the "personalization" facility.

Additionally the overheads that are introduced by personalization and the benefits are to be evaluated.

Related non-functional requirements

- Avoid overloading system-processing requirements due to personalisation information.
- Preserve query semantics.

7 PROCESS MANAGEMENT

This section provides the functional specification of the process management-related services (process design & verification, process execution & reliability, and process optimization) as part of WP1.5 ("Process Management").

7.1 CS Management

Based on the ARTE and ImpECt user requirements documents, the following section summarizes the functional specification for the management of compound services. As can be seen from Figure 1, giving an overview of all use cases described in this section, the use cases are grouped in four packages "Process Design & Verification", "Optimization", "Process Execution & Reliability" and "Resource Management". The first three packages are described in detail within this section, whereas the details of the latter, Resource Management, contains use cases which the others interact with, but which are not part of process management.

There are three actors, "DL User", "DILIGENT Administrator" and "DILIGENT" Service that have to be distinguished in interaction with the described use cases. Two types of compound services exist. First, there are "User CS" which can be designed and run by any user of the DL having sufficient rights assigned. Second, there is a set of compound services the overall system of DILIGENT relies on and which are triggered by the system itself, i.e. DILIGENT services, and which can only be managed by a DILIGENT administrator.

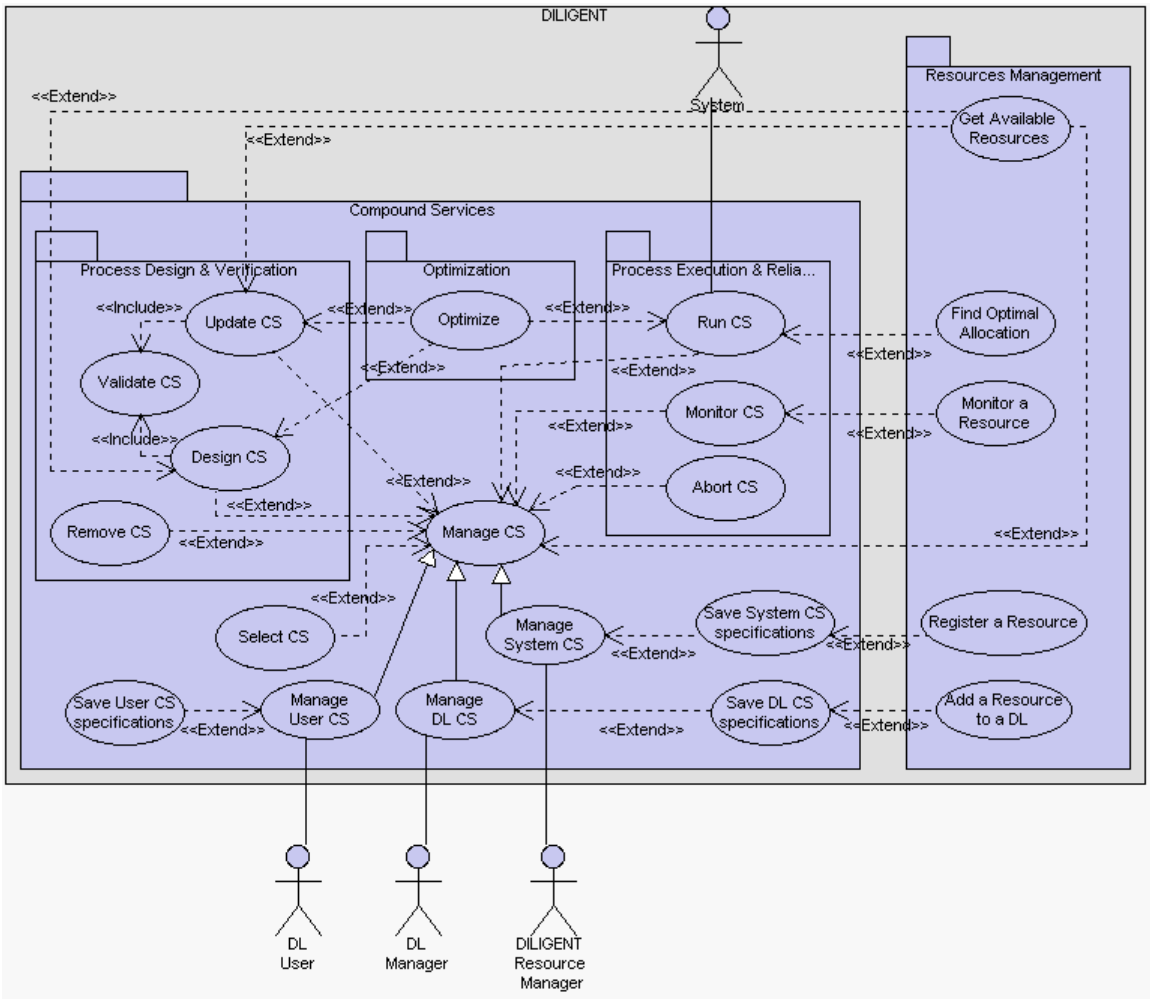

Figure 61: CS Management (use case diagram)

7.1.1 Manage CS

Description and priority

"Manage CS" is the most important use case, since all the other use cases can be seen as a special case of it. In particular, it comprises both the necessary services for designing and verifying processes (the terms 'compound service' and 'process' can be considered as synonyms; in order to be compliant with the DILIGENT DoW, we will use the term process in the following) and also for executing processes in a reliable way.

Due to the abovementioned characteristics, this use case is very important since it includes all services needed for process design & verification and also for process execution & reliability.

User Requirements Fulfilled

This functionality fulfils the ImpECt_ucd08 'CSs Management' UC.

Numbers

This use case is invoked several times per month (according to the requirement document). Actually, since it comprises core functionality for the definition and especially for the management/execution of compound services, the service has to be designed and implemented in a way that it provides a very high degree of scalability.

Process design and verification services can be performed in online (i.e., interactive) mode only, process execution should allow both interactive and batch mode.

Constraints and Assumptions

The process workflow includes: user interaction, interaction to the authentication component/service, service selection (based on user input preferences). This means that appropriate service management and access to existing services is required (both in terms of process design and execution).

It assumes that there exists a DILIGENT portal that permits the operations described above. Moreover, there must exist an Information Service that extracts the currently existing information about the available nodes, their configuration and their current load. Thus, the execution of a certain process should be performed based on this information. A Broker Service (as requested in the functional specification document for work package 1.2) to implement matchmaking algorithms between service requirements and nodes availability and resources is also required. These two functionalities are particularly useful for our "Run CS" use case.

UML Diagrams

See Figure 61: CS Management.

Grid Exploitation

Management of compound services / processes is essential in a grid environment. Especially in terms of process execution, a core feature is efficient resource consumption, i.e., to invoke a (basic) service at a provider that has sufficient resources or that is less loaded compared to other providers of the same service. Therefore, tight interaction with load balancing support and dynamic service selection capabilities of the underlying grid environment are crucial.

Mapping between functionalities and DILIGENT services (system integration)

Some operations of this use case are part of the Process Design and Verification Service and of the Process Execution & Reliability Service (details see below).

Use Stories

The detailed use stories will be given in the specification of the sub-cases of CS Management (details see below).

Testing issues

The detailed testing issues will be described in the specification of the sub-cases of CS Management (details see below).

Related non functional requirement

All process-related services need to be realized in a way that they provide a very high degree of reliability. This is particularly true for process execution and reliability. However, also process design and verification is affected, since alternative execution and failure handling strategies have to be addressed already at process build time (definition of compound services).

7.1.2 Design CS

Description and Priority

This use case is responsible for identifying the appropriate services and building a compound service that includes these services. Configuration and compatibility checks must be possible. On-the-fly validation of the newly designed compound service is required.

Two situations can arise: the user can manually select from the list of available services or the services that will form the compound service are the output of a query (in which case the user has to specify input data, functionality, desired output etc.).

Once a compound service is designed, it can be saved/referenced for later use (see "Update CS"). Besides defining and configuring the compound service, the user is able to specify the results to be saved (log files, output, intermediate files) and also environment requirements.

Requirements

The "Design CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case. Note that this use case also comprises the functionality of the "Create CS" use case given in the ImpECt requirements document; the decision to merge them was taken because their functionality does not seem to justify the existence of two separate use cases (where "Create CS" would include "Design CS").

Numbers

This use case is invoked several times per month (according to the requirement document).

Constraints and Assumptions

Although the described functionality does not require an Authorized User to design a compound service, it may be desired that access is not granted to any user to browse and select from any list of available services.

The requirements document defines the "Design CS" use case result as the creation of a new compound service by the selection of some of the available services. Although it may be desirable that the configuration and definition of an already designed compound service is stored for later use, one should have in mind the case when one or more of the services involved is no longer available and implement proper failure handling mechanisms.

Since on the fly validation is required by the ImpECt scenario, we also introduce the "Validate CS" functionality. This functionality is described in the "Validate CS" use case

At design time, a set of sharing rules should be established, concerning who can use the process, under what conditions and so on.

UML Diagrams

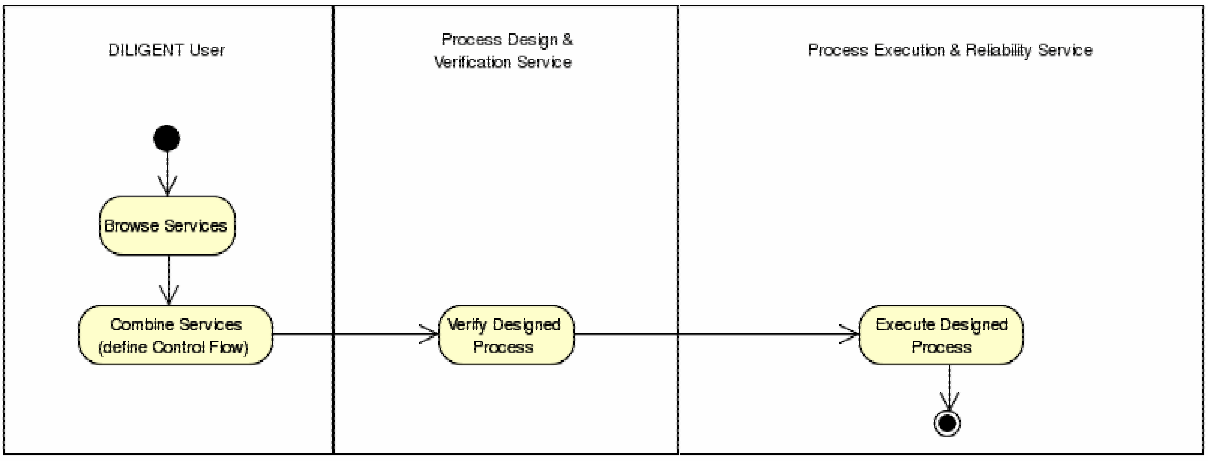

Figure 62: Design CS (activity diagram)

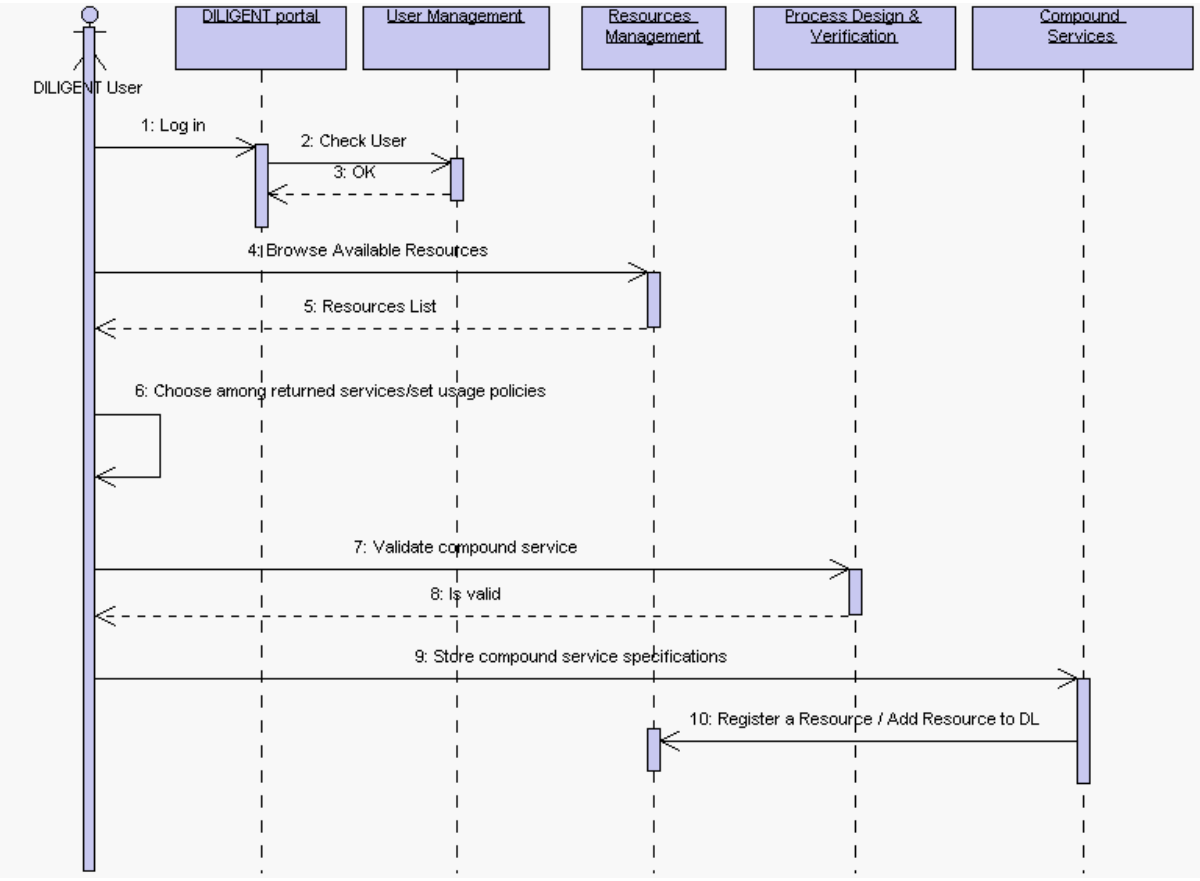

Figure 63: Design CS (sequence diagram)

Grid Exploitation

Usage scenarios of the "Design CS" scenarios may occur in *online* (i.e. interactive) use only. There is no need for data intensive or computer intensive tasks, but real-time performance is desired. Since the service selection relies on search functionalities (browsing a list of already available services), depending on the implementation of the latter, and on where the information about the desired type of services is stored, task parallelization may also be desirable.

System Integration

This operation is part of the Process Design and Verification Service.

Use Stories

One possible use story for this use case is the graphical modeling tool of the ETH-UMIT hyperdatabase prototype (OSIRIS), called O'Grape, for easy composition of web services and/or grid services into processes. The so-called whiteboard of the current process contains the global variables of a process instance. During process execution, the whiteboard of a process instance is first filled with the process arguments. The activities of the process, i.e., service invocations are drawn as nodes in a process graph. For each activity, the designer can choose among the list of services and processes known in OSIRIS. A service invocation retrieves its input parameters from the whiteboard and maps output parameters back to the whiteboard. Further, arcs between activities define the control flow of a process.

Testing Issues

Testing of this service includes the usability of the graphical design and verification front-end.

Related non-functional requirements

Security issues should be taken into account here, relating to which user is authorized to invoke which service.

7.1.3 Create CS

Description and Priority

Once a compound service is designed, it can be saved/referenced for later use (see "Update CS"). Besides defining and configuring the compound service (see "Design CS" use case), the user is able to specify the results to be saved (log files, output, intermediate files) and also environment requirements.

Requirements

The "Create CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case.

Numbers

See "Design CS" use case.

Constraints and Assumptions

See "Design CS" use case.

UML Diagrams

See "Design CS" use case.

Grid Exploitation

See "Design CS" use case.

System Integration

See "Design CS" use case. From the process management point of view, this functionality does not differ from the "Design CS" use case.

Use Stories

See "Design CS" use case.

Testing Issues

See "Design CS" use case.

Related non-functional requirements

See "Design CS" use case.

7.1.4 Update CS

Description and Priority

This use case allows a user to modify an existing process, e.g., change its design, its services, its input data or execution parameters like the OS in which it has to run.
The modified CS has to be verified using the "Validate CS" functionality before the changes can be committed.

Requirements

The "Update CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case.

Numbers

See "Design CS" use case.

Constraints and Assumptions

See the "Design CS" use case for remarks regarding authentication and authorization. An issue that has to be addressed additionally concerns the online modification of currently running processes: one can expect that an on-the-fly modification is not possible if the CS is being substantially modified; the process has to be stopped, the modification has to be delayed or cancelled (in some cases, it may be possible to migrate a running process to the new version).

Finally, if a CS is itself part of another CS, then it may not be modified in a way such that its semantics change.

UML Diagrams

See "Design CS" use case.

Grid Exploitation

See "Design CS" use case.

System Integration

See "Design CS" use case.

Use Stories

See "Design CS" use case.

Testing Issues

Test cases will be similar to the tests for the "Design CS" use case. Additionally, modifying running processes and modifying processes that are part of another CS will have to be tested.

Related non-functional requirements

See "Design CS" use case.

7.1.5 Validate CS

Description and Priority

This use case provides validation functionality; it performs compatibility checks between the services chosen to form a compound service. Validate CS cannot be derived directly from the user scenarios but will provide core functionality for process design and verification

service since it allows to make sure that process specifications are inherently correct and can be executed accordingly.

Requirements

See "Design CS" use case.

Numbers

This use case is invoked every time one of the "Design CS" or "Update CS" use cases is executed; according to the numbers given in the requirement document for these use cases, this is expected to be several times per month.

Constraints and Assumptions

For proper validation, compound services may not contain loops, and knowledge about the side effects of all involved services is assumed. Whenever a CS produces side effects, compensation strategies or alternative branches within the compound services have to be present.

Grid Exploitation

See "Design CS" use case.

System Integration

This operation is part of the Process Design and Verification Service.

Use Stories

Validation is a mandatory part of each "Design CS" and "Update CS" operation. These operations may only be committed if validation of the CS was successful.

Testing Issues

Testing of this component should focus on the correctness of the verification algorithms; the assumptions mentioned in the "Update CS" use case also need special attention.

Related non-functional requirements

None.

7.1.6 Run CS

Description and Priority

The "Run CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case. It is responsible for actually executing a compound service. Once the process is launched, it can be monitored or aborted. The Run CS use case is the main use case for the Process Execution & Reliability Service.

Numbers

This use case is invoked several times per month (according to the requirement document), and it can be performed in either batch or online (i.e. interactive) mode.

Constraints and Assumptions

In this case the requirements document is specifying as a precondition to the use case execution that the user is authorized to launch the compound service.

We also propose a scheduling functionality, for batch processing.

The "Run CS" component should first interact to an Information Service in order to obtain information about the available nodes, their loads and system configurations. A Broker Service is then matching from the list of available least loaded nodes those that satisfy the requirements of the process to be run.

For a better process management, scheduling automatic runs of processes may be implemented. For batch processing, some compound services may be scheduled to run when the infrastructure is less used or not used at all. The input data and output data format should be specified. Except for the case when online monitoring is desired (e.g., for testing purposes), batch processing can be done at a time when the system is least loaded, in order to optimize execution times. In addition to explicitly started processes by DILIGENT users and batch processes, also automatic (system) processes have to be supported. These automatic processes (designed by a DL administrator) have to be started automatically whenever consistency within the DL has been violated and needs to be restored (e.g., when, in the case of replicated repositories, one repository is updated, the changes have to be propagated to the replicated repositories; this propagation has to be transparent to DILIGENT users).

UML Diagrams

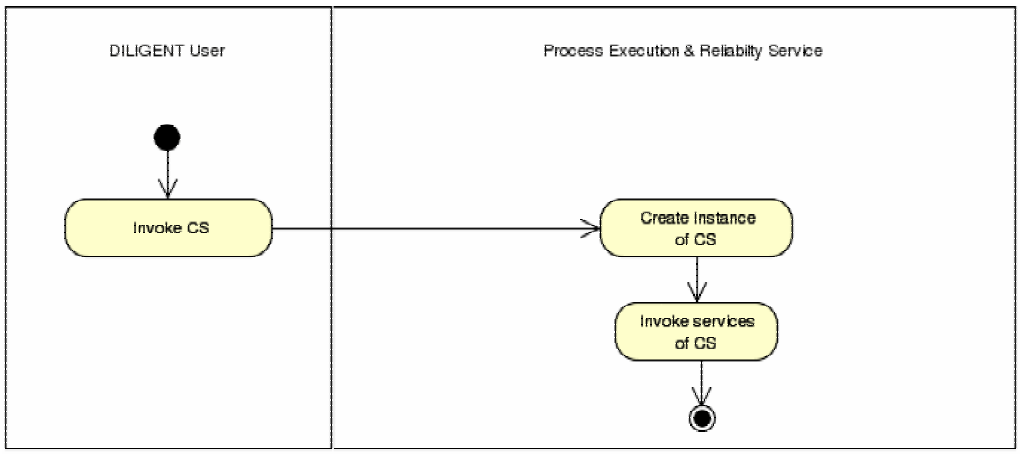

Figure 64: Run CS (activity diagram)

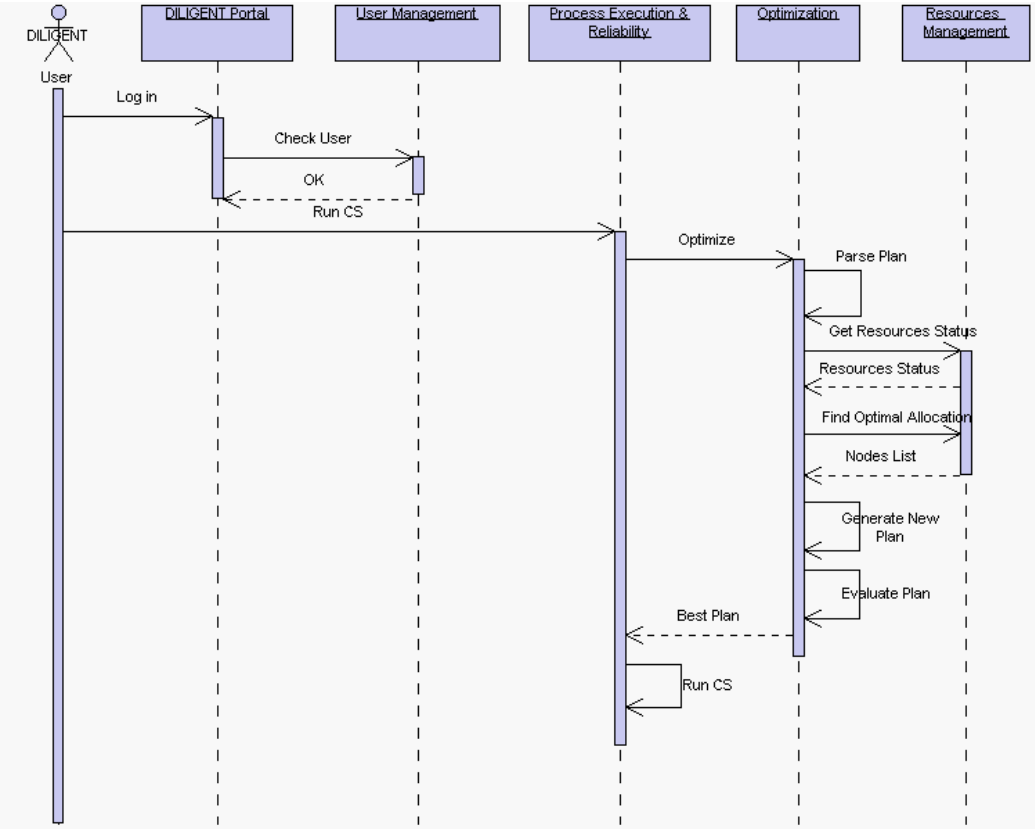

Figure 65: Run CS (sequence diagram)

Grid Exploitation

Especially when the usage scenarios of the "Run CS" scenarios may occur in batch mode, there is need for data intensive or computer intensive tasks. In some cases real-time performance is also desired. Depending on the complexity of the compound service, its throughput and/or execution costs, task parallelization may also be desirable. A major aspect is that this service must be provided in a highly reliable way. Failures of service providers have to be automatically resolved. For this purpose, three approaches (with different semantics) exist: i) choose another service provider offering the same (or semantically equivalent) service, ii) choose an alternative execution strategy (that has been defined by using the process design and verification service), or iii) invoke the process optimization service that will alter the overall process specification.

Use Stories

This service will be used by DILIGENT users that invoke a compound service (process). In addition, it will be used in a batch mode (process is scheduled for execution, either by system or by user request). Finally, processes might also be started automatically, due to some events within DILIGENT.

Testing Issues

The Process Execution & Reliability service needs to be tested and evaluated in terms of:

- Reliability: can the different kinds of failures be handled correctly (given the correct and verified specification of single processes). This can be tested by both simulating a grid infrastructure and virtually disconnecting failed nodes that would be needed to execute the compound service, or within a real world testbed.
- Efficiency: this includes the resource consumption of the service itself but also the efficient resource consumption in the grid in terms of the services to be invoked within a process

- Correctness: this includes the execution of a process according to its specification.

Related non-functional requirements

Security issues should be taken into account here, relating to which user is authorized to launch which service. Most importantly, the reliable execution of processes is required. This includes both failures of (basic) services in the grid (semantic failures, failures of service providers) but also failures of the Process Execution & Reliability Service.

An evenly important issue is the reliability of the compound service execution. Due to the nature of distributed environments, and especially in the scope of grid systems, the availability of resources as well as the quality of service they can offer within a single point in time is not predictable. Therefore, it is of high probability that services needed to completely execute a compound service, especially in the case of long running services, might be disconnected during runtime or fail in some other way. It would be undesirable in this case that the execution is aborted or fails. The expected behaviour is rather that execution is continued on other nodes in the grid if possible, or that it is, if reasonable in the special scenario, hold until the needed resource is reconnected or a substitute can be arranged.

7.1.7 Monitor CS

Description and Priority

This use case can be invoked for a service that is currently being executed. Although not specified in the requirement document, online and offline monitoring should be possible. By default, log files (audit logs) should be written to special folders in a predefined format. These files should be available in order to perform statistical measurements and tests. For the online monitoring requirements, the tools providing it should allow the monitoring of an ongoing process (according to the requirements document). In terms of online monitoring, different views are possible: i.) a process-centric view (how many instances of a process currently exist, what is the state of these instances), ii.) an instance-centric view showing details of one selected process instance, and iii.) a service-centric view showing how many processes currently invoke a particular service.

Requirements

The "Monitor CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case.

Numbers

This use case is invoked several times per month (according to the requirement document), and it can be performed in online, i.e. interactive mode (online monitoring) and in offline mode (process analysis, process mining).

Constraints and Assumptions

In this case the requirements document is specifying also as a precondition to the use case execution that the user is authorized to launch the monitor service.

Log files for offline statistical calculations should be available anyway.

In normal operation mode, the user should receive standard monitoring information, such as the percentage of CPU use, estimated remaining time, which task is currently running, output produced. It may be desirable that one can also choose to monitor a process in expert mode, where extra information is prompted to the user. The distinction of normal mode and expert mode is orthogonal to the three views described above.

UML Diagrams

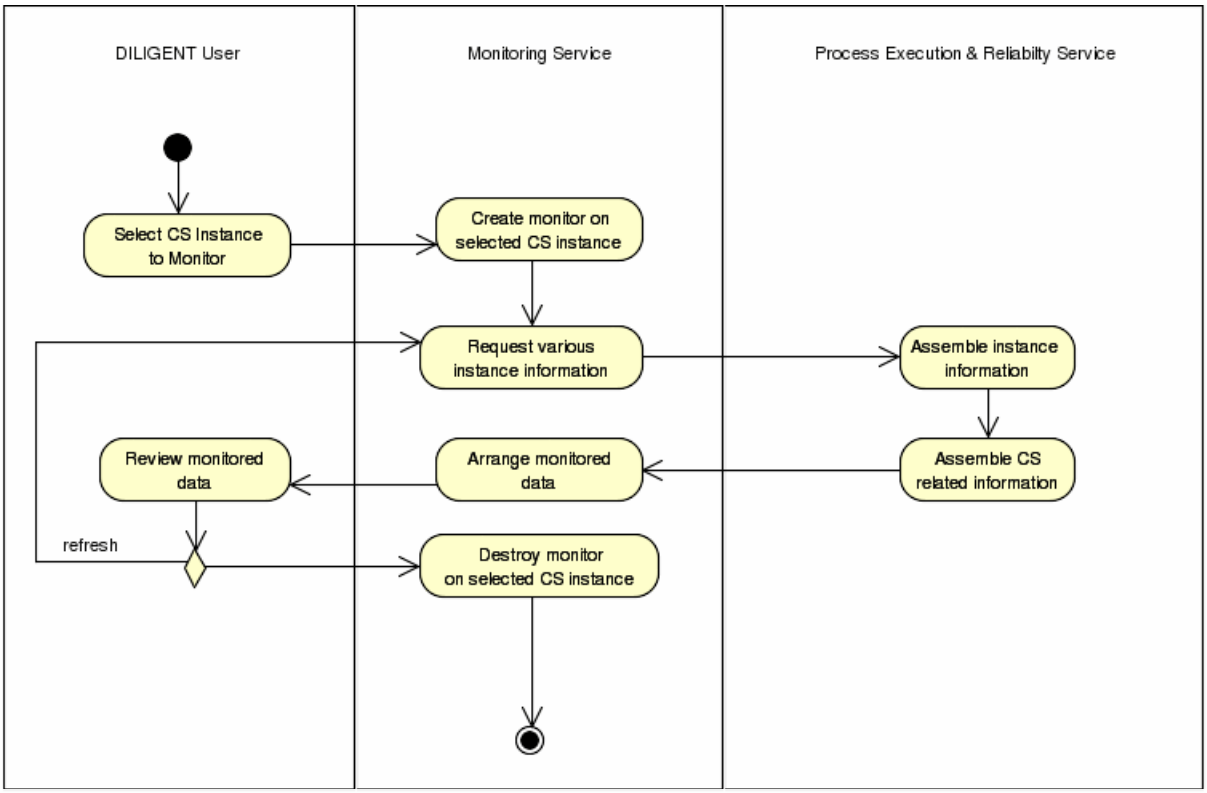

Figure 66: Monitor CS (activity diagram)

Grid Exploitation

See "Run CS".

Use Stories

The simplest justification for this use case would be that online monitoring allows the user to perform simple statistics about the process that is currently running, check the remaining execution time and, most important, monitor warning and error messages. Thus if a problem should occur, the monitoring tool would offer a first hint to what went wrong. For this monitoring tool, especially in the process-centric view and the instance-centric view, the graphical process model (as it has been created by using the Process Design and Verification Service), e.g., O'Grape, can be used.

Testing Issues

See "Run CS"

Related non-functional requirements

See "Run CS"

7.1.8 Abort CS

Description and Priority

This use case can only be invoked for a service that is currently being executed. Although not specified explicitly in the requirement document, by abort here one means implicitly both stopping the current process execution and interrupting the current process execution.

Requirements

The "Abort CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case.

Numbers

This use case is invoked several times per month (according to the requirements document), and it can be performed in online (i.e. interactive) mode only.

Constraints and Assumptions

As mentioned in the "Monitor CS" use case, log files should be written during the process execution, since the requirements document specifies that in case of process aborting, the user should be prompted with a log file describing the processing activities done until the abort. Depending on the process specification, the abort of a process might require certain activities to undo the effects of the aborted process (according to the failure handling strategies defined within the process).

Another functionality mentioned in the ImpECt requirements document for the "Abort CS" use case, the possibility to evaluate intermediate results of a running process, will not be catered for by this use case, but rather by the "Monitor CS" use case. Reasons are that the "Monitor CS" is tailored to that specific functionality, and, more importantly, that any actions taken by a process have to be undone (compensated for) on process abort – thus, a partial execution of a process to gather only intermediate data is not possible.

UML Diagrams

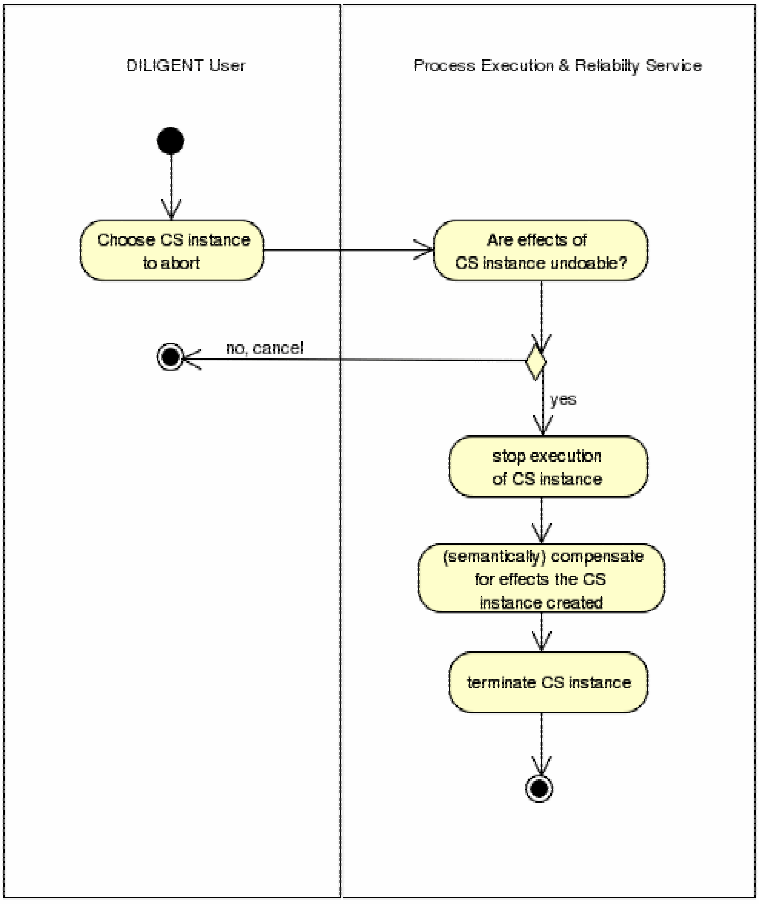

Figure 67: Abort CS (activity diagram)

Use Stories

The abort of a process will be used by the DL users that have invoked a compound service or by a DILIGENT administrator. In both cases, the effects of the process to be aborted have to be considered by the Process Execution and Reliability service (these have to be undone) such that the process abort does not leave any inconsistent states.

Testing Issues

This operation needs to be tested in terms of the correctness of process execution, i.e., no intermediate data / side effects may survive the abort (they have to be undone during process abort).

Related non-functional requirements

See "Run CS"

7.1.9 Remove CS

Description and Priority

As before, the user should be authorized to remove a compound service. The conditions under which the compound services can be removed also have to be specified. The removal of a compound service implies (according to the requirements document) that the log files written during the execution of that process, and/or possible intermediate results have to also be removed.

Requirements

The "Remove CS" use case has been introduced by the ImpECt requirements document as a component part of the "Compound Service Management" use case.

Numbers

This use case is invoked several times per month (according to the requirements document), and it can be performed in both batch and online (i.e. interactive) mode.

Constraints and Assumptions

We propose that the conditions under which the respective compound service can be removed should be specified at creation time, since this is more a configuration issue rather than input data. Removal of a service must not be available when this service is to be used in currently running processes.

Since batch execution is also possible, we propose that automatic removal of a compound service should be possible. One may schedule compound services that are no longer used to be automatically removed.

UML Diagrams

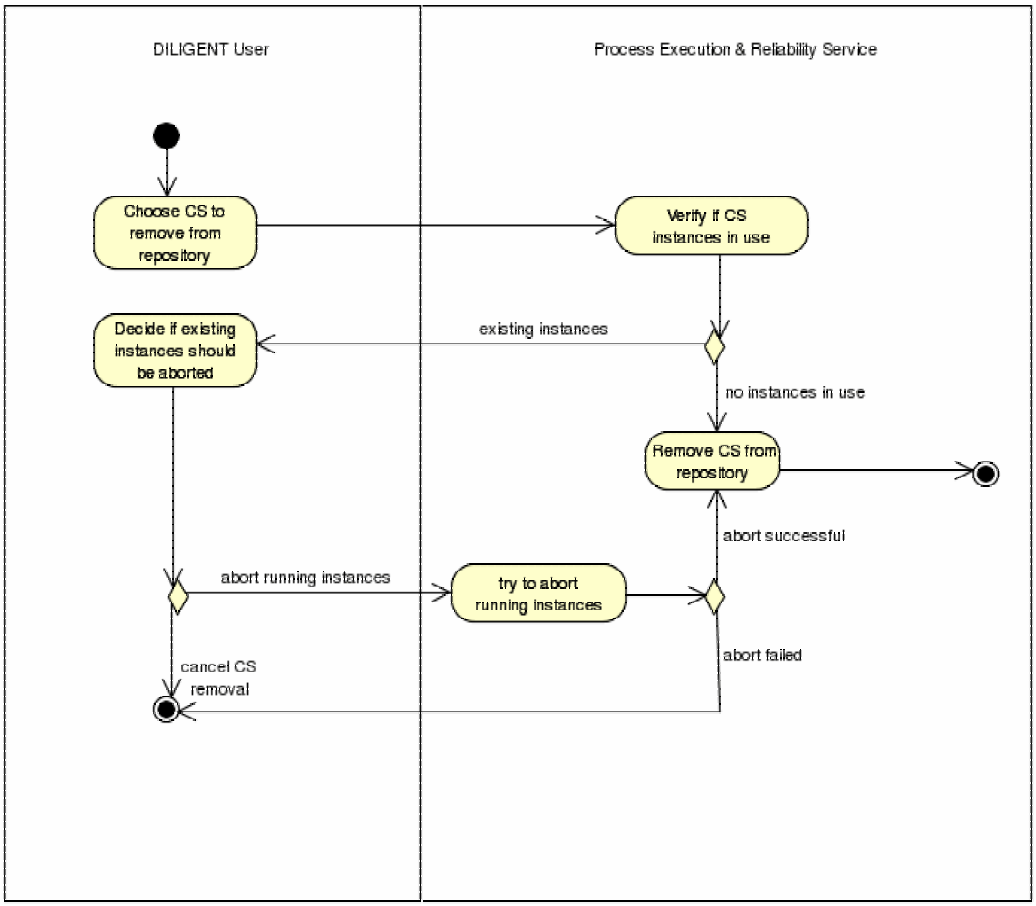

Figure 68: Remove CS (activity diagram)

Grid Exploitation

See "Run CS".

Use Stories

Removal of a service is used by a DILIGENT user (only the user who has defied a compound service shall be able to remove this service) and by DILIGENT administrators. Prior to removal, it has to be checked whether running processes uses this service.

Testing Issues

This operation needs to be tested in terms of the correctness, i.e., no user/administrator must be able to remove a service when it is still in use. The same is true with the removal of batch services.

Related non-functional requirements

See "Run CS".

7.1.10 Optimize

Description and Priority

The Optimize use case is not being derived from any use case scenario. It is the only use case of the Process Optimization Service and its sole purpose is to process a workflow of operations and transform it in order to be carried out in an "optimal" manner.

The role of the Process Optimization Service is to process a workflow comprising service invocation and provide a new modified workflow, which will be carried out in an optimal

way. Optimal is a term that cannot be defined in absolute terms (i.e. faster, cheaper, more accurate etc). The abstract meaning of it is "at a minimal value of a cost estimation function".

Due to its nature, the process optimization service provides only one operation to its clients, the Optimize.

The optimization process is very likely to produce sub-optimal results however this is well acceptable in the current context.

Many of the aspects of the service need to be further discussed and defined within the project since the area of optimizing a workflow of service invocations, in a dynamic extremely complex environment such as the Grid is a very challenging area widely open to research.

The cost estimation function is yet to be defined, however important metrics include processor, network and storage utilization, overall execution time, financial cost, fault tolerance etc.

Optimization will be performed on Direct Acyclic Graphs, which are quite adequate for a large number of typical problems.

The service could be eliminated and the design could assume that end-users or other actors (i.e. users & software) provide optimal workflows of operations to be passed to the actual execution management service. Even "hand" made workflows will exploit some rough optimization performed by end-users.

However, in the dynamic and complex environment of the Grid, this could potentially lead to severe mis-utilisation of resources.

Taking into account that, the overall service priority is medium, since the system can carry out its work if it is not present at all, even with significant cost.

Requirements

There is not end-user requirement that specifically instructs the inclusion of this functionality. However the requested capability of composing workflows implies that these are to be somehow executed in an efficient manner. So the Optimize Process functionality is mostly derived by the experience of the system designers on similar cases.

Numbers

Currently there is no estimation on the "volumetrics" of the Optimize functionality. Although it is a service that can be used by various DILIGENT components, it is expected that its main usage will be to optimize workflows that are being manually created by end-users to achieve some desired functionality. Examples are: a batch job on a Digital Library Collection, or a batch feature extraction etc. Experience on workflows produced by other systems shows that they could range from a simple 2-node graph to a 1000-node complex diagram. However in our case diagrams are expected to be in the neighbourhood of 10 to 100 nodes.

Constraints and Assumptions

Invocation of the service is directly triggered by the process execution and reliability service prior to executing a workflow. The invocation of the service is optional, i.e. the caller might decide not to pass a workflow through optimization for various reasons, i.e. the user has forced "no optimization" or the workflow has already been optimized etc

Service related preconditions include:

- Configuration (Cost estimation function configuration, operators etc)
- Information (i.e. resource existence, capabilities and utilization)
- Workflow (Cost X1)
- Workflow Cache

Service Post-condition include:

- Configuration (unmodified)
- Information (unmodified)
- Workflow (Cost X2<=X1)
- Workflow Cache (contains new workflow if no previously optimized, LFU?)

These exceptions are being internally caught by the service and cause it to stop execution

- Any of the pre-conditions not present with the exception of the cache
- Errors in the supplied workflow
- Repeated failure to perform optimization

UML Diagrams

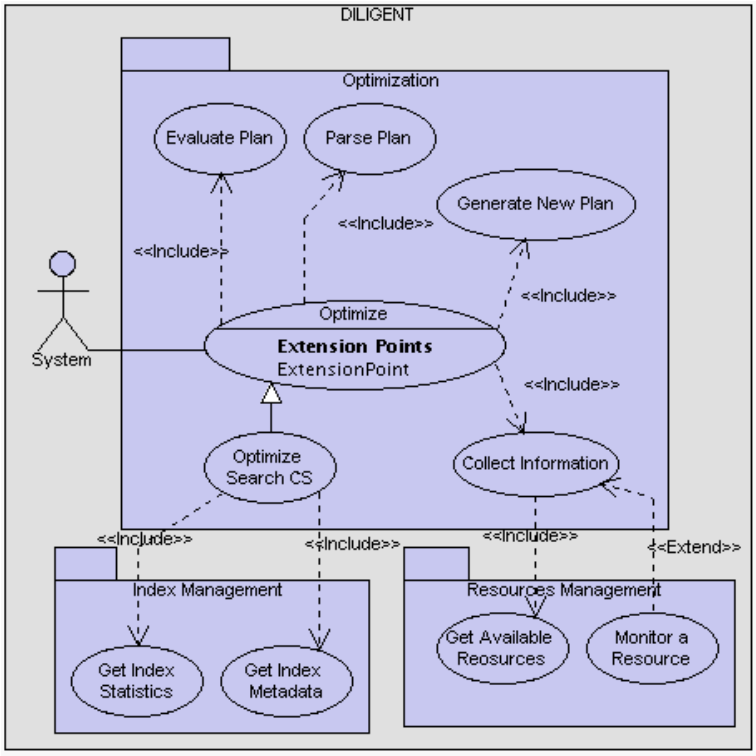

Figure 69: Optimize Process Use Case

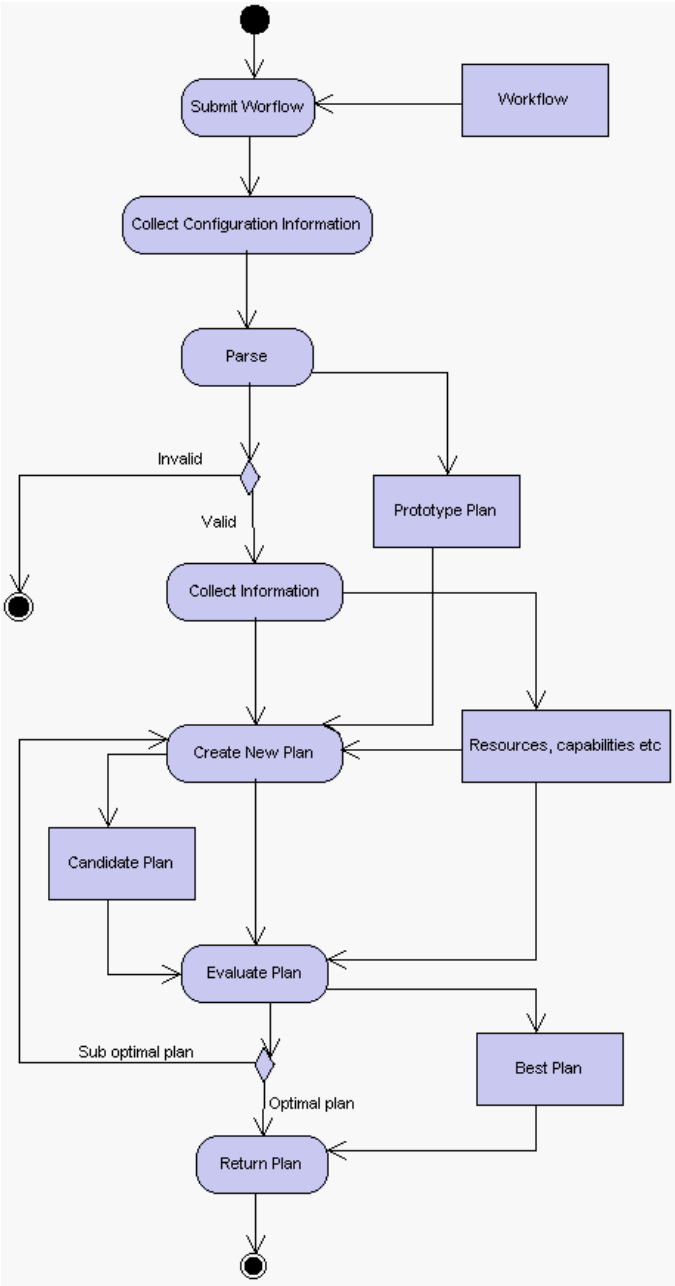

Figure 70: Optimize Process (activity diagram)

Grid Exploitation

The optimize functionality can exploit Grid substrate in various ways, however not all of them might be applied in order to achieve a desired outcome:

- Collects information on resources from Information Services.
- Exploits distributed processing for minimizing optimization time for large workflows of operations.
- Produces outcome that is compatible with underlying Grid components and services (e.g. DAG etc)

System Integration

The use case presented here belongs to the Process Optimization Service.

Use Stories

Other diligent services as well as end users might be potential beneficiaries of the Process Optimization functionality. A typical example of is usage would be:

- User submits a flow of operations:
  - Get data chunk **A**.
  - Get data chunk **B**.
  - Locate operation **OppX**.
  - Get data chunk **C**.
  - Locate operation **OppZ**.
  - Process data **A** and **B** through **OppX** and produce data chunk **D**.
  - Locate operation **OppY**.
  - Process data chunks **C** and **D** through **OppY**.
  - Return the results to node **NdU**.
- The system collects information on all operators and that are invoked in this sequence and the various alternatives as well as constraints.
  - Examples of operators are:
    - Named operators (e.g. OppX, OppY etc)
    - Implied operators (e.g. MoveData)
  - Examples of resources are
    - Nodes (e.g. NdU, NdG etc)
    - Pathways (e.g. NdU-to-NdG etc)
    - Data chunks (e.g. A, B, C etc)
  - Examples Information on resources
    - Location and size of data chunks
    - Processing power of computing nodes
    - Throughput & Response time of pathways
    - Accessibility (e.g. access rights)
  - Other constraints
    - QoS related constraints
    - Accounting related constraints
- Decides what is the best way to execute the operation workflow in order to minimize cost and safeguard the constraints. Alternatives could be:
  - Move all data and process code to node NdU and do all work locally.
  - Process the large initial data blocs A and B locally (after an initial move of A close to B) by moving the code close to them, and then move the small expected result C to NdU along with necessary code and remaining data to complete the operation locally.
  - Move code closer to data and only the final result to NdU (e.g. NdU is very far from other nodes and the result data are very small)

Testing Issues

There are two aspects of this functionality that need severe testing:

- The validity of the output workflow with regards to the original one, since semantics should always be the same.
- The performance of the optimized workflow when compared to the original one.

Both of them can be checked in one step via "brute" testing: Submitting workflows on test data twice, before and after optimization and gathering and evaluating the produced results. Cost and match can be both evaluated at the same time.

However more formal approaches on testing semantic validity can also be applied.

Related non-functional requirements

Optimization service should pose minimal resource usage, so that the gain it offers is not overtaken by the cost to run it.

Safeguarding the semantics of the process being optimized is a key priority of the service.

Output execution workflow should always be at least as good as the input one, in terms of predefined cost function.

Security layer (authentication/authorization) should be capable of supporting the creation of a-priori valid workflows, with regards to user access rights.

Service should be interoperable with underlying grid services and components that expose overlapping facilities.

8 APPLICATION SPECIFIC

8.1 Introduction

This functional area aims to cover the remaining functionalities that are closer to the users and in particular to those requirements that are more specific in nature, i.e. requirements related with specific applications instead that with generic functions that could be used to support and implement other ones. For this reason we have organized this area of functional specifications in two sub areas:

- **Portal Functional Specification** (see Section 8.2);
- **Report Management** (see Section 8.3);

The former area covers various aspects related with the DILIGENT graphical user interface, i.e. the Portal. These aspects range from the configuration of the portal engine to the login and search functionalities. Moreover the presentation layer related with the workshops, courses and exhibition catalogue management is modelled.

The latter area presents the functionality related with the management of reports, i.e. virtual documents whose content is identified and produced on demand.

8.2 Portal Functional Specification

The following sections describe a set of functionalities which are relates to interfacing the user to the core functionalities provided by the DILIGENT platform. From a technical perspective every user interface component is considered to be a portlet or a subsection of one. Thus in the remaining document the terms "User Interface" (abbreviated to "UI") and "Portlet" will be used interchangeably.

8.2.1 Login

Description and Priority

This function allows users to log in on the Diligent system. Depending on the user rights, the login allows to access the Diligent/DL's functions and resources.

Logging in starts a session for this particular user.

Requirements

User must have an account before being able to log in.

Numbers

This functionality is likely to be used very frequently (it is the first action any user has to take before being able to access Diligent Resources).

UML Diagrams

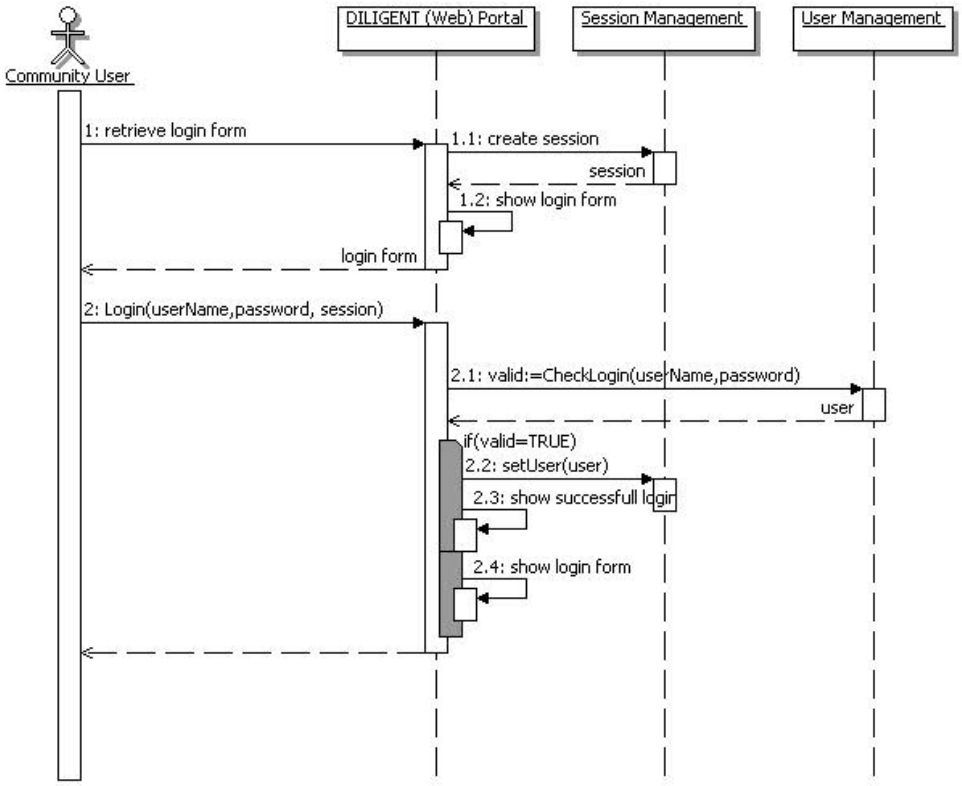

Figure 71: Login (sequence diagram)

Grid Exploitation

The purpose of the Portal is to hide the Grid infrastructure from the user. It is unknown to the user, where the Portal retrieves its information.

Each Portal comes with its own Session Management, so it depends on the Portal whether or not Grid infrastructure is used for the Session Management.

Use Stories

The user tries to log in via the (Web-)Interface.

8.2.2 Logout

Description and Priority

The logout is necessary to allow users to 'correctly' (e.g. checking for not saved data) exit all the Diligent activities

Logout also terminates the session participated in.

Requirements

User must be being logged in, before being able to log out.

Numbers

This functionality is likely to be used very frequently (it is the last action any user has to take before leaving the Diligent System)

Constraints and Assumptions

It is assumed that all communication between User and Portal is executed in the same session.

UML Diagrams

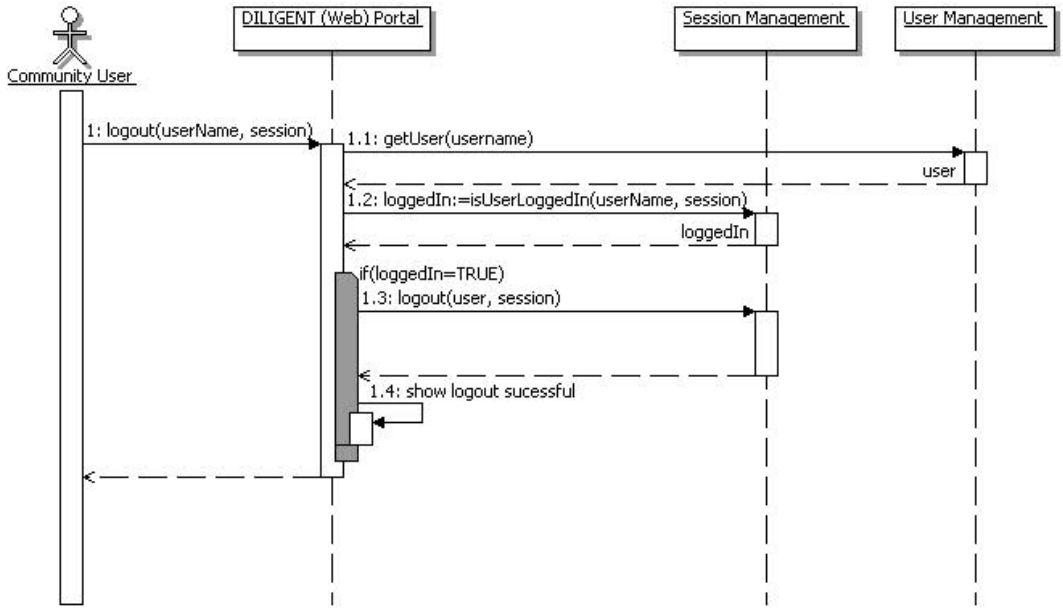

Figure 72: Logout (sequence diagram)

Grid Exploitation

As in the "Login" use case, Grid specific functionality might or might not be encapsulated in the Session Management

Use Stories

The user logs out via the (Web-)Interface

8.2.3 Portal Storage Access

Description

Portal storage access addresses the need for storing and retrieving information related to the Portal Engine and to the hosted portal implementations.

The following sub-functionalities are provided:

- Store Portal Data Object
- Retrieve Portal Data Object
- Delete Portal Data Object
- Browse Portal Data Objects

These functionalities will not be analyzed because of two important observations:

- They relate to internal low-level functionality to be provided to the DILIGENT by existing implementations of Portal Engines.
- They are abstract representations of various different approaches for storing portal configuration and portlet data met in various different portal engines.

Making any further assumptions at this stage would raise risks for the future.

Priority

This use case has a high priority, because they are the primary means to keep persistent portal and portlet behaviour.

Requirements

The inclusion of the functionality is based on a decision taken from evaluation of the technical status regarding the portal engine and portlet technologies.

The portal engine is most likely to support a native storage management facility, which might not be grid aware. Grid awareness at the portal level, although welcomed, is not mandatory.

Numbers

We expect to be indirectly used by all Digital Library users when configuring or accessing the portal.

UML Diagrams

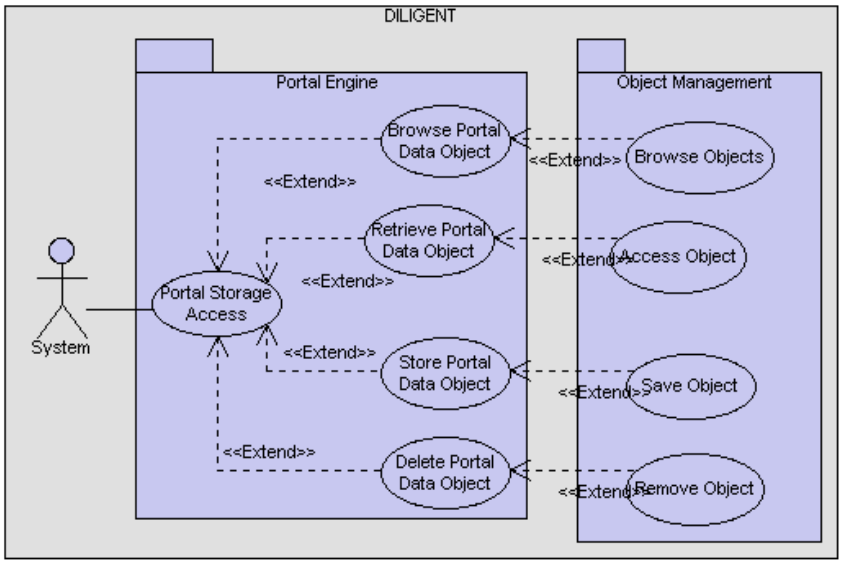

Figure 73: Portal Storage Access (use case diagram)

Grid Exploitation

Portal Storage will not directly exploit any grid facilities unless provided by the Portal Engine and lower level of storage management.

System Integration

Portal storage is being accessed by all portal related functionality.

The functionality could possibly integrate with Object Management (as depicted on the UML diagram) if such a facility is to be easily supplied by the portal engine.

Alternatively, direct access to Grid storage could be utilized, if the engine directly supports it.

8.2.4 Submit User Credentials

Description

This functionality encapsulates the user's login procedure. The user must fill in the username and the password, or supply the credentials in an alternative way (e.g. certificate). The system has to check if the credentials are valid and then to find, retrieve and load personal data for the home page.

If the user is not registered yet, he must first register and then try to login. In order to do that a facility to request registration is provided.

Alternatively a "request for access grant" form should be filled and the procedure should proceed as dictated by the system relevant policy.

Priority

This use case has a priority of level "mandatory".

Requirements

The "Submit user credentials" functionality is stated in user requirements documents, nevertheless it is an obvious requirement in order to exploit the security and accounting mechanisms of the underlying DILIGENT platform.

Numbers

Almost each and every user that uses the DILIGENT platform capabilities through the portal will consume the aforementioned functionality. Exception will be made on anonymous access, if this is being supported by a DL.

Constraints and Assumptions

The functionality will be provided by a dedicated portlet that will exploit the security and hosting facilities of the Portal Engine. The Portal Engine should provide standards-based facilities for portlet inclusion.

UML Diagrams

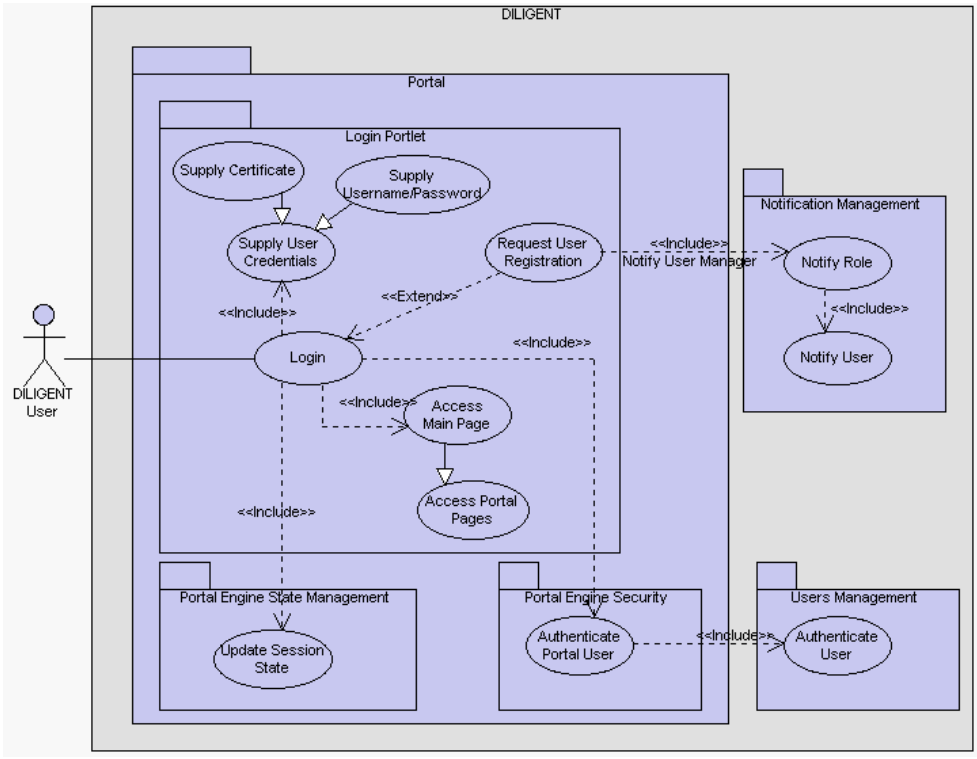

Figure 74: Submit User Credentials (use case diagram)

Grid Exploitation

This functionality has no needs for grid technology other than the indirect consumption of relevant services through the underlying Portal Engine and DILIGENT platform.

System Integration

Login is part of the application layer group of services. It mainly uses the portal engine security mechanism, which accesses the DILIGENT and GRID security layer group of services, based on well-known relevant grid technologies and paradigms.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.5 Request User Registration

Description

This functionality allows users to ask for registering a digital library.

The action to be expected by the system could be one of the following:

- Reject users request (no open registration allowed)
- Automatically accept the user request and register the user (open system that allows registration for enabling personalization and user experience enhancement)
- Notify the DL Administrator for taking further action:
  - Accept the user request and register the user
  - Reject the user request

Implementing any of these policies is out of scope for the DILIGENT.

Priority

This use case has a priority of level "optional", as it can be achieved by alternative means (e.g. e-mail)

Requirements

Requires the existence of a portal engine and sufficient portlets to render the supplied content and provide the desired functionality.

Mechanism and policies for creating users automatically, semi-automatically or manually should be provided by the DILIGENT platform.

Numbers

The usual case with this functionality will be mostly invoked once per user.

Restricted access systems might well omit this functionality.

Constraints and Assumptions

The Portal Engine should provide standards-based facilities for portlet inclusion to render the page. Rendering of a page depends on configuration, personalization, portlets behaviour and content.

UML Diagrams

The functionality is depicted in the Figure 74: Submit User Credentials (use case diagram).

Grid Exploitation

This functionality has no needs for grid technology other than the indirect consumption of relevant services through the underlying Portal Engine and DILIGENT platform.

System Integration

Login is part of the application layer group of services. It mainly uses the portal engine security mechanism, which accesses the DILIGENT and GRID security layer group of services, based on well-known relevant grid technologies and paradigms.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.6 Access Portal Pages

Description

This functionality refers to user navigation through the portal. The functionality should be present for both visitors and authenticated DL users.

Rendering a page depends on

- system configuration
- personalization of portlets
- page configuration
- portlet behaviour
- content

Functionality for non-registered / logged on members might be somewhat limited, depending on a per DILIGENT instance policy.

Priority

This use case has a priority of level "mandatory".

Requirements

Requires the existence of a portal engine and sufficient portlets to render the supplied content and provide the desired functionality.

Numbers

Each and every user that uses the DILIGENT platform capabilities through the portal will consume the aforementioned functionality multiple times per interaction.

Constraints and Assumptions

The Portal Engine should provide standards-based facilities for portlet inclusion to render the page. Rendering of a page depends on configuration, personalization, portlets behaviour and content.

UML Diagrams

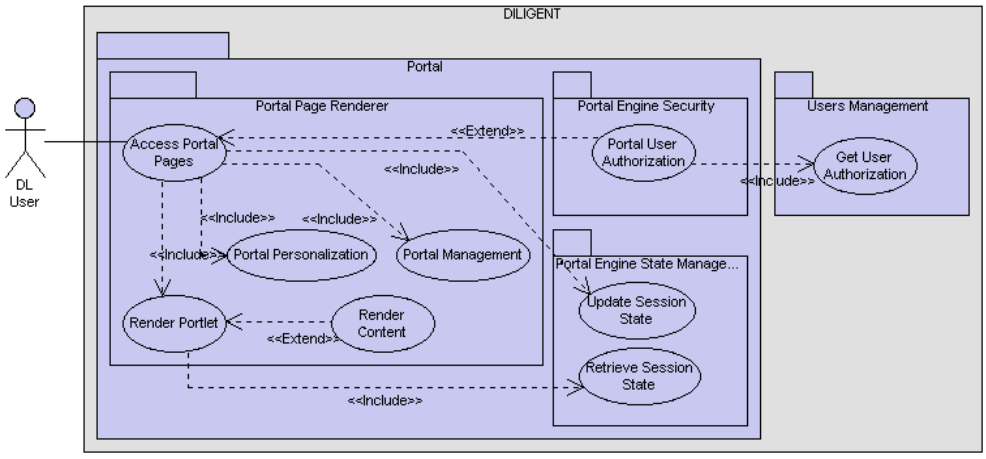

Figure 75: Access Portal Pages (use case diagram)

Grid Exploitation

This functionality has no needs for grid technology other than the indirect consumption of relevant services through the underlying Portal Engine and DILIGENT platform.

System Integration

Accessing and rendering portal pages is part of the application layer group of services. It mainly uses the portal engine rendering mechanism, based on well-known relevant grid technologies and paradigms.

Additionally it indirectly integrates with the security mechanism provided by user management and the storage mechanisms, if supported by the engine.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.7 Manage Digital Objects

Description

This set of functionalities allows a user to:

Upload a Digital Object (DO) to the system for use under various cases:

- Include to a collection.
- Use for similarity search.
- Use as parameter to a process workflow.
- Delete a DO

The scope where the object belongs to is implicitly selected, depending on the context. However some cases should allow scope could be selected manually (e.g. which Collection, or what for upload the DO etc)

Priority

This use case has a priority of level "mandatory" because it is essential for various cases.

Requirements

This functionality is implicitly derived from user requirements.

Numbers

Almost each and every user that uses the DILIGENT platform search capabilities through the portal will consume the aforementioned functionality.

Constraints and Assumptions

The functionality will be provided by a dedicated portlet that will exploit the security and hosting facilities of the Portal Engine. Uploading of Digital objects will be achieved through standard protocols, HTTP / HTTPS being the two most preferable, FTP being a third candidate etc.

UML Diagrams

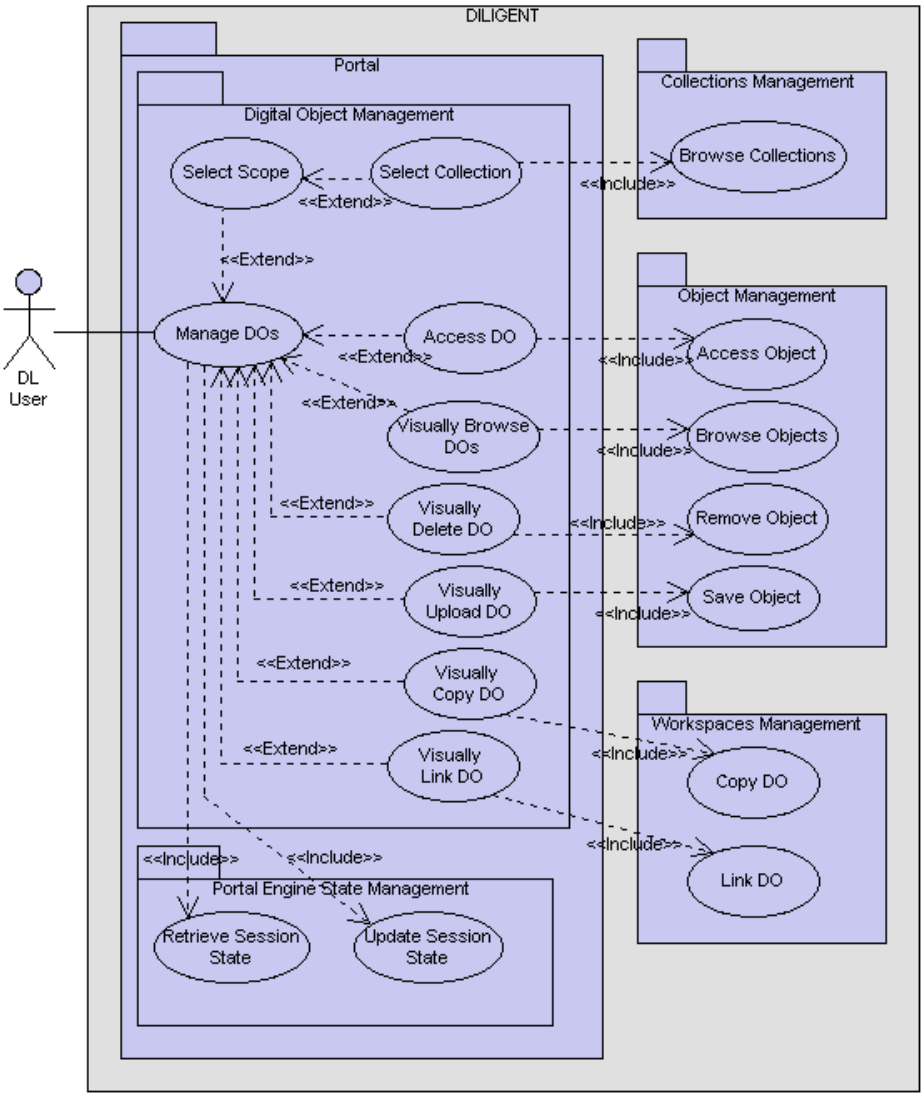

Figure 76: Digital Object Management (use case diagram)

Grid Exploitation

Portal components do not exploit grid by any other means than the underlying DILIGENT services. Exception might be introduced for replicated portal configuration storage.

System Integration

Digital Object Management portlet is part of the application layer group of services and components and is being hosted as a component of the Portal Engine. Although it might not be the main mean for populating a Digital Library, it is necessary for various operations such as creating Digital Collections, performing similarity search etc. It should provide the opportunity for integration with various other portlets and consume the platform functionality for the DO management operations.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.8 Submit Search

Description

The "Submit Search" functionality has to do with the main exploitation of the DILIGENT search service over the portal's User Interface (UI). Its parts are: submit new search criteria, add / remove criteria, produce the query (for search), call search service and browsing the results. Domain specific application logic is added to search submission in order to hide away the complexity of search query language. It uses personalization service for all type of searching.

In case of similarity-based search, the portlet should allow the uploading or selection of content through a Graphical UI (GUI) and typical transfer protocols (FTP, HTTP etc).

The functionality will be provided by a customizable search portlet that will support all the types of search provided by the underlying search engine.

Priority

This use case has a high priority, because it deals with the exploitation of the search service, which is one of the main objectives of DILIGENT.

Requirements

All user requirements mention search as main functionality, although types of search to be supported differ quite a lot. However the functionality provided should allow to:

- Interact with all types of Search Engine services;
- Decouple from complexity of the underlying search language.

Numbers

Almost each and every user that uses the DILIGENT platform search capabilities through the portal will consume the aforementioned functionality.

Constraints and Assumptions

The functionality will be provided by a dedicated portlet that will exploit the security and hosting facilities of the Portal Engine. The Portal Engine should provide standards-based facilities for portlet inclusion.

UML Diagrams

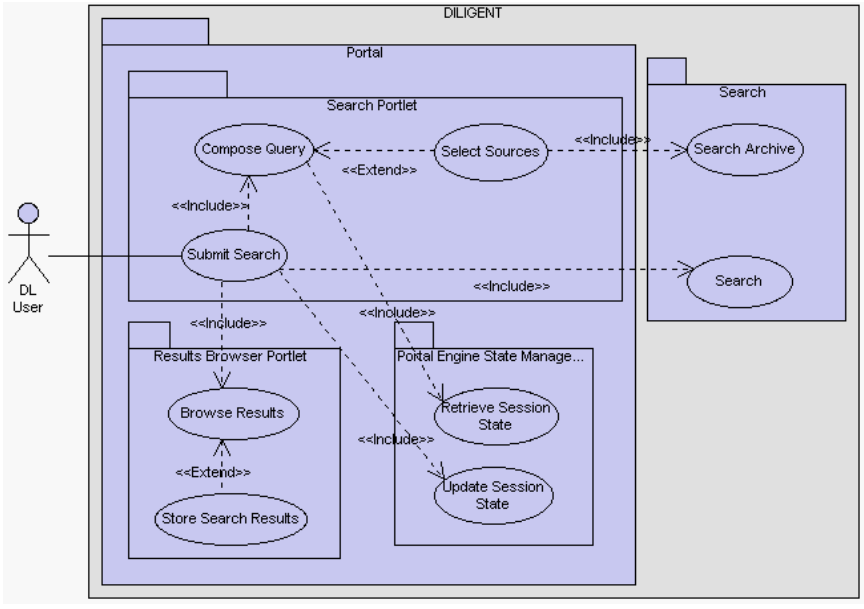

Figure 77: Search portlet (use case diagram)

Grid Exploitation

Portal components do not exploit grid by any other means than the underlying DILIGENT services. Exception might be introduced for replicated portal configuration storage.

System Integration

Search portlet is part of the application layer group of services and components and is being hosted as a component of the Portal Engine. It mainly consumes the DILIGENT Search Engine functionality. However it indirectly accesses storage management functionality, while it is tightly coupled with the results browsing portlet.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.9 Browse Results

Description

The Results browsing functionality reflects the need for presenting search results to the DL user in a user-friendly manner.

The functionality should provide the well-known facilities of movement within the results and it is expected that asynchronous operation will be provided, i.e. gradually present results as they become available, it he submitted query allows such an operation.

The results browsing functionality should allow rich customization for adopting a suitable presentation form to fit in the general portal templates. Additionally there should be some degree of customization with regards to the amount of info presented to the DL User.

Results browsing should support the "drill-in" use case so that the user can further refine or redirect a query.

The optional "feedback" or "advanced personalization" facilities should be attached to this functionality in future versions of the DILIGENT platform.

Priority

This use case has a high priority, as it is the main medium to use the search functionality outcome.

Requirements

The functionality is embedded in all search-related use-cases.

Numbers

Almost each and every user that uses the DILIGENT platform search capabilities through the portal will consume the aforementioned functionality.

Constraints and Assumptions

We assume that this functionality will be a specific visualization portlet for standard DILIGENT partial resultsets. However this is to be discussed and defined.

The functionality is application-domain specific.

In order to further proceed needs information from search portlet components.

UML Diagrams

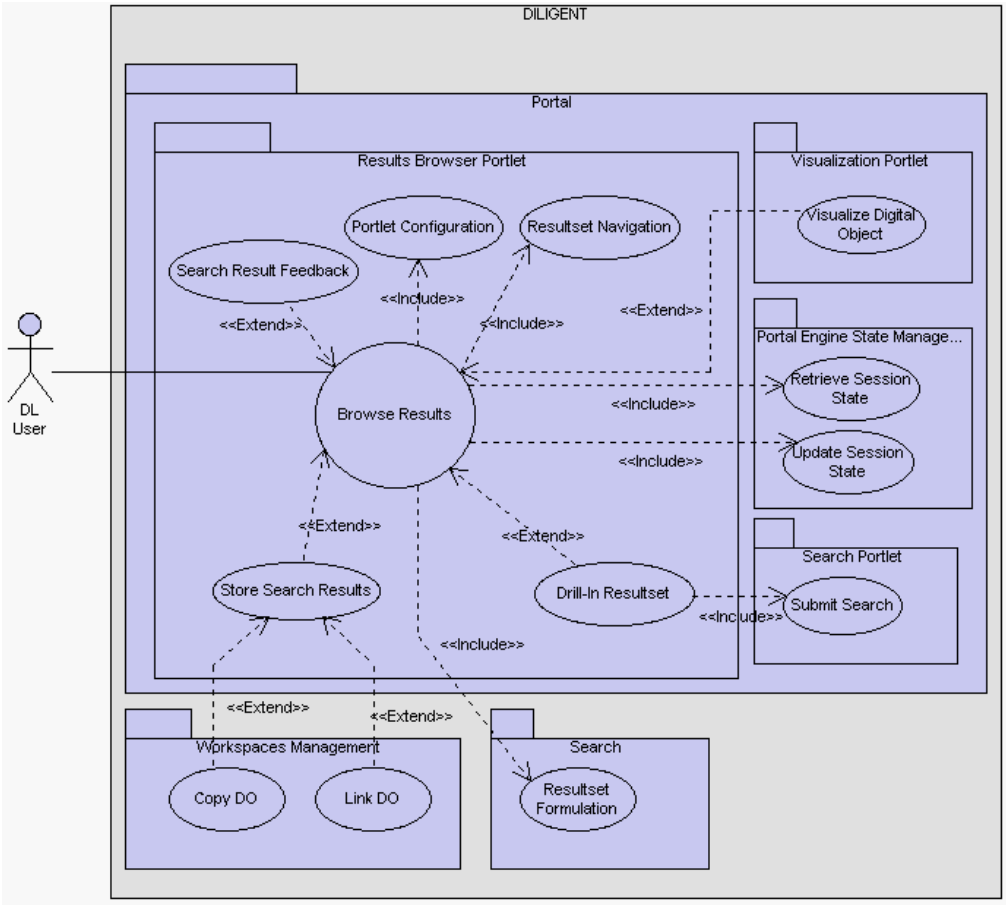

Figure 78: Browse portlet (use case diagram)

Grid Exploitation

This functionality has no significant needs for grid exploitation other than the implied from the dependent services.

System Integration

Browsing functionality is part of the end user applications. It mainly uses the Search engine and is hosted within the Portal Engine. However some functionalities deal with backend storage management, while it integrates with the visualization and search portlets.

8.2.10 Digital Object Visualization

Description

This functionality has to do with visualization of "documents" stored in a Virtual Digital Library. Selection of object is done through browsing a search resultset, or any other mean of locating an object (e.g. an exhibition etc).

The "renderer" will depend on standard formats to present text, image, video, sound and well-known multimedia formats. Client capabilities will be exploited to view the results as appropriate.

Visualization is actually an intermediate between storage, optional visualization services and the browser, however we do not expect to add much functionality as a layer.

The optional "feedback" or "advanced personalization" facilities should be attached to this functionality in future versions of the DILIGENT platform.

Priority

This use case has a high priority.

Requirements

Referred by all user requirements as one of the most basic functionalities.

Depends on client capabilities for displaying standard digital formats of data and services to store or produce these "viewable" formats.

Numbers

Almost each and every user that uses the DILIGENT platform capabilities through the portal will consume the aforementioned functionality.

UML Diagrams

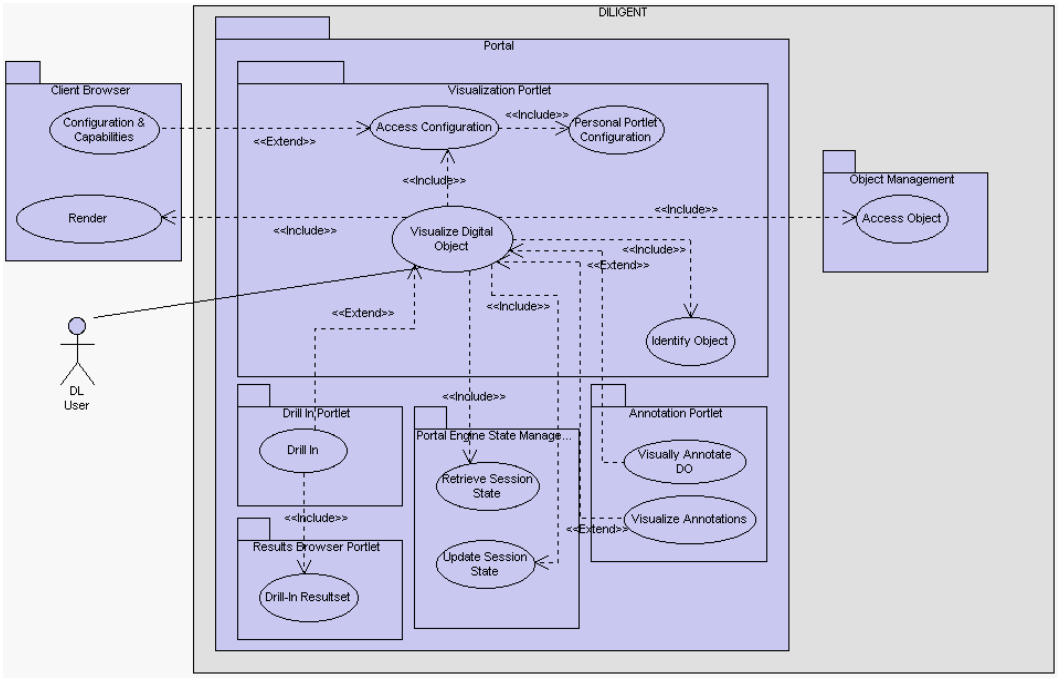

Figure 79: Object Visualization portlet (use case diagram)

Grid Exploitation

There are no significant needs for grid technology.

System Integration

The portlet integrates the Visualization functionality, which is part of application layer, with the browse resultsets and the annotation portlets and exploits the use of Object Management services.

8.2.11 Digital Object Annotation

Description

This functionality refers to ability of the user to annotate the digital objects through a graphical user interface.

Priority

This use case has a priority of level "optional".

Requirements

Requires the existence of a portal engine and sufficient portlets to render the supplied content and provide the desired functionality.

Additionally it requires the existence of the corresponding annotation service.

Numbers

Every user that uses the DILIGENT platform capabilities through the portal might consume the aforementioned functionality multiple times per interaction. However it is expected that actual usage will be quite low for most objects / collections.

UML Diagrams

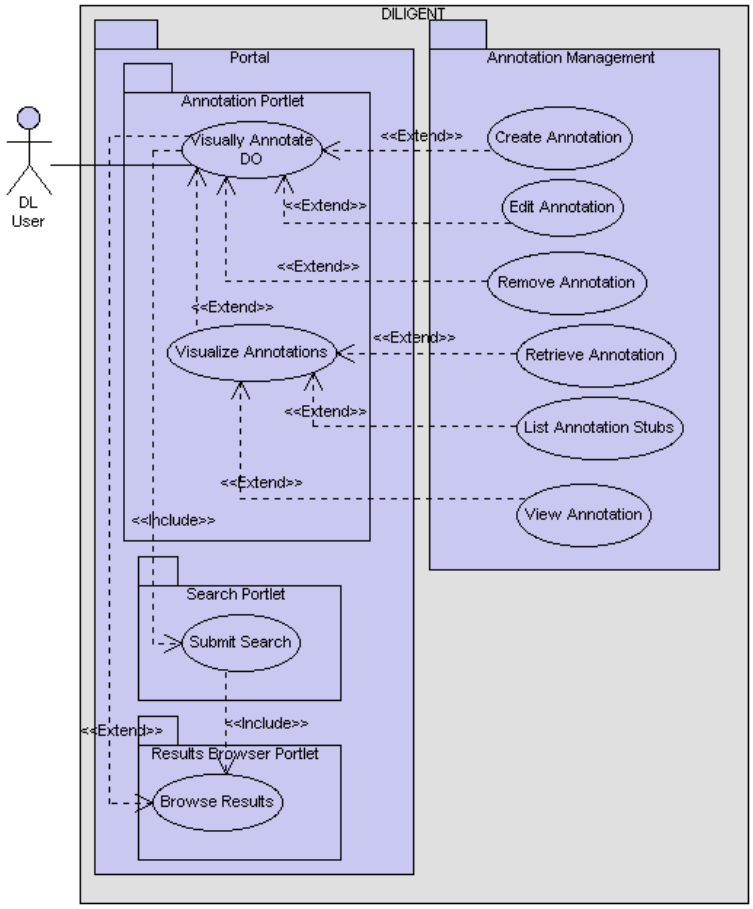

Figure 80: Annotation Portlet (use case diagram)

Grid Exploitation

This functionality has no needs for grid technology other than the indirect consumption of relevant services through the underlying Portal Engine and DILIGENT platform.

System Integration

This is part of the application layer group of services. It mainly uses the portal engine rendering mechanism, based on well-known relevant grid technologies and paradigms.

Additionally it indirectly integrates with the security mechanism provided by user management and the storage mechanisms, if supported by the engine.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.12 Drill In Search Results

Description

The Drill-In functionality is very closely connected to the "search" and the "browse" use cases. Its purpose is to give the DL User the ability to either focus or redefine a search, based on the items that are being currently presented and the original query.

Drill-In is a very application-domain specific functionality, because there is no obvious generic rule on the roadmap to be followed. During the design and implementation phases actual paradigms will be applied.

The inclusion of this functionality dictates that certain state and configuration information must be persisted among user-service interaction stages (i.e. session)

Priority

This use case has a low priority, as the user can achieve similar results by manually constructing a new query.

Requirements

The requirements for this functionality are being stated in the ImpEct portal scenario. Additionally such facilities can be found in various search services with a varying degree of complexity.

Constraints and Assumptions

In order for such facilities to be available, state information must be preserved either in session object or (even better) as resultset metadata.

UML Diagrams

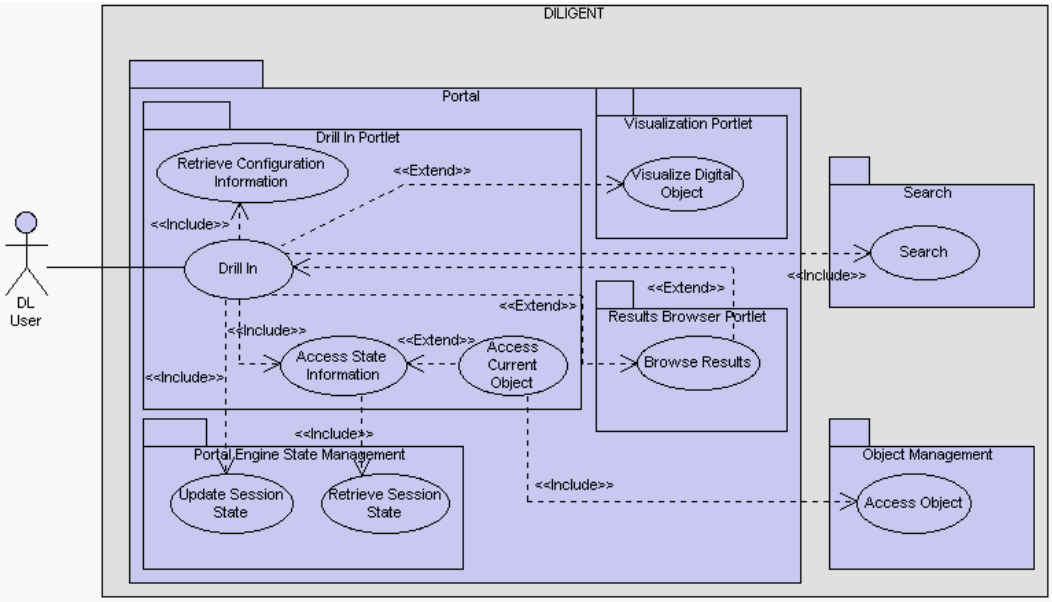

Figure 81: Drill In portlet (use case diagram)

Grid Exploitation

This functionality has no significant needs for grid technology.

System Integration

This functionality belongs to the application domain specific set of services.

It is hosted within the Portal Engine.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.13 Propose Resources & Services for Addition

Description

This functionality is a "helper" functionality that serves as a "messaging" system between DL users and DL Managers / DL Admins, for improving a DL towards covering all DL user needs.

Through this facility DL Users will be able to submit their requests for new collections, services or other types of resources to be included in a VDL.

DL Admins and DL Managers will be supplied with the appropriate means to browse and manage user requirements. However this will be restricted to the functionality of managing the proposals log records.

Priority

This use case has a low priority, as its functionality can be achieved by other means as well.

Requirements

Both end user scenarios imply the existence of such a capability, for a different purpose. However alignment shows that a single functionality group is required.

Numbers

There is no clear estimation about the volumetrics of the aforementioned functionality, however we expect to be used by small portions of authorized Digital Library users.

UML Diagrams

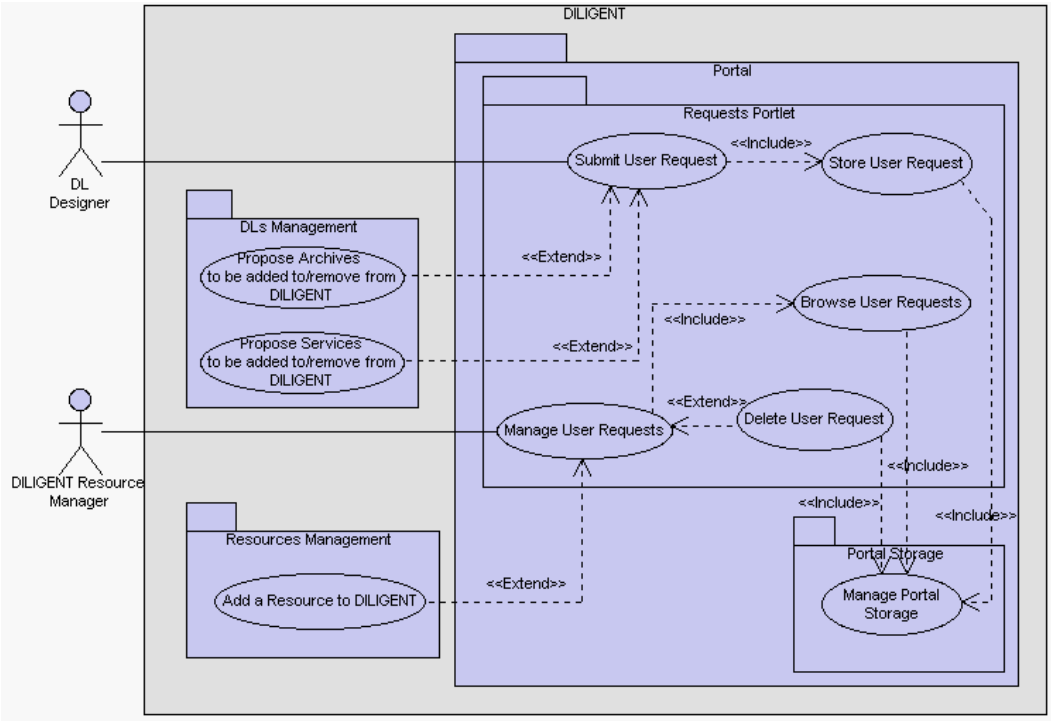

Figure 82: Requests portlet (use case diagram)

Grid Exploitation

Visualization has no significant needs for grid technology.

System Integration

Visualization is part of the application service layer. It is hosted within the portal engine.

Related non-functional requirements

The functionality should best be provided by components based on well-known standards that would allow migrating from platform to platform and version to version.

8.2.14 Course Management and Participation

Description

Course management and participation functionality supports creation and maintenance of educational courses within the thematic area of a DILIGENT-hosted digital library.

The functionality has two faces:

- the managerial one, which allows users to create courses and define the roles of various users, and
- the DL user face, which allows users to participate a course as:
    - Students
    - Teachers

The two types of participants supported have slightly different rights when accessing course material.

The system provides basic tools for collaboration, such as uploading and downloading of educational material (lectures) and a discussion facility for making commentary or interrogative posts.

The entire functionality is based on existing DILIGENT services (search, collections, annotations, storage, security etc.) and is made accessible to the user through a dedicated set of portlets.

The system supports browsing of courses and a sub-functionality for requesting participation. If a course is not being marked as "open" then the course manager (DL Manager or an authorized DL User) must register the user in the course.

Priority

This functionality has a low priority since it is not essential for the operation of a Digital Library.

Requirements

The functionality described here is a direct consequence of the ARTE user scenario.

Numbers

There is no clear estimation about the volumetrics of the aforementioned functionality, however we expect to be used by very small portions of authorized Digital Library users.

UML Diagrams

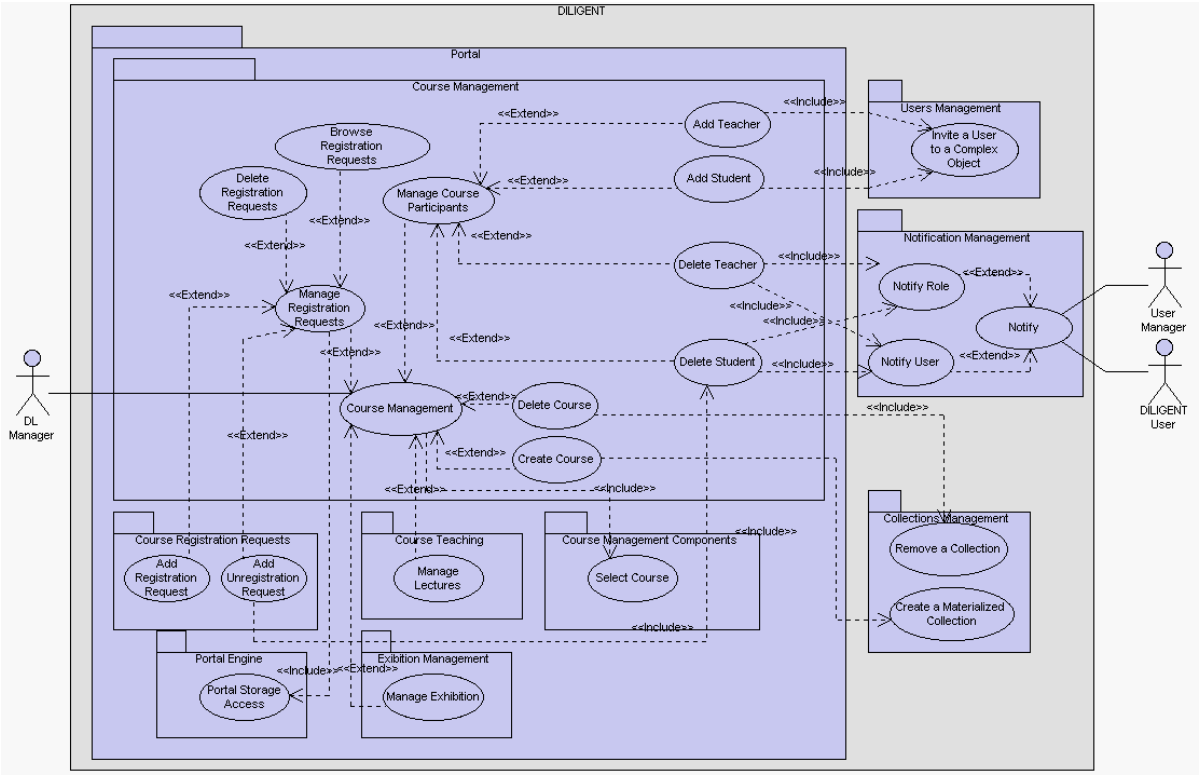

Figure 83: Course Management portlet (use case diagram)

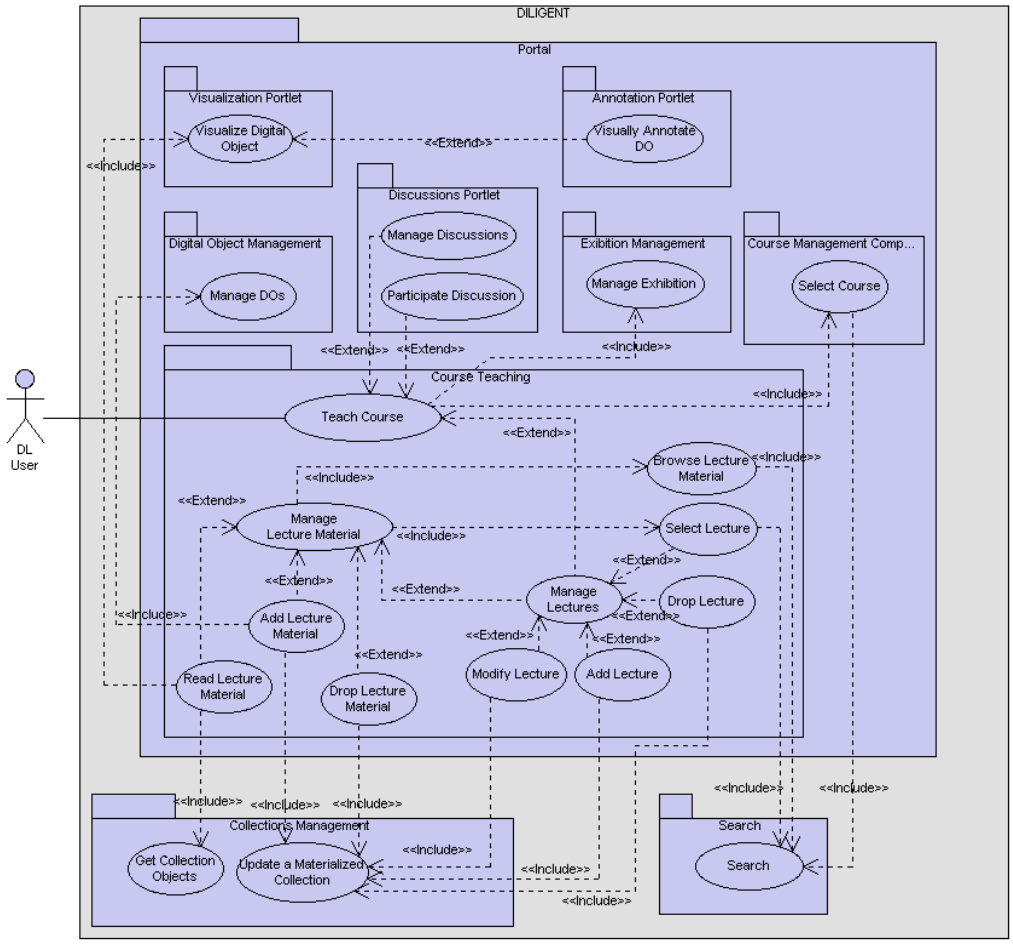

Figure 84: Course Teaching Portlet (use case diagram)

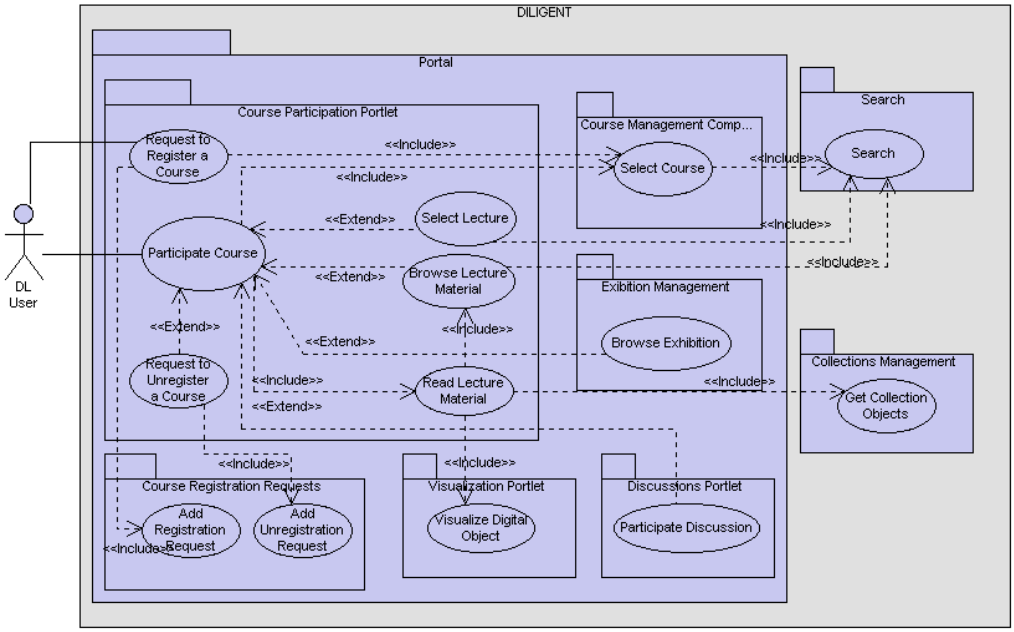

Figure 85: Course Participation Portlet (use ase diagram)

Grid Exploitation

There is no need for grid exploitation specific to this functionality.

System Integration

This functionality is part of the application layer of services. However it exploits the functionality various other services and facilities such as:

- Search
- Annotation
- Discussions
- Storage
- Visualization

8.2.15 Workshop Management & Participation

Description

Workshop management and participation functionality supports creation and maintenance of workshops within the thematic area of a DILIGENT-hosted digital library.

The functionality is basically similar to the one provided by the "course management", differences are related with the users involved in the scenario and with the semantic of the actions, e.g. even if there are functions for inviting users, invite a student in a course is different than invite a speaker for a workshop.

Priority

This functionality has a low priority since it is not essential for the operation of a Digital Library.

Requirements

The functionality described here is a direct consequence of the ARTE user scenario.

Numbers

There is no clear estimation about the volumetrics of this functionality, however we expect to be used by small portions of a registered Digital Library users.

UML Diagrams

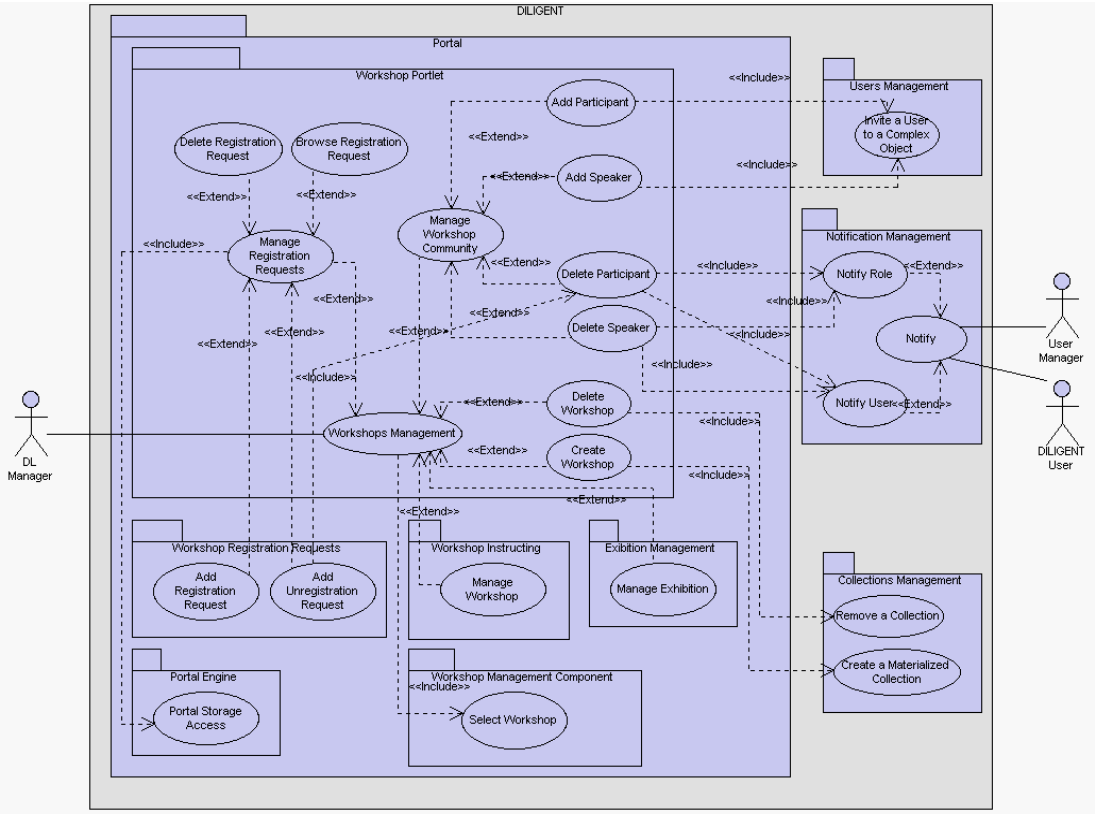

Figure 86: Workshop Management portlet (use case diagram)

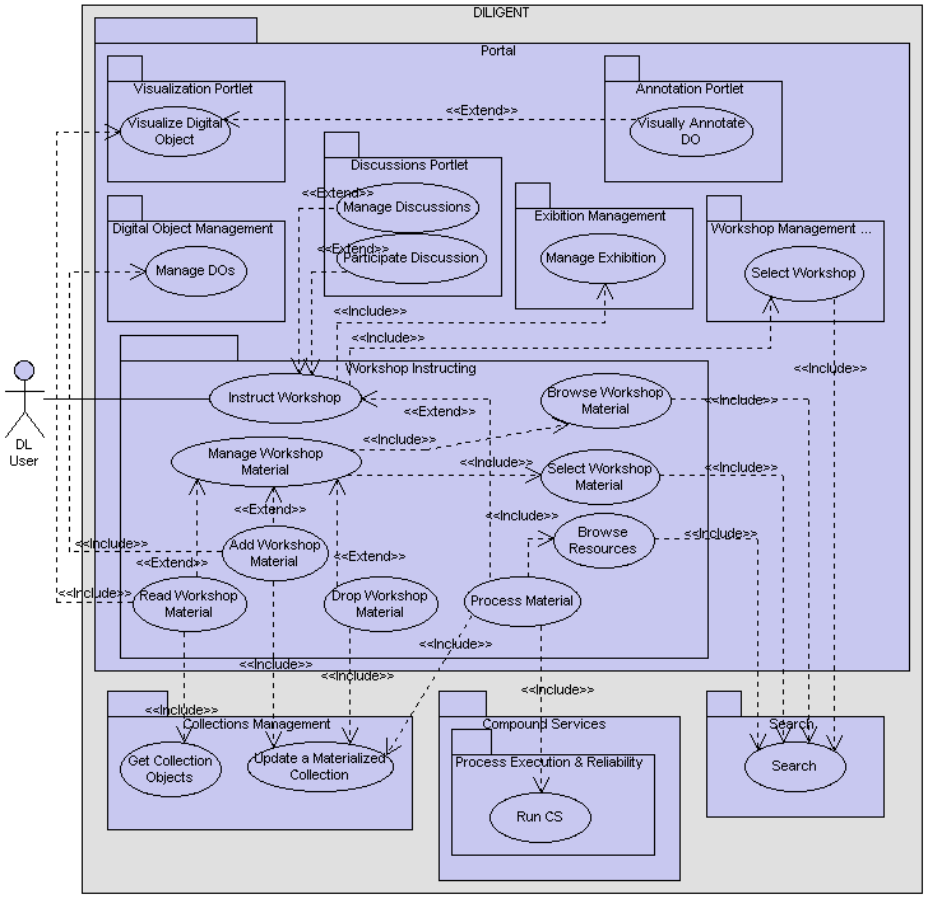

Figure 87: Workshop Instruction portlet (use case diagram)

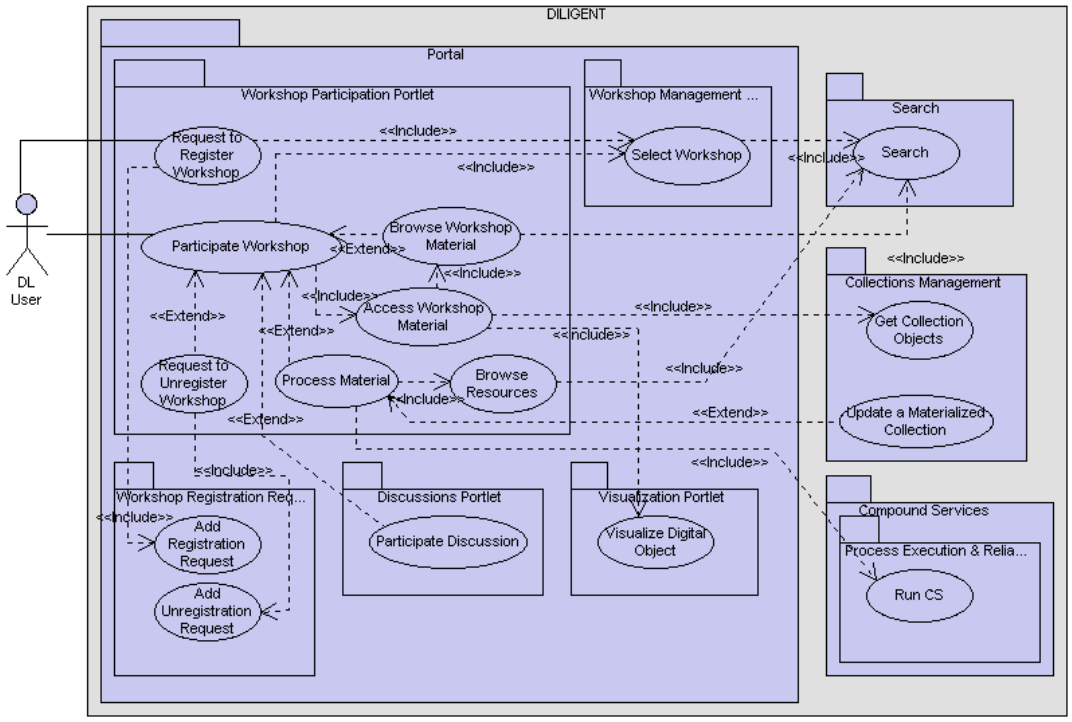

Figure 88: Workshop Participation portlet (use case diagram)

Grid Exploitation

There is no need for grid exploitation specific to this functionality.

System Integration

This functionality is part of the application layer of services. However it exploits the functionality various other services and facilities such as:

- Search
- Annotation engine
- Discussions
- Storage
- Visualization

8.2.16 Exhibition Management

Description

Exhibition management is a group of facilities that allow the design of virtual subcollections of a Digital Library objects for the creation of virtual exhibitions.

The functionality to be provided allows three types of uses:

- Administrative, that will allow creation and low level management of Exhibitions;
- Authoring, that will allow the addition and removal of objects from an exhibition, as well as the annotation of them within the context of the exhibition;
- Browsing, that will provide the means for viewing the objects exhibited in an exhibition.

These functionalities will be supported by dedicated user interface constructs that will allow the selection, editing and visualization of the various attributes and objects.

Priority

This functionality has a low priority since it is not essential for the operation of a Digital Library.

Requirements

The functionality described here is a direct consequence of the ARTE user scenario.

Numbers

There is no clear estimation about the volumetrics of this functionality, however we expect to be used by small portions of a registered Digital Library users.

UML Diagrams

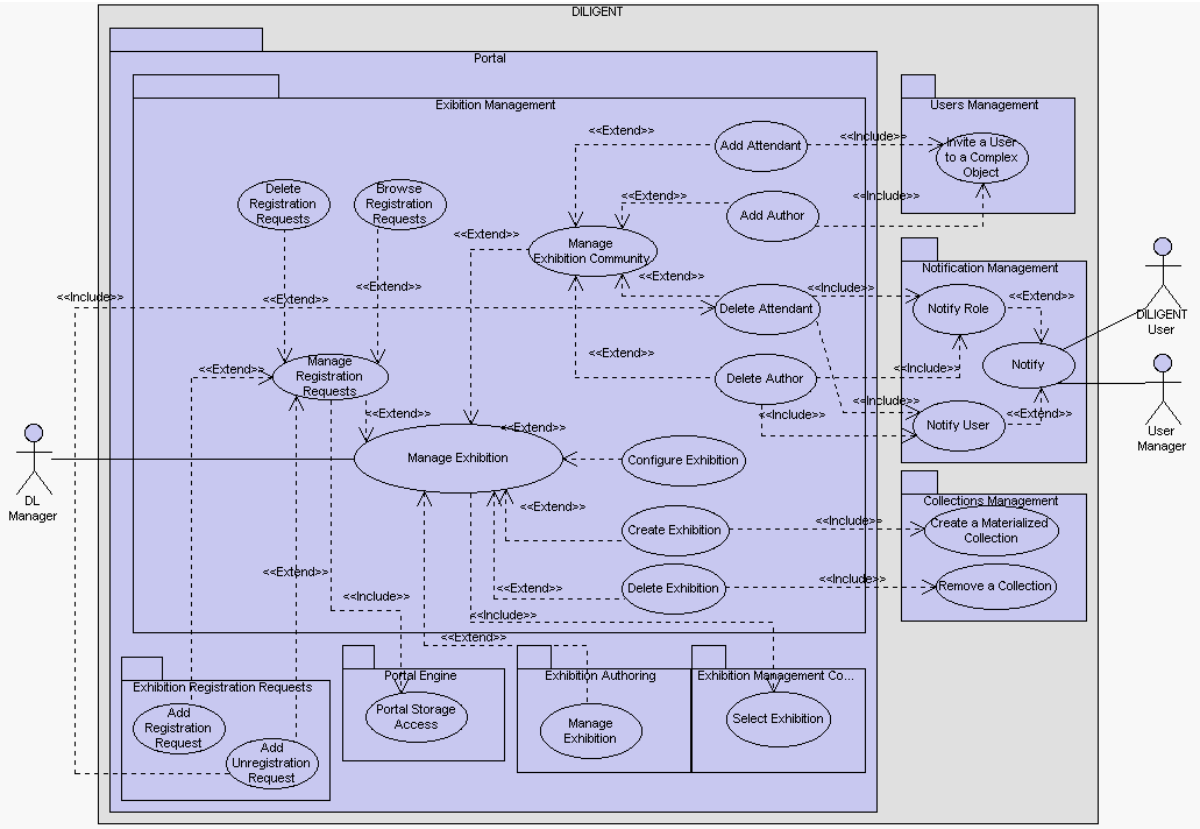

Figure 89: Exhibition Management portlet (use case diagram)

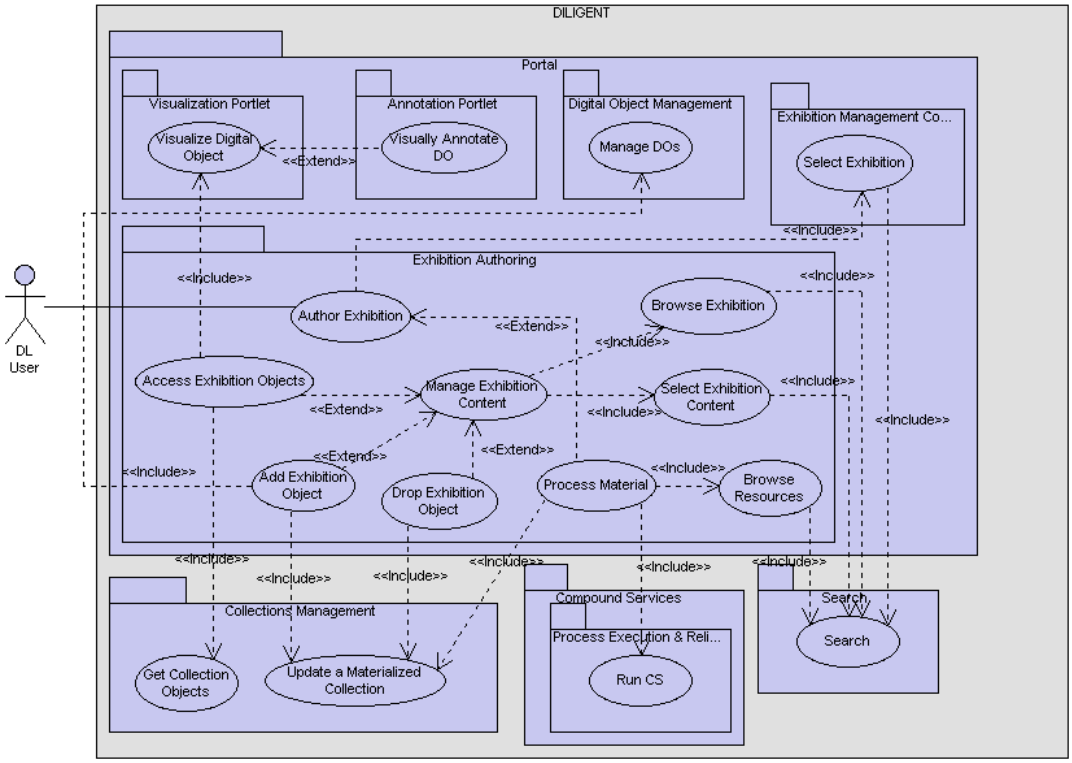

Figure 90: Exhibition Authoring portlet (use case diagram)

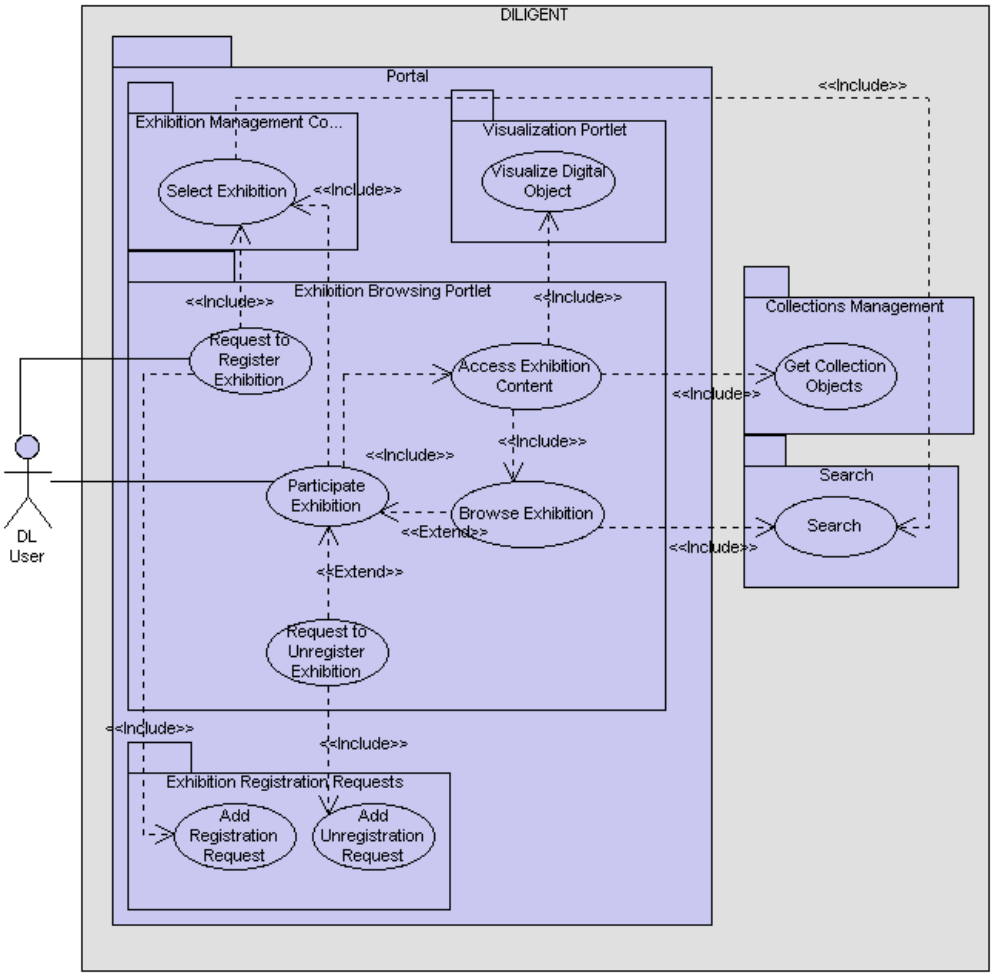

Figure 91: Exhibition Browsing portlet (use case diagram)

Grid Exploitation

There is no need for grid exploitation specific to this functionality.

System Integration

This functionality is part of the application layer of services. However it exploits the functionality various other services and facilities such as:

- Search
- Annotation engine
- Storage
- Visualization

8.2.17 Portal Management

Description

Portal personalization refers to the capability of the generic Diligent Portal System to adapt to application specific needs.

This functionality will be supplied by allowing the DL Manager to configure the behaviours of the portlets that are available to him/her, enable them to end users etc.

Priority

This use case has a "mandatory" ranking of priority. The selected portal engine and the specific portlet implementations will provide support for it.

Requirements

The inclusion of the functionality is based on a decision taken indirectly from user requirements.

In order to provide the functionality the Portal Engine must provide the overall concept and the specific portlets must support configurability.

Numbers

Digital Library Managers are the ones to be supplied with adequate privileges to configure it.

UML Diagrams

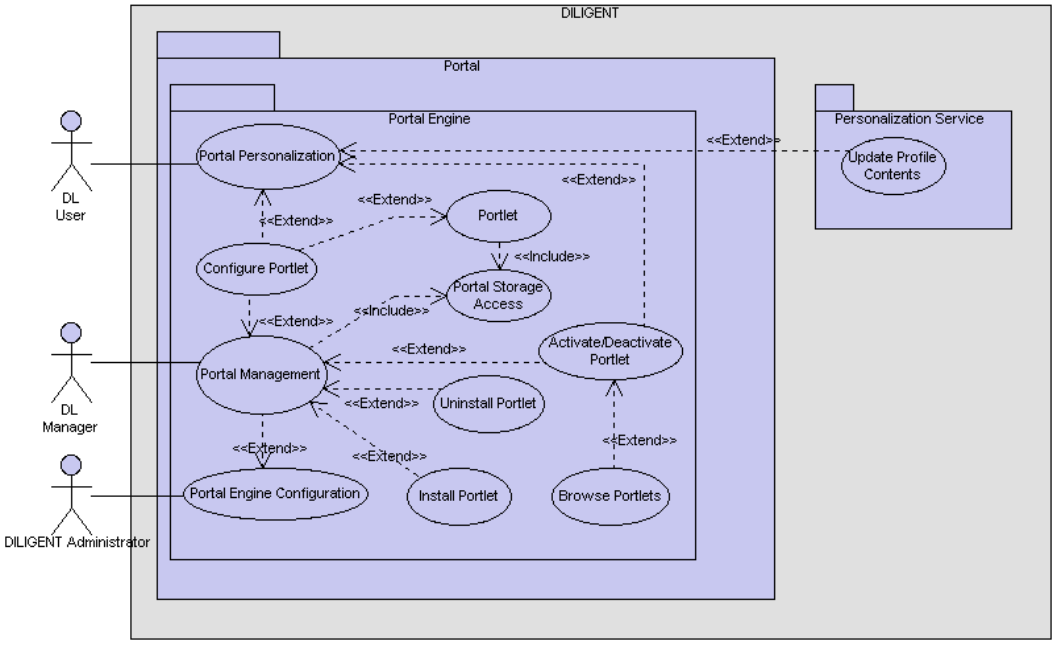

Figure 92: Overall Portal Configuration (use case diagram)

Grid Exploitation

Portal Engines usually do not exploit grid facilities. However if the engine supplies such an option the Diligent Portals will exploit it.

System Integration

Portal Engine is part of the application layer services and configuration / personalization falls into the same scope. They might require support for storing preferences and they will indirectly require authentication and authorization support by the substrate.

Related non-functional requirements

An open-source engine that bases its operation on well-known standards should provide the functionality. This will allow migrating from platform to platform and version to version.

8.2.18 Portal Engine Configuration

Description

Portal engine configuration refers to the low level configuration of the portal engine, which usually deals with managing configuration files, environment settings, configuration of dependent components etc.

Tools for doing this are low-level editors, platform specific configurators etc.

Priority

This use case has an "optional" ranking of priority. The selected portal engine and the specific portlet implementations will provide support for it.

Requirements

The inclusion of the functionality is not based on user requirements.

The need for functionality rises from the observation that most Portal Engines at some point require configuration at a lower level than the Structural configuration of the portal.

Numbers

Every authorized Digital Library administrator will make use of the Portal Engine at least when initiating a DL. Subsequent access of the functionality will be required to fine-tune it and adapt to user requirements or environment changes.

UML Diagrams

Figure 92: Overall Portal Configuration (use case diagram) contains the relevant use cases.

Grid Exploitation

Portal Engines usually do not exploit grid facilities. However if such an option is supplied by the engine, it will be exploited by the Diligent Portals.

System Integration

Portal Engine is part of the application layer services and configuration / personalization falls into the same scope. They might require support for storing preferences and they will indirectly require authentication and authorization support by the substrate.

Related non-functional requirements

The functionality should be provided by an open-source engine that bases its operation on well-known standards. This will allow migrating from platform to platform and version to version.

8.2.19 Portal Personalization

Description

Portal personalization refers to the capability of the Portal to adapt to user preferences.

This functionality will be supplied by allowing the user to configure the behaviours of the portlets that are available to him/her.

Priority

This use case has an "optional" ranking of priority. The selected portal engine and the specific portlet implementations will provide support for it.

Requirements

The inclusion of the functionality is based on a decision taken indirectly from user requirements.

In order to provide the functionality the Portal Engine must provide the overall concept and the specific portlets must support personalization.

Numbers

Every authorized Digital Library user will make use of the Portal Engine and will be able to configure it. However it is most probable that this functionality will not be provided to visitors, or at least it will be severely limited.

UML Diagrams

Figure 92: Overall Portal Configuration (use case diagram) contains the relevant use cases.

Grid Exploitation

Portal Engines usually do not exploit grid facilities. However if such option is supplied by the engine it will be exploited by the Diligent Portals.

System Integration

Portal Engine is part of the application layer services and configuration / personalization falls into the same scope. They might require support for storing preferences and they will indirectly require authentication and authorization support by the substrate.

Related non-functional requirements

The functionality should be provided by an open-source engine that bases its operation on well-known standards. This will allow migrating from platform to platform and version to version.

8.2.20 Discussions

Description

Discussions functionality is a simple facility to allow DL users to perform a conversation on a given topic.

Two types of access will be allowed:

- Administrative, which allows creation and management of topics to be hosted within a DL
- User, which allows posting of comments on a given topic.

Priority

This use case has a low priority, because users can achieve similar results by other means, e.g. external forums or e-mail lists.

Requirements

The inclusion of the functionality is based on a decision taken indirectly from user requirements, in the ARTE Scenario, to support Course and Workshop management use cases.

Numbers

There is no clear estimation about the volumetrics of the aforementioned functionality, however we expect to be used by small portions of a registered Digital Library users.

UML Diagrams

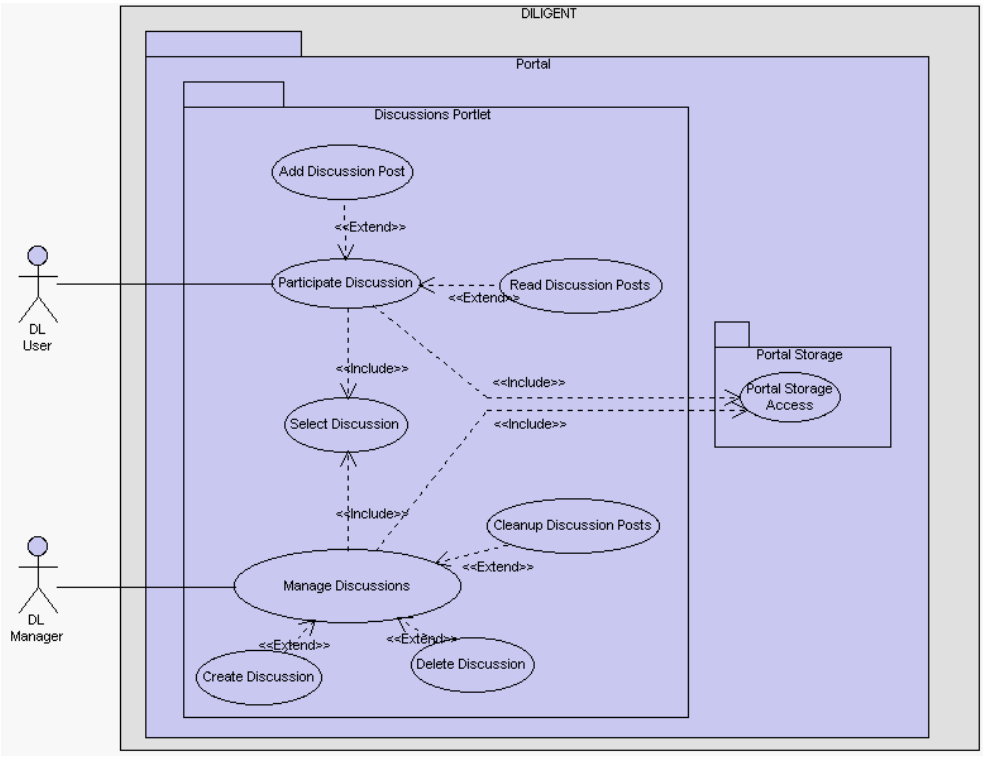

Figure 93: Discussion portlet (use case diagram)

Grid Exploitation

Will not directly exploit any grid facilities unless provided by the Portal Engine.

System Integration

Discussions functionality belongs to the application specific layer services.

8.2.21 Process Image

Description and Priority

According to the requirements document this functionality is the origin of all image-processing operations. It comprises image segmentation, which partitions a given image into a number of segments. In terms of the goals expressed in the DoW, this functionality is of importance (medium priority) for content-based access to multimedia digital libraries, which frequently encompasses search requests for images that contain some given image.

Functional Requirements

In terms of the ARTE user requirements document, this functionality is mainly intended to split up given drawings into *semantic* partitions.

Numbers

This functionality will be invoked whenever a user decides to (1) have the system identify partitions of an image or (2) search for images in the DL by similarity based on a part-of relationship.

Constraints and Assumptions

Given the state of the art in image segmentation which rests upon low-level features like color distribution, shape detection, spatial proximity etc., semantic partitioning is, in general, a difficult problem. This problem is even exacerbated by the fact that the documents covered by ARTE research subjects widely differ from "typical" assumptions in image segmentations. For instance, colour information is completely missing in many of those images. Therefore, the segmentation task heavily relies on appropriate image segmentation techniques, which are to be provided by the "Resolve Image into Parts" functionality.

UML Diagrams

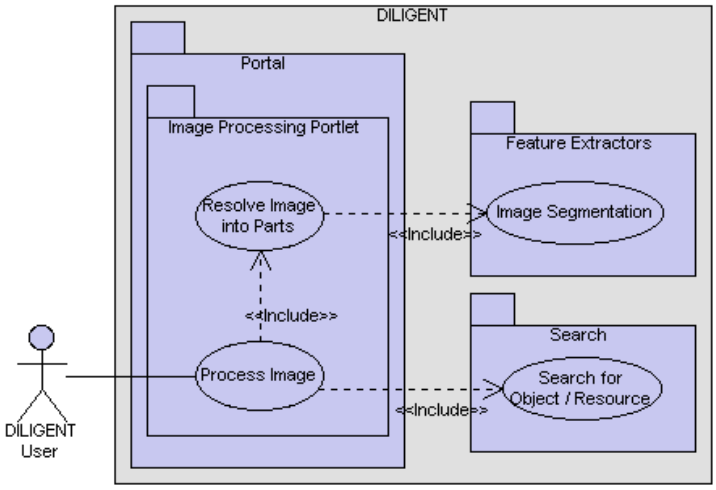

Figure 94: Image Processing portlet (use case diagram)

Grid Exploitation

Usage scenarios of the "Process Image" scenarios may occur both in *batch* and *online* (i.e. interactive) use. That is, images which are to be stored in the DL will typically be batch-processed when they are fed into the system or at some other point in time way *before* the

result of this processing will be used by some search service (e.g. "Search for Object/Resource"). Sophisticated image processing algorithms are computationally intensive. Batch image processing can be parallelized in a way that images may be independently distributed onto different nodes. As image partitioning is executed only once, real-time performance is not an issue.

Online invocation of the service will occur on demand in query processing on the given query image. Typically, region matching does not establish a simple 1:n relationship between the whole query image and one or more segments of the images in the DL but rather assume a m:n relationship of some query image's segments.

System Integration

Process Image functionality belongs to the application specific layer services.

Use Stories

See ARTE requirements document.

Testing Issues

Besides ordinary software evaluation, the major focus in testing this functionality must be put on how well the image segmentation (1) identifies semantically related parts of the image as one partition and (2) to what extend it strips out non-related objects from it. Based on a hand-segmented set of sample images, this can be related to classic information retrieval measures. Another important aspect is given by the runtime performance of online image segmentation that must meet the requirements of real-time answering times for a issued similarity query.

Related non-functional requirements

It must be pointed out that the reliability of this functionality is bound to the potentials of the underlying image segmentation algorithm/component. As we do not "invent" new feature extraction mechanisms in DILIGENT but rather incorporate existing ones it is important to teach potential users (i.e. the ARTE community) in the restrictions in automated image processing as a building block for their (image) retrieval tasks. In particular, it must be emphasized that we cannot guarantee results that precisely meet the user's expectations.

8.2.22 Resolve Image into Parts

Description and Priority

The "Resolve Image into Parts" Use-Case has been introduced by the ARTE requirements document as a component part of the "Process Image" Use-Case. It is responsible for identifying the semantic partitions of an image while the actual partitioning is carried out by the "Process Image" Use-Case.

Functional Requirements

The requirements that a functionality of this kind would fulfil lie within the foundations for the "Process Image" Use-Case. In terms of feature extraction (presumably carried out by external proprietary components), a distinction between the segmentation identification and image partitioning functionality is no longer justified nor desirable. In detail, "Process Image" does not necessarily have to physically separate an image into a number of image files but rather generate the geometric boundaries of distinct partitions and possible establish relations between them (like "belongs together"). In that sense, the following statements augment the statements with respect to the "Process Image" Use-Case.

Numbers

The usage numbers equals the invocations of the "Image Segmentation" functionality.

Constraints and Assumptions

One or more image segmentation components must be present in order to provide this functionality. It is likely, that different application scenarios have a widely different notion of meaningful image segmentation. Therefore, this functionality must be parameterised in order to pick the right image segmentation algorithm and invoke it appropriately.

UML Diagrams

See Figure 94: Image Processing portlet (use case diagram).

Grid Exploitation

For the time being, grid exploitation is given by batch processing (i.e. segmentations) of large image collections stored in the DL. Online (interactive) use (like in query processing) will likely involve one or few query objects, only.

System Integration

Being merged with the "Process Image" Use-Case this functionality belongs to the application specific layer services. It make use of the "Image Segmentation" offered by the "Feature Extraction" service.

Use Stories

See ARTE requirements document.

Testing Issues

See "Process Image" Use-Case.

Related non-functional requirements

See "Process Image" Use-Case.

8.3 Report Management

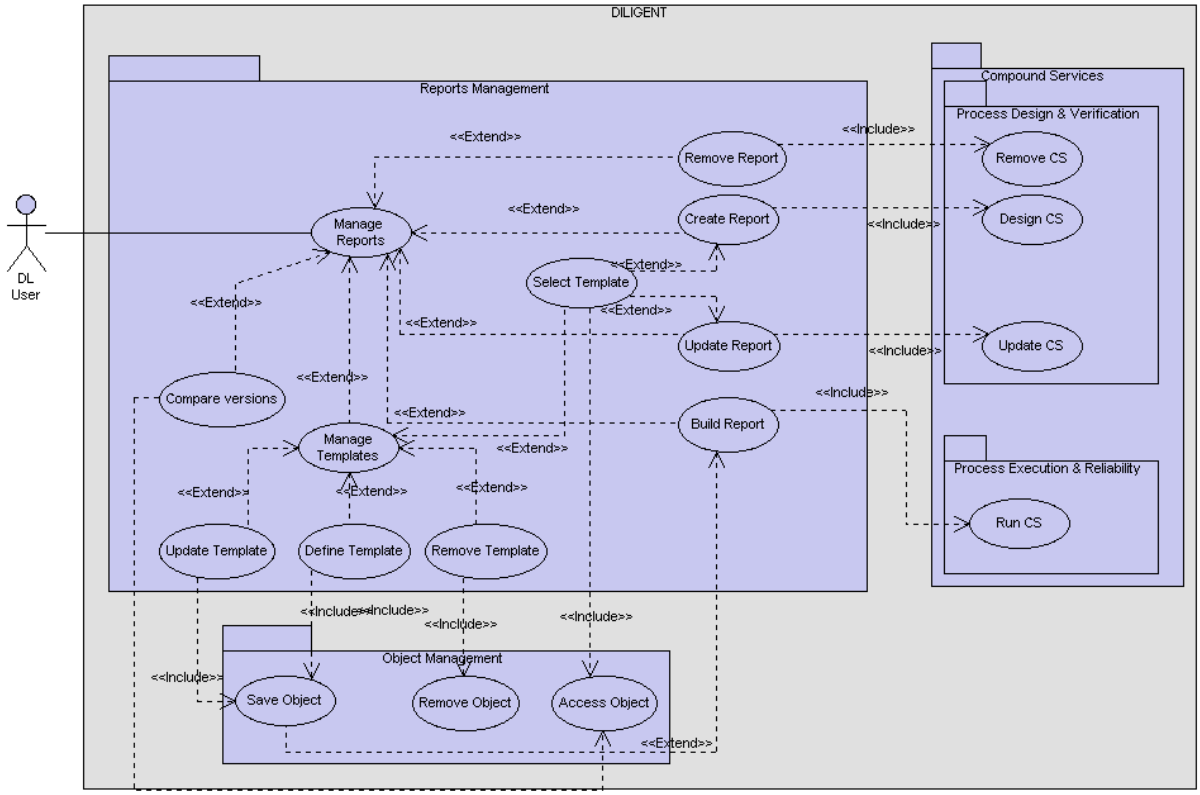

Figure 95: Report Management (use case diagram)

8.3.1 Manage Report

Description and Priority

Report generation is an important end-user requirement in the ImpECt scenario. Automatic report generation using templates is popularly used as front-ends with databases at the back end. However report generation is a complex problem, where many layers of data-representation are required. At the superficial level, there is the layout and styling data, which is above the service layer that gets data from the data layer and transforms it accordingly and formats it according to the style.

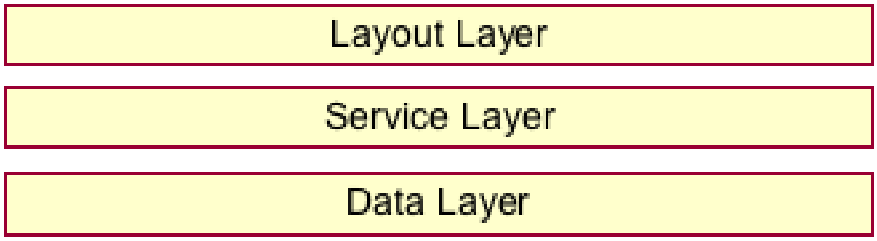

Figure 96: Multiple layers of an automatically generated Report in DILIGENT

Report Templates

Automatically generated reports are often associated with templates that can be visually manipulated and new content can be added which uses the existing styles in the template. A template could specify a style for each kind of *object* in the DILIGENT scenario (e.g. images have to be maximum of width 400 pixels and therefore should be automatically scaled and then centred). For this purpose using *slots* inside the template might work in the following way; each *slot* could specify what kind of objects can be fitted into this slot, and

the template would have a collection of styles for each kind of object that can be fitted into the slots in that template.

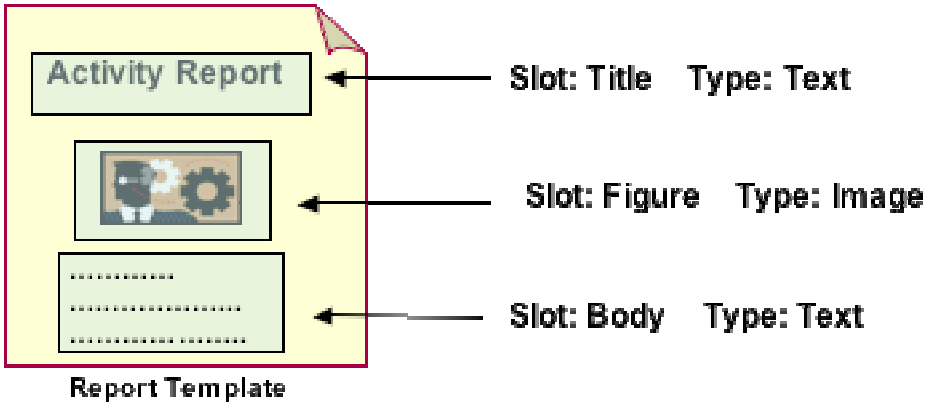

Figure 97: Report Template structure example

Requirements

Report Management requires input from the users to design 1-2 templates from the layout point of view. These templates could be in XHTML format that can further be used to define slots.

Numbers

This functionality is likely to be used frequently (10-100 times a week)

Grid Exploitation

Report Management has a 3 layered architecture with the user interacting only at the service and layout layer, i.e. for every slot, the user specifies a service which provides input to the slot, and then also define the style for this kind of object.

8.3.2 Create Report

Description and Priority

This functionality enables a DILIGENT user to create reports based on existing templates made available in a template repository.

Requirements

For the user to be able to create reports, templates should be available in the template-repository. Also, the user should have the requisite permissions to create new reports.

Numbers

Several times per year.

Constraints and Assumptions

In the early version, the user will only be able to select from existing templates. Creating new templates or modifying existing templates is currently not supported.

Grid Exploitation

The report creation service can easily be deployed on different grid nodes to improve performance.

8.3.3 Select Model Definition

Description and Priority

This functionality allows the report-creator to choose from the already provided templates. The user can then use this template to fill up with objects and tie up with services that generate them.

Requirements

The end-users have to provide feedback on the kind of templates required. 1-2 of such templates can them be designed and made available in the system.

Numbers

Several times per year.

8.3.4 Update Report

Description and Priority

With this functionality, the report creator can change an existing report.

Update can mean:

- removing an object from a slot and adding a new one
- duplicating a slot and adding a new object there
- removing a slot
- re-order slots

Requirements

A report has already been created and the user has prerequisite access rights

Numbers

Several times per year

Grid Exploitation

This functionality can be easily deployed on different grid nodes since the underlying services used to provide objects for the slots can be invoked from any grid node.

8.3.5 Remove Report

Description and Priority

Even though reports tend to be permanent documents, it should be possible to remove a report if required. If supplied its identifier, a particular report can be retrieved from the report repository and then removed.

Numbers

Several times per year.

8.3.6 Build Report

Description and Priority

In the creation stage of the report, the slots in the report template are filled with bindings to the services or to objects on the file system. Building the report evaluates all the bindings and creates a static document.

Requirements

All the services/ bindings invoked in the report exist and there is no type mismatch for any slot.

Numbers

Several times per year.

Grid Exploitation

This process can be gridified since all nodes can invoke service on the grid and access the file system.

8.3.7 Compare versions, Visualize versions

Description and Priority

This functionality allows a report creator to find the differences in two versions of the same report. Since version management is a complicated functionality, and there exist very good free tools like CVS and subversion for this functionality, we propose using CVS as the backend for version management of reports. The report management tools can provide a report – specific functionality on top of CVS.

Further, the comparison is visualized by using a visualization component.

9 CONCLUSION

This report presents the result of the activity conducted within the "WP1.Test-bed functional and architectural design", task "T1.1.1 Test-bed functional specification" of the DILIGENT project in the period November 1st 2004 - February 28th 2005. By exploiting the analysis of the user-communities requirements, collected in WP2.1 and WP2.2, the *Test-bed functional specification* describes and specifies the functions and features of the DILIGENT system that will be perceived by the users and it formalizes how the users will interact with the system.

The functional specification is given in the formal notation recommended by the Unified Process software engineering methology, according to Annex I – "Description of Work"; this formal notation is accompanied by texts as expressed by the DILIGENT technological partners.

This activity has first analysed the user requirements reported in "D2.1.1 ARTE Scenario Requirements Analysis Report" and "D2.2.1 ImpECt Scenario Requirements Analysis Report" in order to identify common basic functionalities matching the requirements of the user scenarios; then these two groups of functions have been associated with the main functional area of the DILIGENT project represented by WP1.2-WP1.6; after that, each partner has focused its analysis on the specific aspects in which it is mainly involved and in which it has more competences, contributing in this way to the specification of the functional view of the entire system; finally all contributions has been collected, analyzed, and integrated by CNR, leader of Task 1.1.1, with the support of all the other partners.

The complete functional specification will be used by WP1.2-WP1.6 in order to define data elements, interfaces, and outputs for the rapid prototyping and testing of all services.