EVS4CSCL PROJECT

Electronic Voting System for

Computer Supported Collaborative Learning

Final Report

OPEN UNIVERSITY OF CATALONIA (UOC)

Master's thesis in Computer Science

Computer Science Engineering (2nd cycle)

Master's thesis candidate:
Daniel Rodríguez Calafat

Advisor:
Santi Caballé

June 2011.

I wish to give my special thanks to my advisor, Mr. Caballé.

Abstract

The EVS4CSCL project starts in the context of a Computer Supported Collaborative Learning environment (CSCL). Previous UOC projects created a CSCL generic platform (CLPL) to facilitate the development of CSCL applications. A discussion forum (DF) was the first application developed over the framework. This discussion forum was different from other products on the marketplace because of its focus on the learning process. The DF carried out the specification and elaboration phases from the discussion learning process but there was a lack in the consensus phase. The consensus phase in a learning environment is not something to be achieved but tested. Common tests are done by Electronic Voting System (EVS) tools, but consensus test is not an assessment test. We aren't evaluating our students by their answers but by their discussion activity. Our educational EVS would be used as a discussion catalyst proposing a discussion about the results after an initial query or it would be used after a discussion period in order to manifest how the discussion changed the student's mind (consensus). It should be also used by the teacher as a quick way to know where the student needs some reinforcement. That's important in a distance-learning environment where there is no direct contact between the teacher and the student and it's difficult to detect the learning lacks. In an educational environment, assessment it's a must and the EVS will provide direct assessment by peer usefulness evaluation, teacher marks on every query created and indirect assessment from statistics regarding the user activity.

Keywords: Electronic Voting Systems, EVS, Computer Supported Collaborative Learning, CSCL, CLPL, Discussion Forum, DF, consensus, e-learning.

Project Area: Computer Supported Collaborative Learning - CSCL

Contents

Introduction

The Electronic Voting System for Computer Supported Collaborative Learning project (EVS4CSCL) starts in the context of an e-learning Discussion Forum (DF) prototype [42] created by the UOC [6]. This DF was a proof of concept for a Generic Platform for CSCL applications (CLPL) [1][2]. This framework brings to their users the potential to create a complete CSCL application from specification to implementation in the time of one semester.

After the success collected by the DF in terms of satisfaction of students and tutors there is an opportunity to improve the DF with add-on applications to improve the educational discussion experience. The discussion forum carried out the specification and elaboration phases from a discussion but the consensus phase was uncovered. Being the consensus phase something that is not achieved but tested [14][19] an Electronic Voting System (EVS) will be integrated with the DF for that purposes.

Project justification and context: starting point and project contribution.

The research manifests that EVS systems were already successfully used in lectures from many universities and schools world-wide [8][15][16]. Some papers show the EVS as a catalyst for the discussion in the classroom [5]. Experts talk about the success of an electronically enhanced classroom interaction [9][17]. There is a new educational methodology involving large audiences because of the teacher facility to get direct and immediate input from the students [7].

Even if these references are about real, synchronous, face-to-face interaction, the results are good enough to start thinking about how to bring these benefices to our on-line students in a distance learning environment.

Our project will implement an application that can be used as a discussion starter, an immediate feedback application for the tutor and finally a discussion forum consensus tool. We want to offer to our on-line students the same educational opportunities as the face-to-face students get from other EVS or Audience Response Systems (ARS)[18].

Project objectives

To create an application that can be used as a consensus maker in the context of the discussion forum.

To create an application focused in educational specific requirements. Facilitating the user's awareness about the learning process and providing the tutor by effective data that can be used to elaborate the student assessment.

To create an application that can be easily adapted and extended in order to act as a base system for further developments in the area of the EVS CSCL applications.

Metodology

EVS4CSCL follows the classic life cycle for information technology applications. Each phase will produce a deliverable approved by the advisor to assure the project development success.

Planification

The project phases are scheduled by the advisor from his experience with similar projects.

Deliverables	Begin Data	Data due
Project Plan	03/03/2011	14/03/2011
Project Specification	15/03/2011	28/03/2011

Project Design	29/03/2011	18/04/2011
Project Implementation	19/04/2011	30/05/2011
Project Memory and presentation.	31/05/2011	20/06/2011

Chapter description

Project Specification

This is a document describing the result of analysis, brainstorming and technical issues. In this document we will describe the application with detail enough to design it in the next phase. The reuse of the CLPL Platform specification will be useful in order to increase our productivity.

Project Design

It's a document describing all application components and interactions. We will create a document with all the design diagrams needed to go easily into implementation without surprises. We will continue reusing diagrams and concepts from the CLPL Platform as a key feature to obey the project timeline.

Project Implementation

In this phase we will deal with technology issues and the learning curve of all the technologies involved in the project. The final deliverable in this phase will be a working EVS application integrated by the DF and an application user guide.

Chapter 1. Specification

State of the art of EVS

Learning context

Electronic Voting System (EVS) is a largely referred term in learning environments but almost all papers are about interactive synchronous classrooms. A common reference is: *The use of electronic voting systems in large group lectures: challenges and opportunities* [17].The document is centered on students acting like the public in the TV quiz show *who wants to be millionaire?* This kind of EVS is also known as *Audience Response System* [18] (ARS).

There are some references about face-to-face classrooms using EVS to improve the CSCL at some phases of the course [16]. Even it's important to get a look over this point of view, in our e-learning environment we don't have any face-to-face interaction in the whole course. We don't have a blended course but a true distance course.

E-Learning context

Reading about voting systems in referenced papers we find two key concepts.

1st. When evaluating the benefices of voting systems in a classroom it seems clear than there is a benefice. Actually the success comes from the interactivity in the classroom more than for the voting system itself [3].

2nd. Apparently it's difficult for the students to interact in a discussion forum. Even if the CSCL provides a good forum, a large number of students are uncomfortable in this kind of communication. As seen in course evaluation tests, the students would prefer other communication tools like e-mail or messenger for discuss and they will use the forum barely to show the final results [4][11].

Both papers confirm the role of the tutor is very important in this kind of environments, and the tutor needs to be trained to get special skills in these new communication scenarios.

According to the papers, students are asking for "closed answer models", templates or other tools helping them to be comfortable collaborating in the discussion.

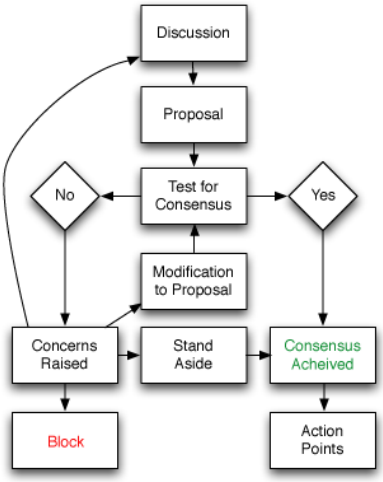
When the students are in public anonymous forums they write without any limitations but in a classroom discussion forum they're afraid by the colleagues and teacher's perception of their comments. The template should be useful to show the results of their work without being personally exposed.

Referred papers told us about how important is to the student to be anonymous. Every time they offered the option anonymous (alias) the student took it instead of the real name.

Consensus and decision making

Our EVS will be a consensus, or even a decision making tool. A look in that area is needed to know how an EVS can help in this process. How the EVS will be interacting in the classroom to achieve consensus? What kind of queries are the best for that? When initiate the query?

As seen in on-line references [14 - 16] there is no way to achieve consensus by a query. Consensus will be achieved by discussion; the query is shown in the picture below [19] as a *test of consensus* and acts as a discussion catalysis tool [5] but it's only a piece in the whole process. Being our EVS an important component in the diagram, it's important to have integration between the *discussion* and the *test of consensus*. We are integrating our previous DF with our new EVS component in order to complete this whole process.



We will assume that the teacher/tutor is who could make a well usage of the queries and help the group to achieve consensus. In a student's group, we will need a member acting as a consensus member. This member needs to be concerned about consensus in spite of a personal victory. The success of the process is not relied to the tools themselves but to the human interaction. Choosing the topic and testing for consensus at the right time will be the key of success and only well trained teachers will be able to do it.

However there is an educational interest in giving to all our students the chance to make their own queries to create more discussion, not necessarily consensus. Bringing the query creation to all our students it's a way to create new discussions, and that way the query creation will be also evaluated as part of the complete student assessment.

The problem here is how to deal with a large number of queries. For sure, we are creating queries with a probably low number of votes, even zero votes.

We don't have references about that over EVS, but over common discussion forums we can find threads without any answers at all. Compete for attention will be a good way to motivate students to make interesting queries. For sure there will be a lot of queries unanswered if the number of queries grows up to uncomfortable limits.

Further projects may study these suppositions and determine where the limit of comfortable open queries is. Finally, without references about, we can't figure out how to keep the voting system comfortable and we will not define any technical rule to keep the system usable. We will assume that students and teachers can do it by themselves.

Students and specially teachers need to be trained in the use of this new way to collaborate. Our voting system will be good for consensus only if users are educated in how to use voting systems in a consensus scenario [8][10][13].

Being a community tool the success of the project remains in the user's community most than in the application itself.

Conclusions

The major problems reflected above are methodological, educational or social and will not affect our project. But there are some ideas to keep in mind from the above lines:

The vote should be anonymous, or almost optionally anonymous. Otherwise we would get only "socially correct" answers from our students not the answers they really wanted to give.

Showing the vote results in early times may derive in a "follow the leader" response [4]. The right time to show results may be eligible by the query owner, and not only date/time centered to improve the learning process.

Giving to all our students the creation query tool may derive in an estimated 20% active, more technology comfortable students, conducting all the queries with their own problems [4]. A simple one-click form template should be used in order to facilitate the query creation to all our users.

Working in little groups is a methodology to avoid public individual social exposure; our project should consider the team support in the query creation process to facilitate all students' integration and participation.

Finally if open responses or rationale answers are described as the better learning tools, our EVS should include this kind of quiz in the development. Although we will not forget the simple queries because students showed more comfortable with templates and options that not expose their identity or idiosyncrasies publicly on-line.

State of the art for voting systems

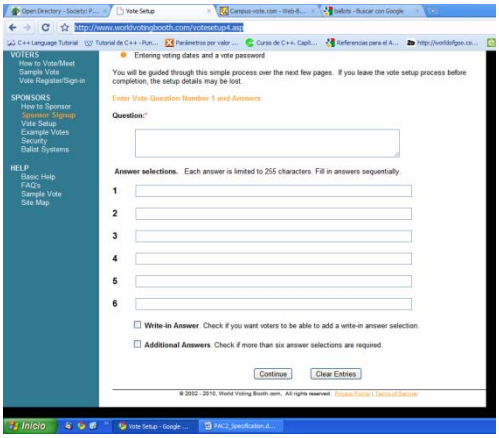
Almost all the EVS papers and references are about hardware components. When referring to software tools in CSCL environments we will find tools to do polls, surveys, classroom additive asserts and final evaluation (test).

We didn't find[1] software EVS tools to improve the learning process in e-learning communities as the EVS hardware tools do for classrooms lectures.

Better than using the same tools in face-to-face and distance communities, changing a hardware piece for a software component, we needed a new approach trying to offer to e-learners and to e-teachers the same learning opportunities. [5]

Most of the voting systems observed are democracy tools [12]. These tools are focusing in democracy, more than in learning or consensus. We have selected few applications to get a wide view of the voting systems applications marketplace. Keywords used in our searches comes from decision making to consensus tools, including obvious keywords like CSCL, EVS, query, voting system and so.

URL. *www.worldvotingbooth.com*

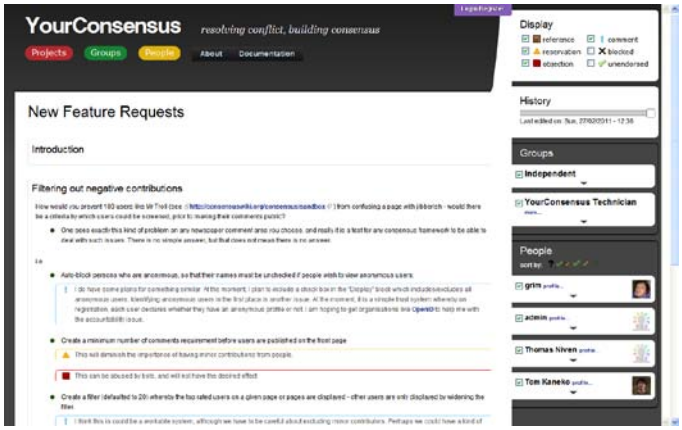


The **world voting** is a killer application in electronic voting systems. Offering the most used voting methods and a secure and clear interface they only focus on the voting system, not in the discussion.

- **Standard Voting**. Conventional, most common one person, one vote method.
- **Approval Voting**. With approval voting the user can select as many answers as it wish.
- **Rank Voting**. If there are six answers the user will 6 to its first choice, 5 to second, etc. This method is also known as the Borda Count method.
- **Cumulative voting**. Each user gets 10 votes to distribute among the options as it desire. Similar to approval voting but with a weighting factor.

[1] To the best of our knowledge, software EVS applications have been little investigated to improve the learning process.

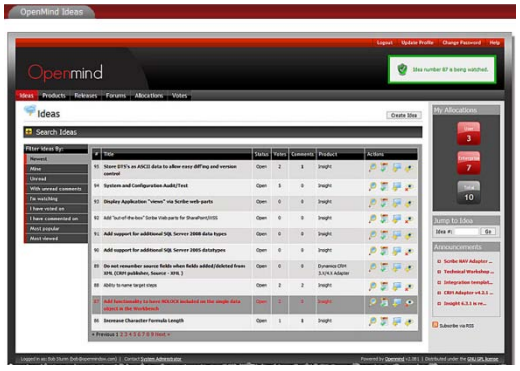
URL. *www.yourconsensus.org*

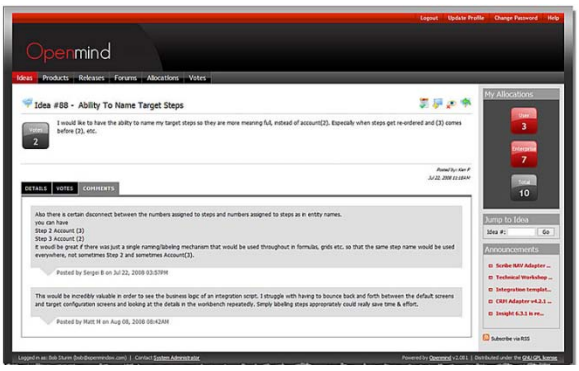


This application tries to get consensus from discussion, but the consensus is never tested. Finally we are in front of an improved discussion forum with mechanisms to avoid endless discussions in community forums but in fact it is not a real consensus tool and for sure not an educational discussion application.

Nevertheless their interface usability is clear and clever and it would be the starting point for further DF and EVS developments.

URL. *openmind.sourceforge.net*





The **OpenMind** application is an open source discussion forum using a voting system to give relevance and order to the forum contributions.

Nowadays this approach has been taking by software companies deciding which features are the first to be developed. Customers can discuss in a forum about new features they want and other customers can vote for a feature to be the first to be developed.

Even if their interface is innovative and intuitive the use of the votes they do is already done in our DF forum as a *usefulness evaluation* of each contribution. It is not the kind of use of an EVS we want to do in our consensus application and it's a business application not an educational one.

URl. *www.optivoteformoodle.co.uk*

Moodle[31] is one of the most used e-learning platforms worldwide. This courseware features polls, assessment tests and quizzes. From the EVS point of view the major development is this hardware ARS integration with the *Moodle* quizzes.

They aren't targeting educational discussion or consensus and of course not an on-line EVS system allowing distance students to use the same learning methodologies as the physical ones [29][30]

Conclusions

We found many EVS applications in different contexts but none has been designed for e-learning environments. They weren't designed to help the teacher or the students in the learning process. None was designed to improve the discussion or the consensus from an educational point of view.

When they were focused on voting they forgot the discussion and when they carried on the discussion there was a lack in voting and monitoring.

Looking at educational platforms, current projects are focusing face-to-face users and assessment or self evaluation tests [13].

Even if we can be inspired by these applications in some areas, a new development is needed to target the e-learning process in general and the consensus achievement in particular. We must provide the right tool for an e-learning scenario [5].

Current Discussion Forum (DF) will be a good starting point. With all the features on the discussion area already done, we will be able to focus on the voting process and keep improving our e-learning platform.

Study into general requirements

The goal of the project is to create a standalone application supporting all the common features from a CSCL application with voting system capabilities. In addition our voting system will offer consensus and decision making capabilities and integration with the DF.

All the common modules (user management, security, knowledge management, functionality and administration) to create a complete CSCL application will come from the CLPL framework.

In fact, all the requirements from the EVS will fit in the CLPL common components.

- The Electronic Voting System is in fact a "Shared Resource" in terms of CLPL.
- A query is in fact a "Contribution" in terms of CLPL.
- A vote is an "Action" in terms of CLPL.

Keeping this in mind, we can specify all the EVS application in terms of CLPL. We will consider two kinds of project components.

- **Core EVS components**: these components will deal with query management, vote and the query results presentation.

- **Common EVS components**: these components are common to CSCL applications like user management, security, logging and general administration.

For the core components, a group of specific voting system features will be created over the CLPL Functionality –create a query and vote it- and CLPL Knowledge –get the results and show them to the users-.

For common project components, we can reutilize without changes components from the CLPL framework. Administrate users, groups, resources, authenticate and authorize aren't functionalities related to the voting system itself but to any CSCL.

EVS Functionality

This component deals with query creation and votes.

First at all, we must select what kind of query will be the best in this scenario. The **selected vote systems** [3][4][8] are:

- Select only one answer.
- Select many answers as you need.
- Rank your choices.
- Rank your choices giving points.

Other decisions related to vote systems are:

- Allow blank vote.
- Allow to show the results before the end of the query.
- Allow open answers.
- Allow explanatory text answers for each answer.

General considerations:

- The query needs an end date.
- The query can only be edited or removed before getting any vote.
- A query will be created from an existing closed query. Duplicate queries are allowed but not at the same period of time in the same classroom.
  Observation: we have in mind to ask the same query at different stages of the discussion to see answers evolution.
- Vote will be secret –i.e. anonymous- to improve participation.
- Each student can vote each query only one time. There is no way to edit a vote when it is emitted.

Special discussion forum features

As a discussion and consensus tool it makes sense to imagine queries -consensus tests- related to discussion forum threads. If we want to discuss over the query, a DF is a must have.

A requirement made by the advisor is that our EVS will be related to the DF to easily create queries from the existing posts in the DF. To solve this problem a mechanism will be used to indicate the best contributions to discuss with the voting system. This "proposed to query" post will be eligible when creating a new query in the main thread context.

For quality voting reasons, it should be better to rewrite the main idea in the post than sending the post itself to the query. More than three lines for each query option may degrade the user vote experience. We encourage rewriting to make better queries.

Features related to DF are:

- Make a query for this existing thread.
- Show query proposals for this existing thread.
- Make a query and open a thread related to discuss it.

- Block thread when the voting process is open. This rule will apply when we want to discuss, to vote and to discuss again about the results.

EVS Knowledge

This component will manage the voting results and other counting from the process.

Proposed **stats on voting results**:
Total students, total votes for each option, total votes, total blank votes, total abstentions, total open answers, first vote & last vote days.

Even if results aren't public available for students between voting days, teacher will ever see partial results in the interest of the group learning process.

The teacher will be the only one seeing the text from the open answers or the explained answers. For the students, open answers will indicate that there is a number of students not agreeing with the current proposed arguments but not the answer itself, to protect user anonymity.

As for the *counting algorithm* there is no controversy. Simple vote counting will be enough for our discussion and consensus purposes. We don't need a winner and a loser; we only need to know what is *trendy* in the discussion.

Detecting problems in the learning process with stats:

- Too many absents points to uninteresting query.
- Too many open answers points to bad selected arguments.
- Too many blank votes points to misunderstanding the query
- Queries going too slow may need intervention to animate the discussion.
- Potentially closed/inactive queries may need attention as they can anticipate conclusions.

Evaluation stats:

- Who never votes? Showing a low interest.
- Who uses to blank vote? Pointing to misunderstanding the queries.

Other stats focusing evaluation can be proposed, but for decision making and consensus making, there is nonsense to look at who wins queries, or who has the best answers. Evaluation will come from discussion interaction instead of our EVS.

Actually we aren't evaluating the vote result itself but making a better discussion and helping students achieving consensus.

Awareness and feedback

- The user will get notifications when a new query is open.
- The user will be able to remember their vote.
- The user will easily access to results of closed queries.
- The ser will know how many queries he not yet voted.
- The most active users will be announced to encourage participation and recognition.

EVS User Management

Create and manage user and groups with their respective roles is a common feature in a CSCL environment like our voting system. This component will deal with this functionality.

This component is totally reusable from the *CLPL User Management* component in CLPL framework.

EVS Security

Even we are protecting privacy and making our votes anonymous, all the users will be authenticated and authorized before to access system features.

This component is totally reusable from the *CLPL Security* component in CLPL framework.

EVS Administration

This component will be charged to carry with the system internal control and the resources management. Even users themselves will manage their own resources acting as group administrators; there is a way to manage the system in terms of performance and security.

This component is totally reusable from the *CLPL Administration* component in CLPL framework.

CLPL reutilization

We will use the same business model proposed in CLPL based on 5 components to specify our system: *functionality management, knowledge management, user management, security management, administration management.*

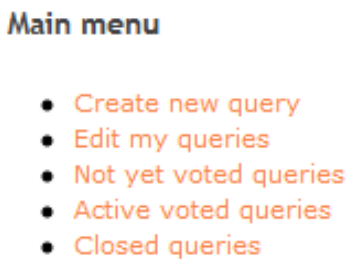
As we show above, our EVS will fit perfectly in the CLPL framework model. From five components three of them –user management, security management and administration management- are completely reusable without changes.

The core components *EVS Functionality* and *EVS Knowledge* will rely on some features related to *CLPL Functionality* and *CLPL Knowledge* but all our concrete functionality and knowledge management will be done without any other help.

Screens specification

Note: we are describing few interfaces to illustrate our specification. Design and implementation will explore the user interface in detail.

- The user will create and edit their queries.
- The user will create new queries from closed queries their own.
- The user will easily gain access to detailed queries information.
- Some of this information will be available as an awareness flags or feedback.



Create Query

When the user creates a new query in a discussion forum context it will see the proposed query arguments and use them, or create a totally new arguments. It will see all the proposals under its current thread level.

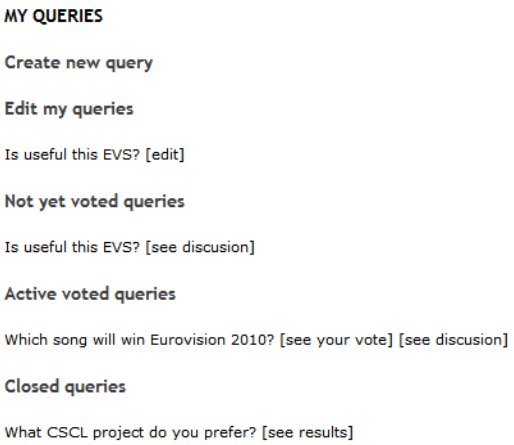
Other features like *block discussion during voting period*, or *make a thread for the query* will be useful in the DF context.

Query will have a timeline (start/end date) and options: allow blank vote? Show results between vote processes? Allow open answer? Justify your answer?

After that, query may be proposed in several ways: ranking –rank your choices-; select only one argument; select many arguments you need; rank counting. Distribute 10 points to your choices.

Queries main menu

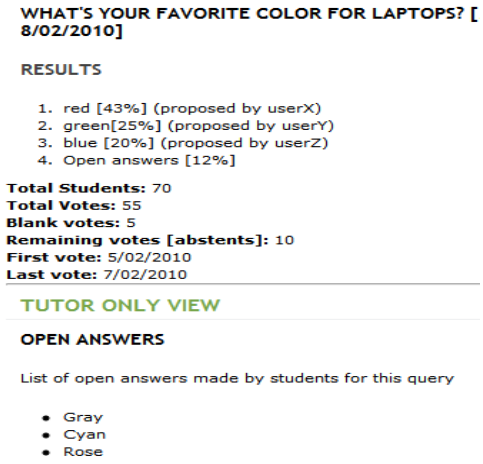
- The user can create a new query.
- The user can **change an existing query if it's not yet voted**. No changes can apply to an already voted query.
- The user will get a notification when a new query is open. (awareness)
- The user will be able to remember its vote.
- The user can access to results of closed queries.



Results

This is the results screen.

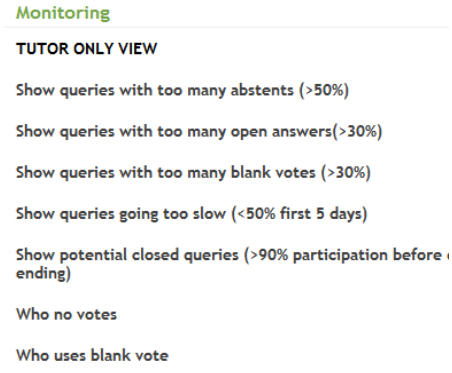
- **Even if results aren't public available for students between voting days, teacher will ever see partial results** in the interest of the group learning process.
- As for open answers only the teacher will see them. For the student point of view an open answer indicates that there are students not agreeing with the current proposed arguments. If needed the tutor will post a list of open answers in the forum to open a new discussion/query.



Monitoring

The monitoring will give to teachers the power of discover lacks in the learning process.

- Too many absents points to uninteresting query.
- Too many open answers points to bad selected arguments.
- Too many blank votes points to misunderstanding the query
- Queries going too slow may need intervention to animate the discussion.
- Potentially closed queries may need attention as they can anticipate conclusions.



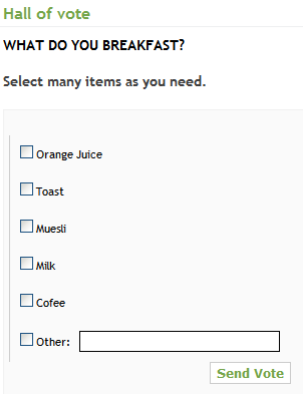
As for evaluation, DF gives support enough to evaluate the discussion learning process.

However EVS specific stats we will introduce are: student activity –created queries-, student reactivity –voted queries-, student abstention –bad attitude-, student blank vote –bad attitude or misunderstanding. We also introduce usefulness peer evaluation and tutor mark.

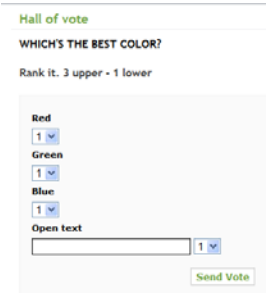
Vote systems

Select only one answer.

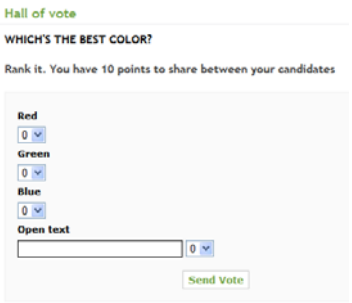
Select many answers as you need.



Rank your choices.

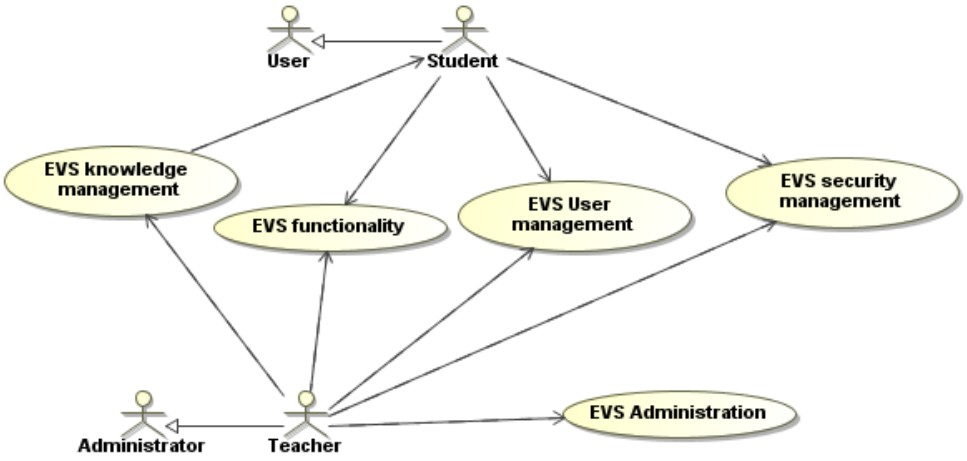


Rank your choices giving points.

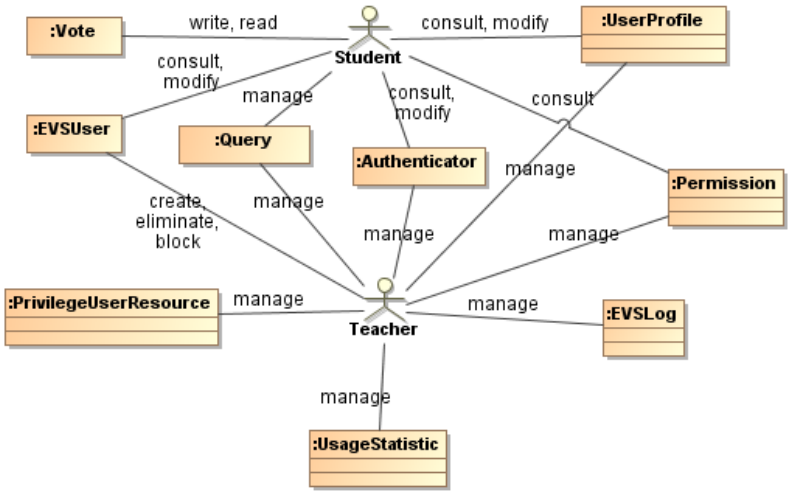


Business model of the entire library

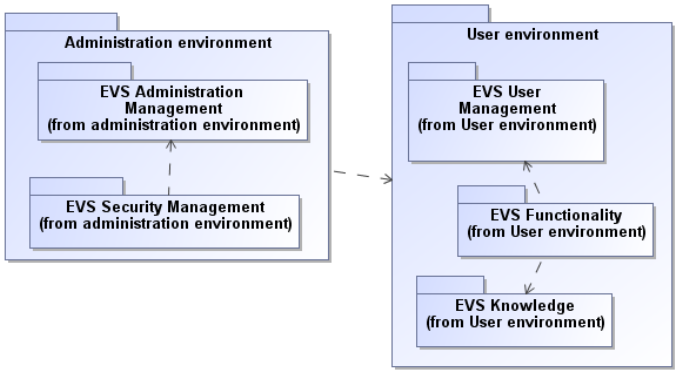
Use case diagram corresponding to the entire component library



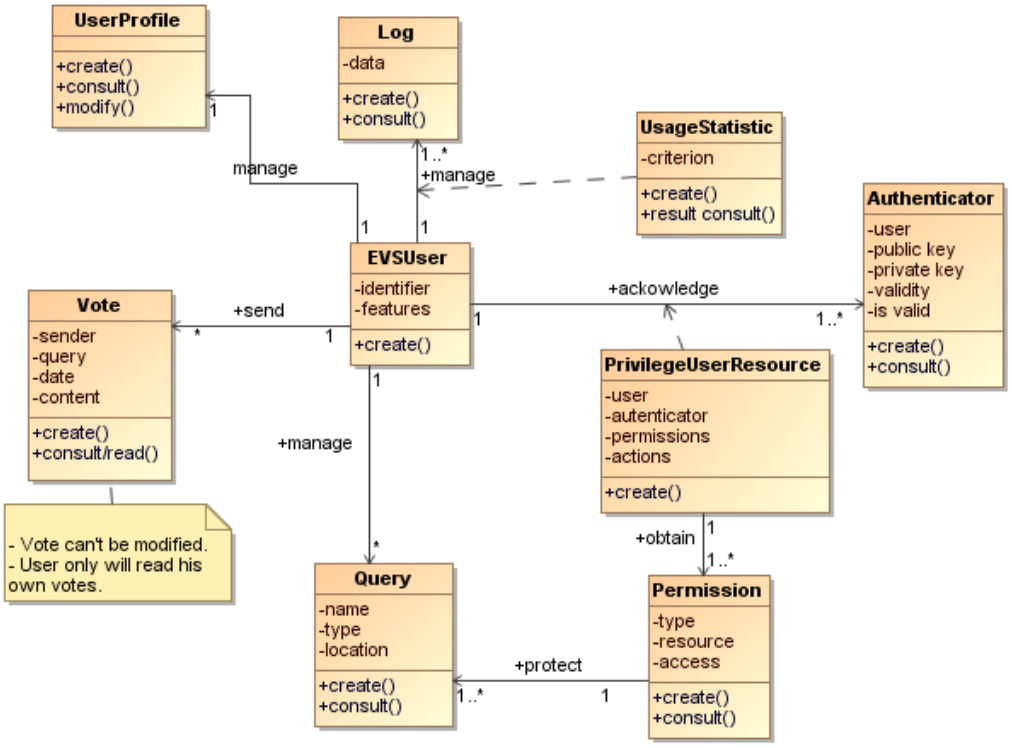
General objects identification which uses the use cases described before by a collaborative diagram:



Package diagram corresponding to main components and their dependency:



General object diagram of the model business:

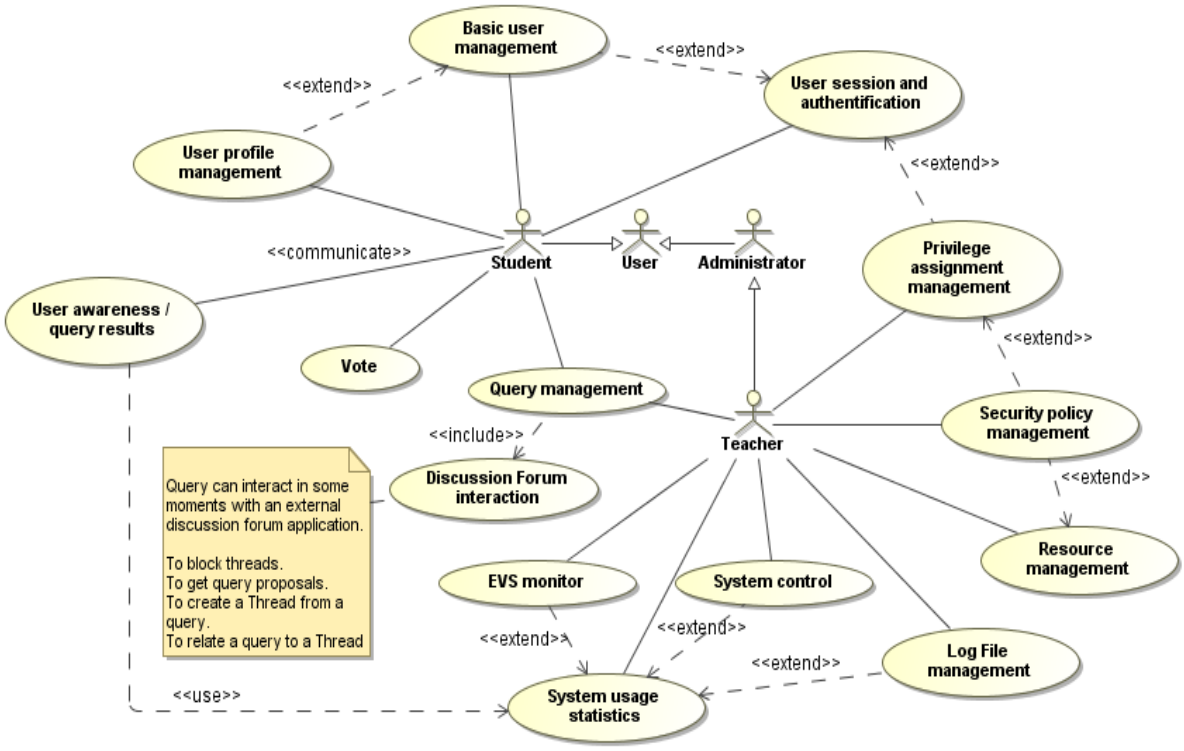


Identifying system's actors:

There are 3 actors: students (user, user coordinator), teachers (tutor) and the system administrator. They are not independent each other given that administrator and user share some roles such as user management, their authentication and their profile. Furthermore, the administrator can act as a system's user, so, the administrator represents a specialization of the user.

- **Student**. He has 7 roles. 4 common CSCL roles and 3 EVS specific roles.
  Common roles**:** to collaborate in their own management (consulting and modifying data), to carry out their authentication, to manage their own user profile and finally, to act as a group member and as a user-group.
  EVS roles**:** to manage queries, to vote and to get query results.
- **Teacher**. He has 7 roles. 5 common CSCL roles and 2 EVS specific roles.
  Common roles**:** to manage the users, authentication/session, user profile, privilege assignment, permission management.
  EVS roles**:** to manage queries and monitoring EVS for education purposes (mainly support to the consensus process and evaluation)
- **Administrator**: He has 7 common CSCL roles.
  Common roles**:** to manage the users, authentication/session, user profile, privilege assignment, permission management, system's events (logs) and to carry out statistics and other studies about the system.

Use case diagram with the general requirements of the entire library:

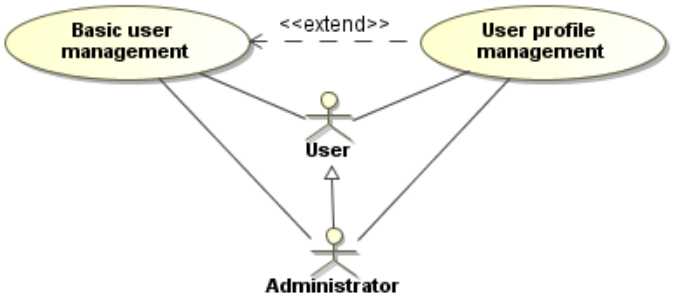


Library components description

The library is made up of 5 components which are independent and logically separated among them according to its general intern functionality:

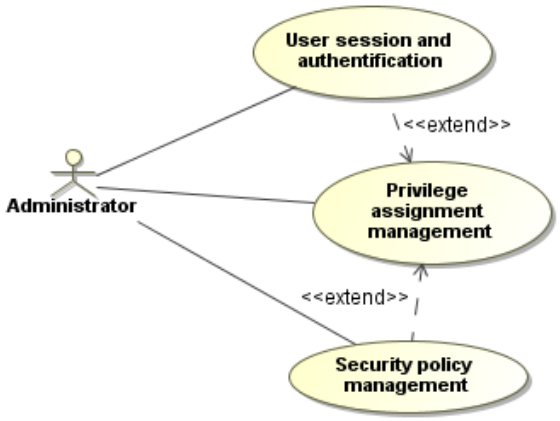
EVS user management component

Description: It will contain all logic related to the EVS system user management as a group coordinator, group member, user-group and system administrator. It will tackle both the user management above-mentioned (namely registration, drop out, modifications, consultations and joining/meeting a group/group members) and the user profile management. Both restricted user and administrator will interact with this component. Hence, this component will made up of the following use cases from the general requirement diagram: user basic management, user profile management.



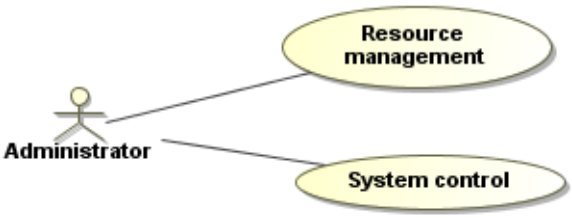
EVS security management component

Description: It will contain all the descriptions of the measures and the rules decided so as to protect the system's resources from either the intentional or unintentional system use by the user as well as the system's access from unauthorized users. Only the system's *User Basic user management, User profile management administrator* will be permitted to interact with this component. This is made up of the following use cases: authentication and user session, privilege assignment management, security policy management



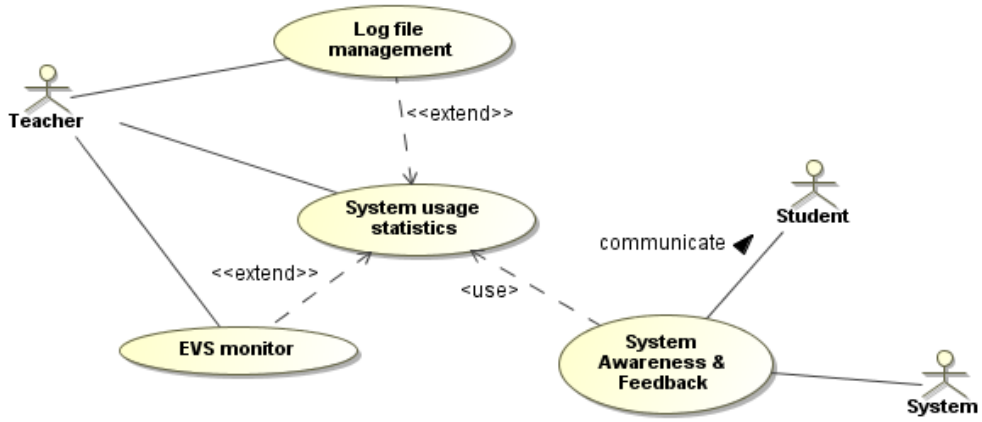
EVS administration management component

Description: It will contain those specific data (in some log files) and processes (statistical calculations) so as to basically carry out all system's control and maintenance to correctly administer the system and aiming to improve in terms of performance and security. On the other hand, it will manage the system's resources taking in account most of these resources will be managed by the users themselves while working within a group and hence these users will act as group administrators. Only the administrators are permitted to interact with this component. It is made up of the following use cases: resources management, system control.



EVS knowledge management component

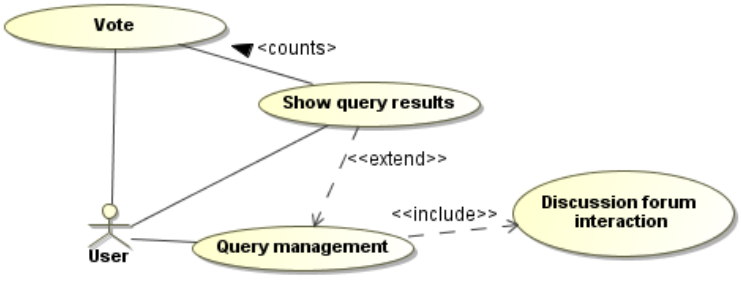
Description: This component will maintain and provide monitor data, awareness and feedback: log file management, system usage statistics, monitors EVS, system awareness & feedback.



EVS functionality component

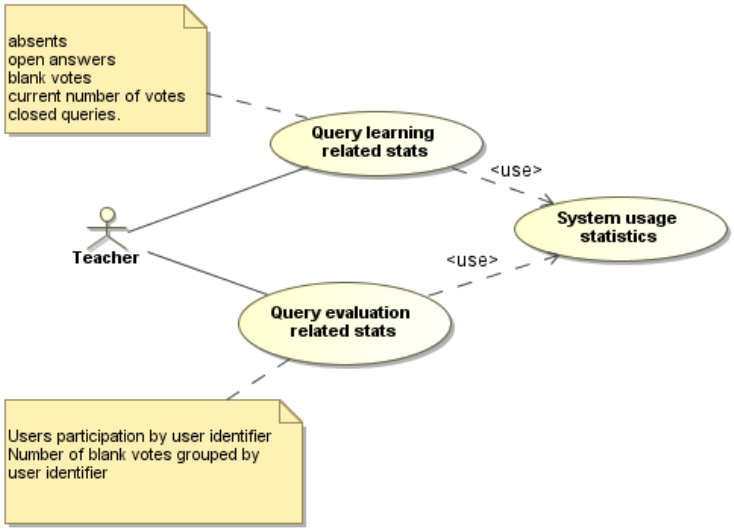
Description: It will form the basis of the EVS giving users the query management and vote capabilities. Query management includes interaction with external discussion forum. i.e. a query would block a DF thread during the vote process. Showing query results is the other major feature in this component.

It is made up of the following main use cases: query management, vote and showing query results.



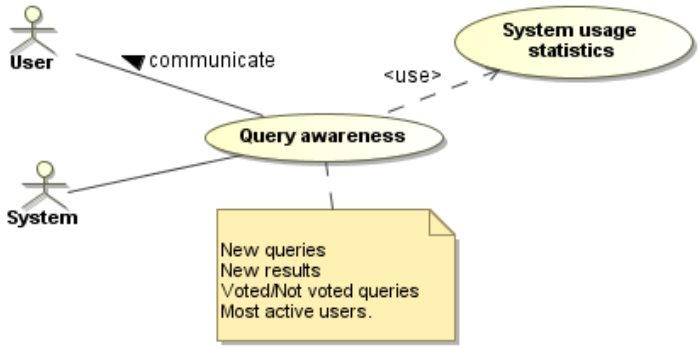
Use cases

EVS Monitor use case



General functionality summary: This shows statistics about query process state. Monitoring will give to the teacher data to discover lacks in the learning process and evaluate some bad attitudes.
Role within user's work: This is one of the main use cases in the teacher role which is carried out exclusively by teacher.
Actor: teacher.
Precondition: There must be some data as an information basis to make a statistical study. These data must be contained in a log file which has to exist in the database.
Postcondition: The planned statistic has been executed and the results have been showed. In case of the occurrence of problem during the statistic execution a self-explanatory message is issued indicating the failure reasons.
Description: A series of criteria will be created which ask log files data for our specific questions about the system and with it will be possible to extract valuable information to be analyzed later. It is also possible for the administrator to consults these results in different formats so as to make different reports and so to facilitate its analysis.
Observations: In order to improve discussion and consensus the teacher will get great indicators from these stats.

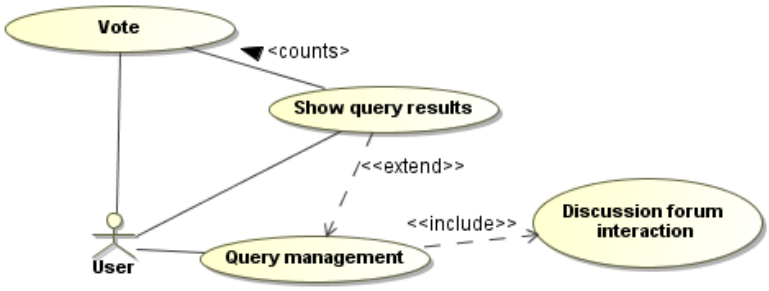
System awareness and feedback use case



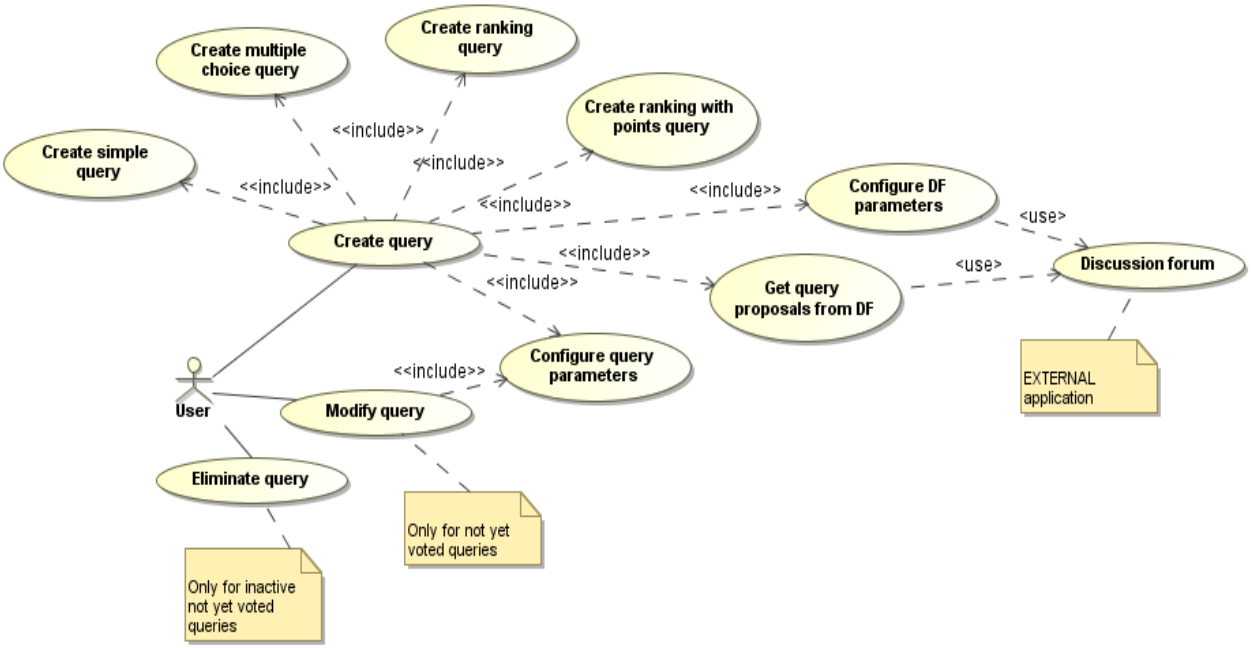
General functionality summary: this shows awareness and feedback statistics over query and communicates system awareness and feedback to the user.
Role within user's work: to get results is one of the main use cases in the student's role. As for query awareness it will be provided by the system itself.
Actor: EVS user (student, teacher).
Precondition: there must be some data as an information basis to make a statistical study.

These data must be contained in a log file which has to exist in the database.
Postcondition: the planned statistic has been executed and the results have been showed. In case of the occurrence of problem during the statistic execution a self-explanatory message is issued indicating the failure reasons.
Description: a series of criteria will get data for our specific questions about the system and with it will be possible to extract valuable information to be analyzed later. It is also possible for an EVS user to consult these results. System user will automatically generate awareness and show it to the EVS user.
Observations: results and awareness are made to help users over discussion and consensus process. Keep students motivated and actives is our other goal when we plan awareness criteria.

EVS functionality component requirements



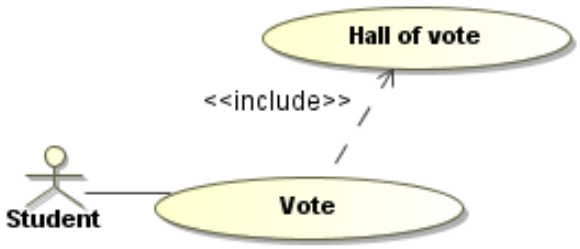
Use case: Query management



General functionality summary: This creates all four proposed kind of queries with all the parameters related to query and DF interaction.
Role within user's work: This is one of the main use cases in the EVS which is carried out by any EVS user.
Actor: EVS user (student, teacher).

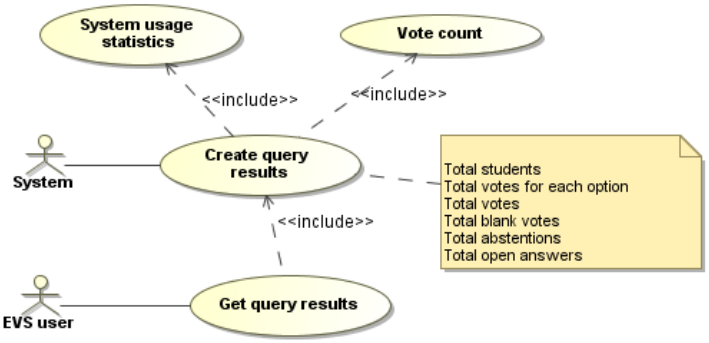
Precondition: Interaction with DF will be enabled to get DF interaction. Elsewhere only standalone –not DF related- features will be applicable.
Postcondition: The planned query has been posted and shared to other EVS users. In case of the occurrence of problem during the query creation a self-explanatory message is issued indicating the failure reasons. Awareness actions will be activated to announce the new query to the users.
Description: Create a query is the starting point of this project. We will create 4 kinds of queries and configure it in a different ways to manage different discussion scenarios. When it is possible some special features directly related to DF will be enabled and they will provide some interesting features like blocking threads during the vote process.
Observations: Queries only can be eliminate or modified if there is no votes in them. DF is an external application with EVS capabilities in mind and it can easily integrate our EVS functionality.

Use case: Vote



General functionality summary: This creates a register in the database with the student vote data. Vote will occur in a private space called *hall of vote* to keep user vote anonymous.
Role within user's work: This is one of the main use cases in the EVS which is carried out by any student. There is no real necessity to vote for teachers and they will not vote.
Actor: Student
Precondition: There is an open query to be voted and we have access to the *hall of vote*.
Postcondition: Vote is send and registered in the database. Points are counted and awareness data will be generated by the system. In case of the occurrence of problem during the query creation a self-explanatory message is issued indicating the failure reasons. If partial results can be showed an screen will appear with current query results.
Description: After selecting a query, the user goes to the hall of vote and sends a vote according to the selected voting system rules. Vote is registered and results are regenerated to refresh the data with the new vote.
Observations: Votes can't be modified or eliminated at all. A warning will be shown to confirm vote before to send it.

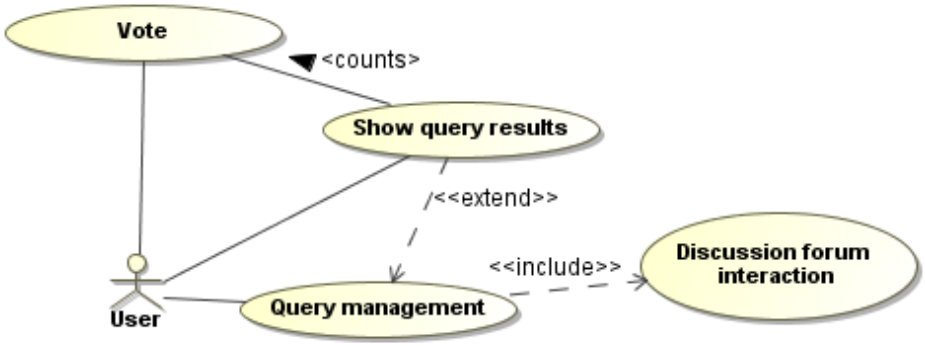
Use case: Show Query Results



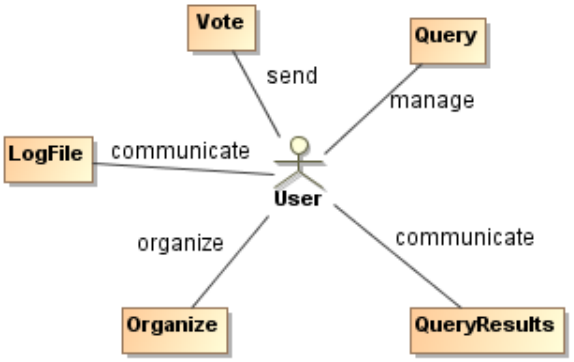
General functionality summary: this use case does a counting over different parameters from system usage statistics and from the registers with the student vote data.
Role within user's work: this is one of the main use cases in the EVS which is carried out by any EVS user.
Actor: EVS user (Student, Teacher)
Precondition: there is an open query with *show results before ending* enabled or a closed query.
Postcondition: vote counting and feedback and awareness data are collected by the System and showed in a friendly environment.
Description: after selecting a query, the user asks to get results. If the query is closed, data from results will be stored. If query is already open results will be calculated in each instantiation.
Observations: Teachers are authorized to see results ever, even if the *hide results before* is enabled.

EVS Functionality component specification

Use case diagram of the business model

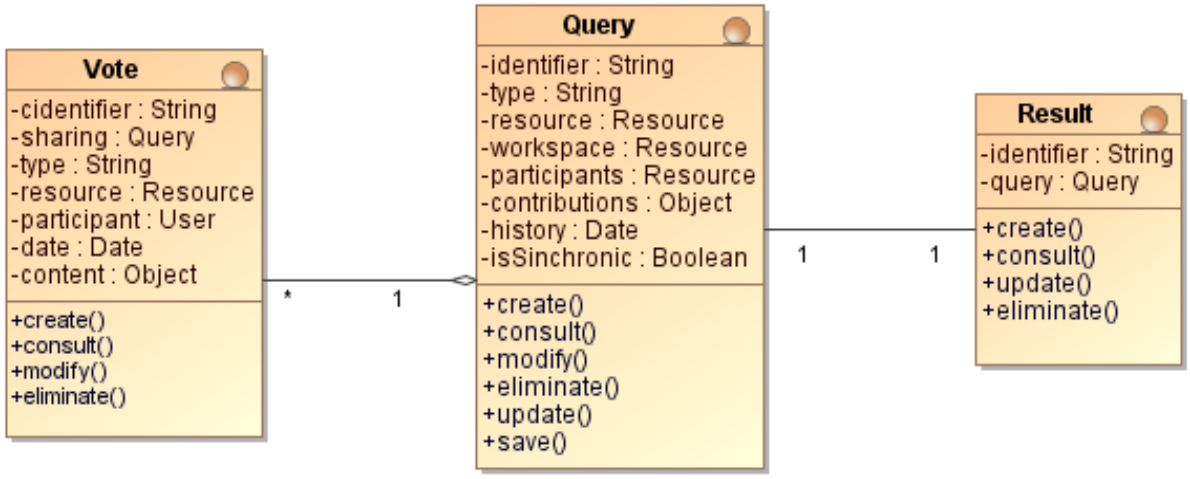


Main object identification



Package diagram

Main object diagram of the business model

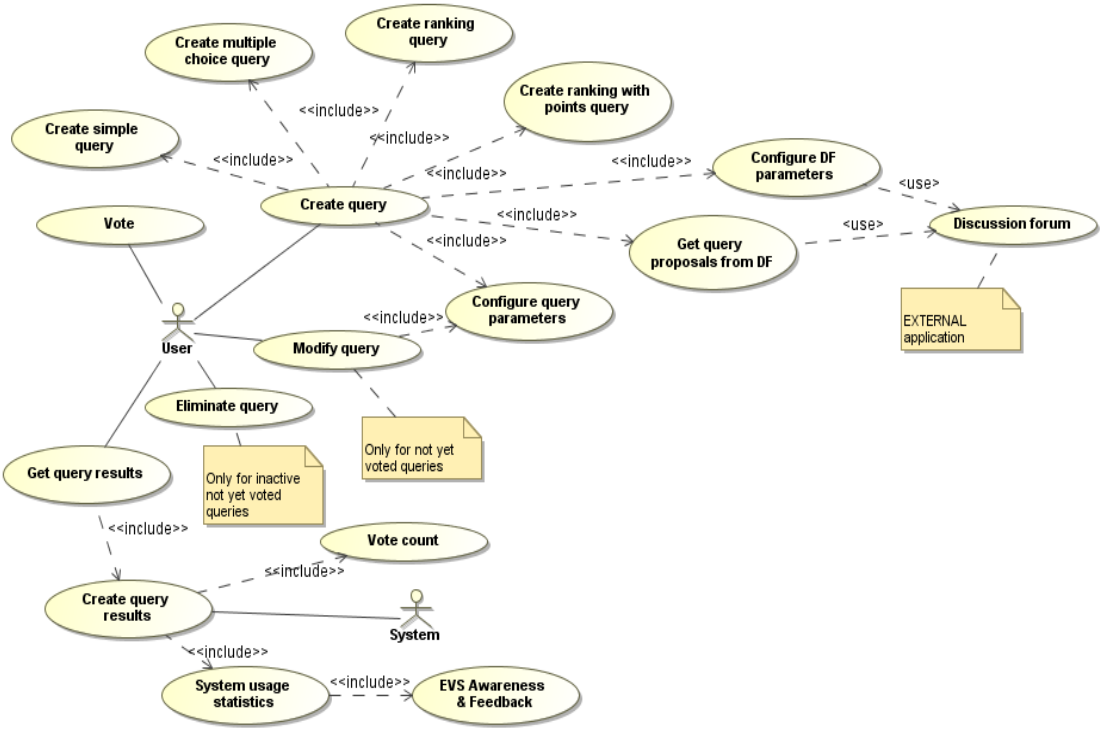


Study into requirements

Identifying system's actors:
There are two actors: Users (*Student and Teacher) and System.*
Even it can be problematic each user can create and manage queries. In fact, users will be the major query creators and voters in the application. We can imagine some context where the teacher will create a query, but for example, there is no needed scenario for teacher vote.

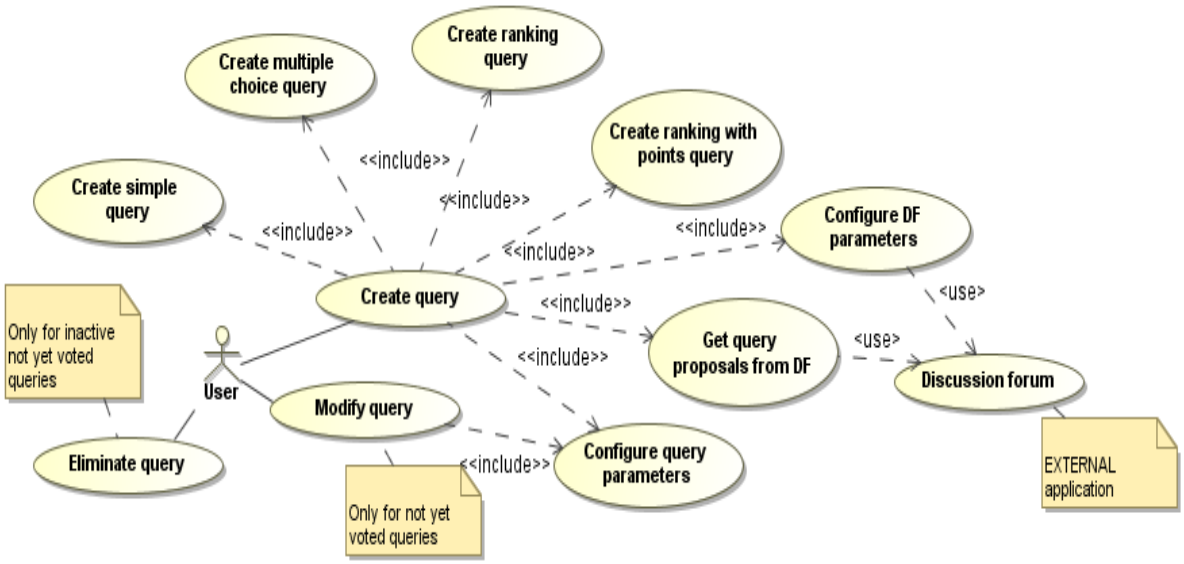
All roles are primary. Type of roles:
User. Has 4 roles: to create and manage a query, to send vote to queries, to transparently communicate actions to the system logs and to organize the queries giving starting and ending times and closing queries before the end date if it is necessary.
System. Has two roles. To keep user aware of what is going on in the system by providing him/her the available awareness and feedback information as well as calculating query results.

Use case diagram of the general requirements

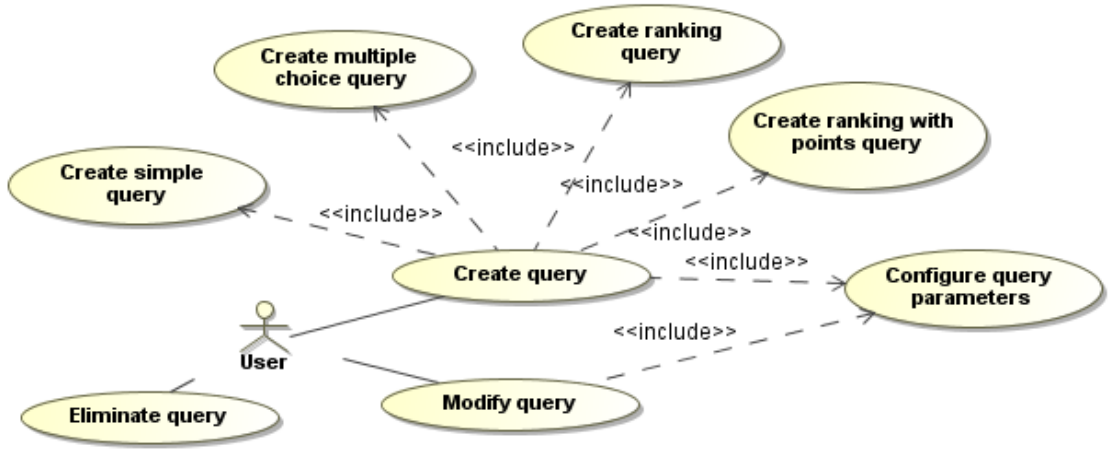


Subsystem requirements

General use case query management



Use case standalone query management



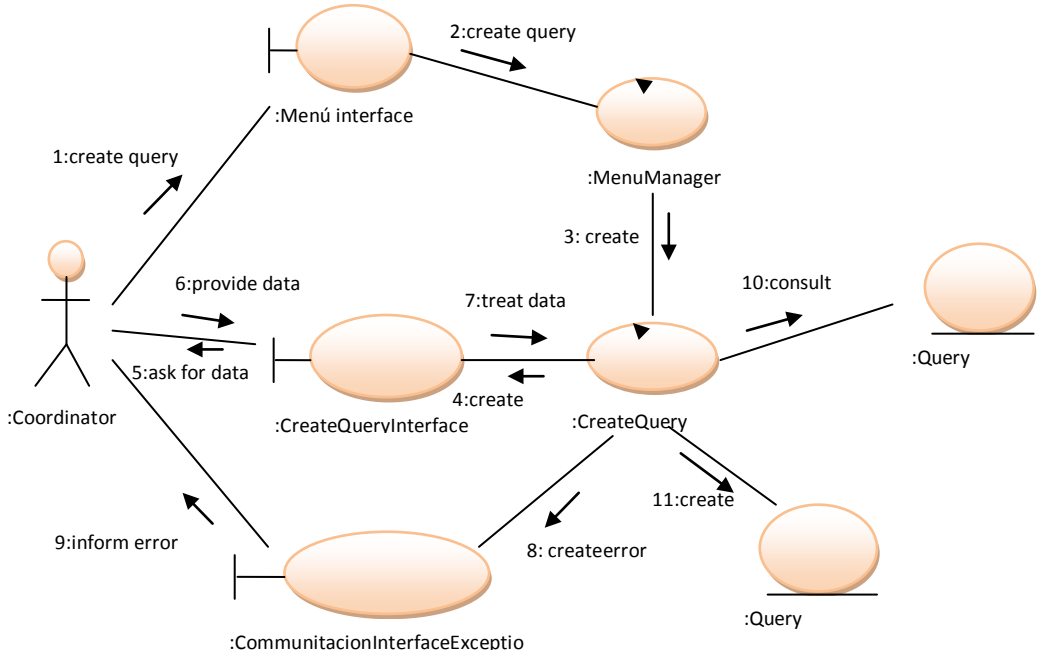
General functionality summary: the user will create a query of one of our four proposed type of queries. Queries can be modified or deleted before to be voted. A query will have some particular configuration like allowing blank vote or not.
Role within user work: this is one of the main use cases in the system user and represents the cornerstone to make possible the consensus in an EVS environment.
Actor: user (Student, Teacher).
Precondition: In case of creation, the query does not exist in the database. The query must exist for the rest of functions.
Postcondition: In case of creation, the query is incorporated in the database. In case of modification, deletion or consultation of a query, either the operation is successful or not a self-explanatory communication is issued.
Description: In case of creation, the query IDENTIFIER is generated and the user introduces his/her questions in the system persistence records.
Observations: All the message exchange between users generates events (e.g. a user reads a query; a discussion is active, etc.). Al these events will be collected and managed by the *EVS*

Knowledge management component and will form the awareness information which is immediately broadcasted to system users so as to make possible for others to be aware of what is going on in the system and above all in the workspace where the learning group is developing.

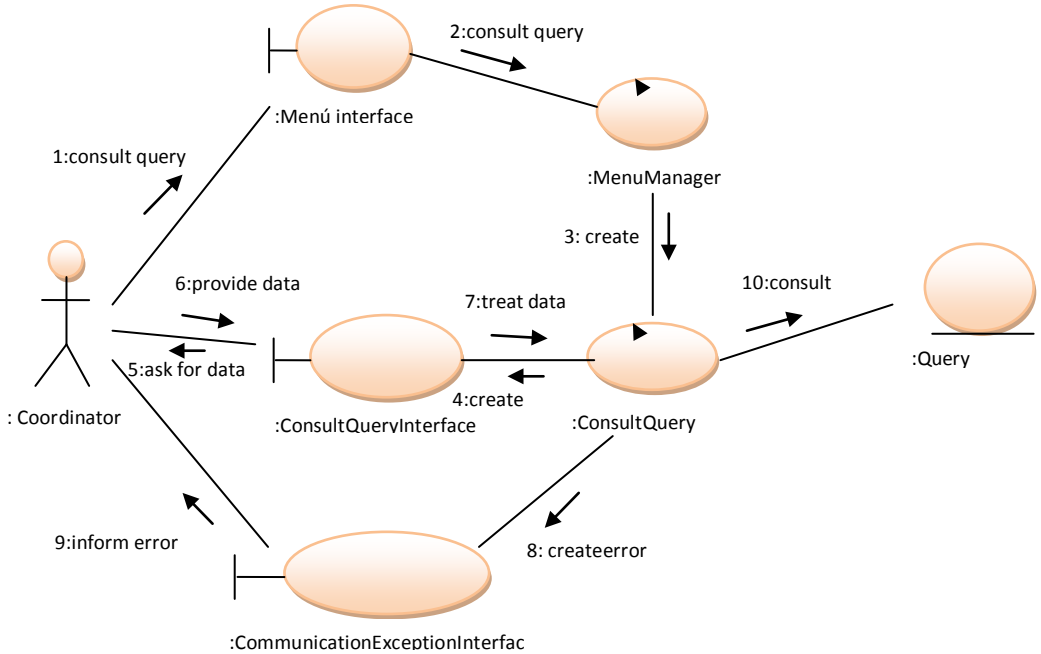
Collaboration diagram

Note: due to space limitations we are showing only few most important collaboration diagrams.

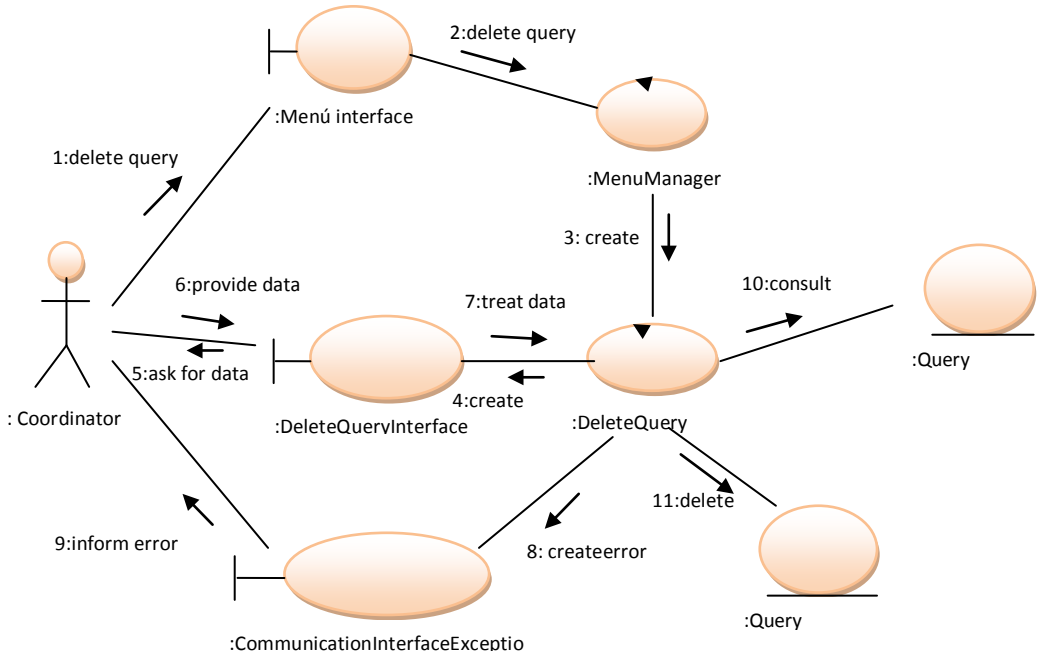
Create query



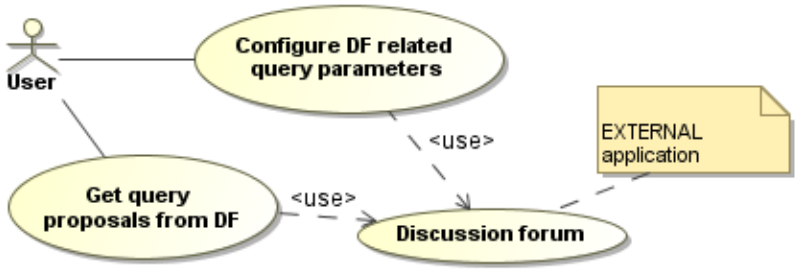
Consult query



Delete query



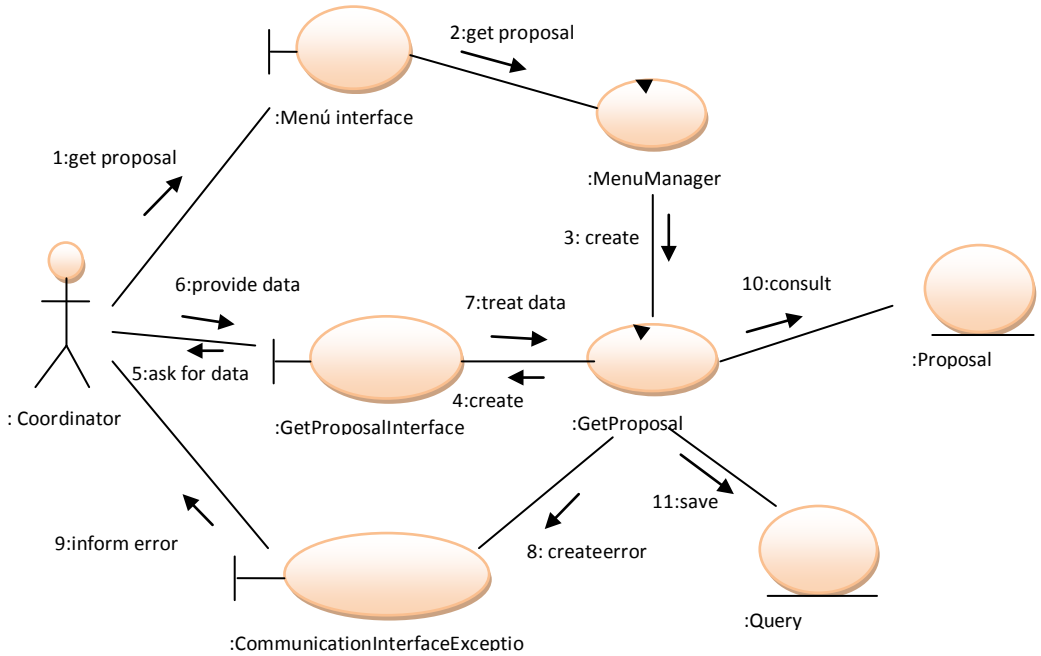
Use case DF Interaction



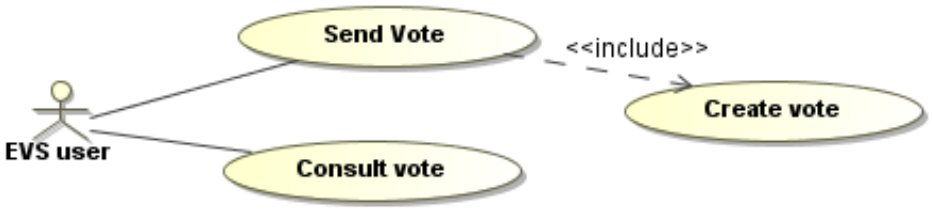
General functionality summary: this configures parameters related to the DF and manages the data collected from the DF as query proposals. This use case is only in case our EVS is integrated with a DF, in other cases there is no effect from this configuration.
If we are connected to a discussion forum, we will get query proposals from the current thread to have a starting point even if finally we don't use any of the proposals.
Role within user work: this is an important but secondary use case in the system but it represents the cornerstone to make possible the discussion and consensus supported by an EVS environment.
Actor: user (Student/Teacher).
Precondition: Our EVS will be in a DF environment and some interaction between the two applications will be enabled.
Postcondition: The selected changes from our EVS will affect the DF. In case of error a message, either the operation is successful or a self-explanatory communication is issued indicating the failure reasons.
Description: When we are interacting with a DF, we can use some DF functionality from our EVS application. In the other hand, the queries will be assigned to a single discussion forum and the pros and cons about the possible query answers will be discussed here.
Observations: All the message exchange between users (or system user) generates events (e.g. a user reads a query; a post used send a part of conversation to a query room, etc.). Al these events will be collected and managed by the *EVS Knowledge management* component and will form the awareness information which is immediately broadcasted to system users so as to make possible for others to be aware of what is going on in the system and above all in the

workspace where the learning group is developing. On the other hand, awareness is very important in this context by showing others' availability for communication.

Get query proposals



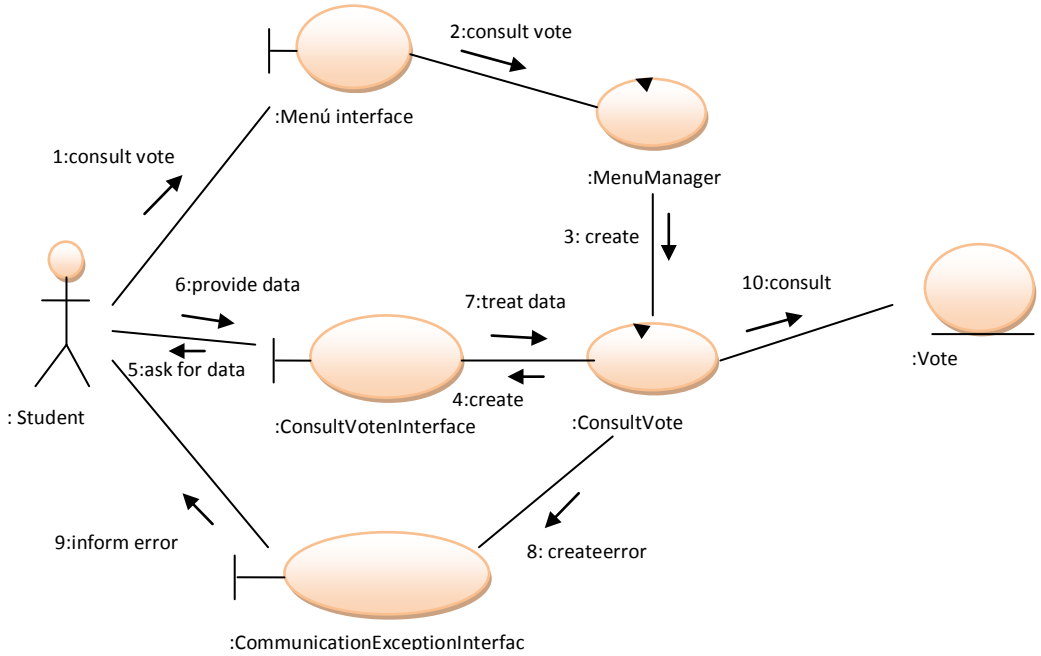
Use case vote



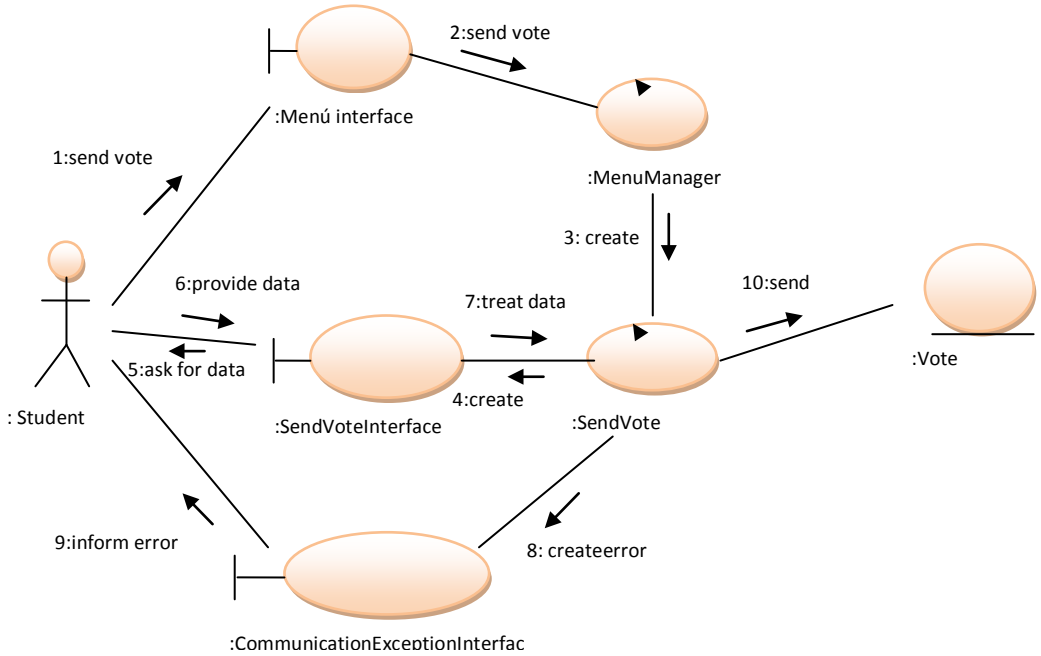
General functionality summary: create, consults and send a vote. Vote is an object variable in terms of what kind of voting system or configuration we have selected. Not all votes are the same and counting will have to deal with this particularity.
Role within user work: this is one of the main use cases in the system. Create a query is the first part but, get it voted is the second and more important thing in an EVS application.
Actor: user.
Precondition: In case of creation, the vote does not exist in the database. The vote must exist for the rest of functions.
Postcondition: In case of creation, the vote is incorporated in the database. Vote can be consulted, but not modified and not deleted if it is send.
Description: In case of creation, the vote IDENTIFIER is generated and the user introduces his/her reference to the persistence records related to the query.
Observations: All the message exchange between users generates events (e.g. a user vote a query). Al these events will be collected and managed by the *EVS Knowledge management* component and will form the awareness information which is immediately broadcasted to system users so as to make possible for others to be aware of what is going on in the system and above all in the workspace where the learning group is developing.

Collaboration diagram

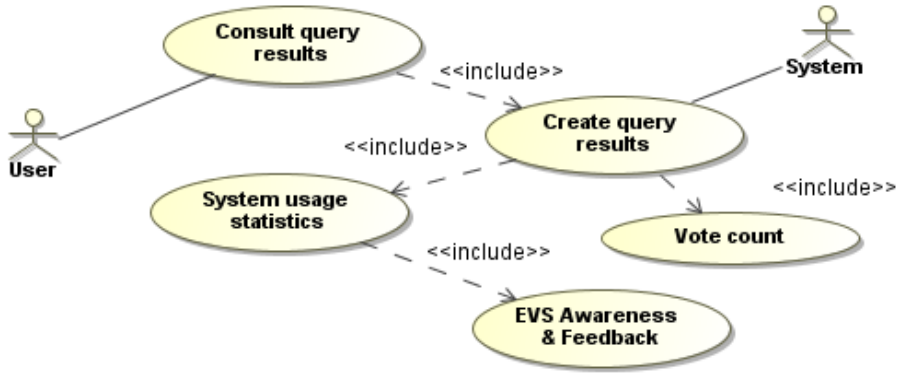
Consult vote



Send vote



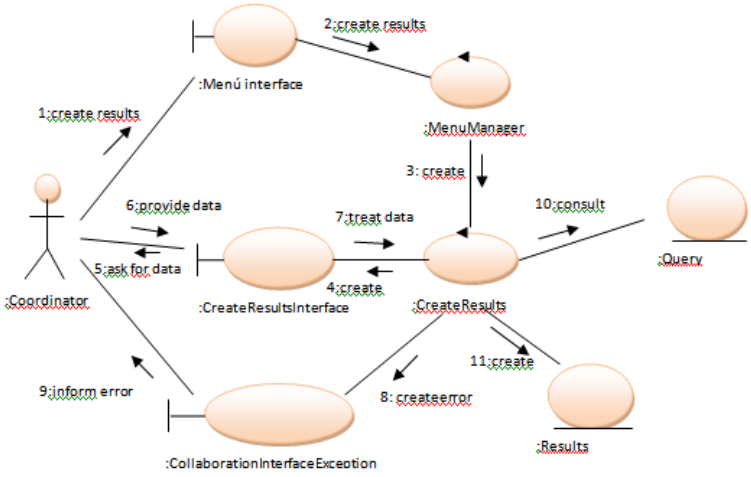
General use case query results



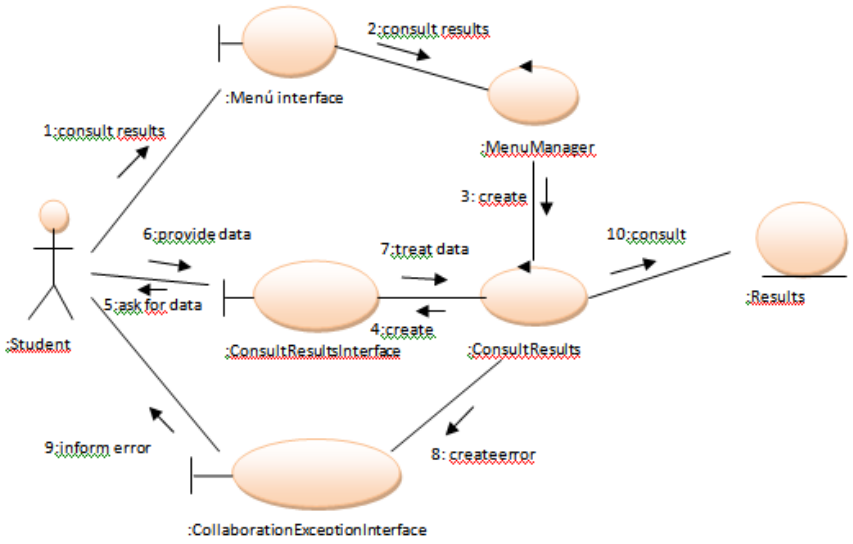
General functionality summary: This functionality attempts to present useful query results to users. Those results are not simply vote counting, but also context information about the query. Keep in mind we are looking for consensus and not for a winner o for democracy. Some data may be important like % of blank votes, % of abstents, % of votes in the first days, % of open data votes and so. We can extract information from that. It is the query well formed? It is the query proposing something interesting the students? The students are agree or not with the proposed answers? and so.
Major stats and results will be created by the system user. Students will not create their own query results, they will only get them from the system. Awareness, feedback and vote counting will collaborate to get complete useful results.
Role within user's work: This is one of the main user use cases.
Actor: User (Students, Teachers).
Precondition: The query exists and it has enough information to present some results
Postcondition: Results are created and showed to the users in a report.
Description: Data for the report is created by the system from awareness, feedback and vote counting process. The report will be focused in consensus information. Stats to be applied to that report are created in the EVS knowledge component.
Observations: When interacting with DF report can be sent to the thread like a post to discuss the results.

Collaboration diagram

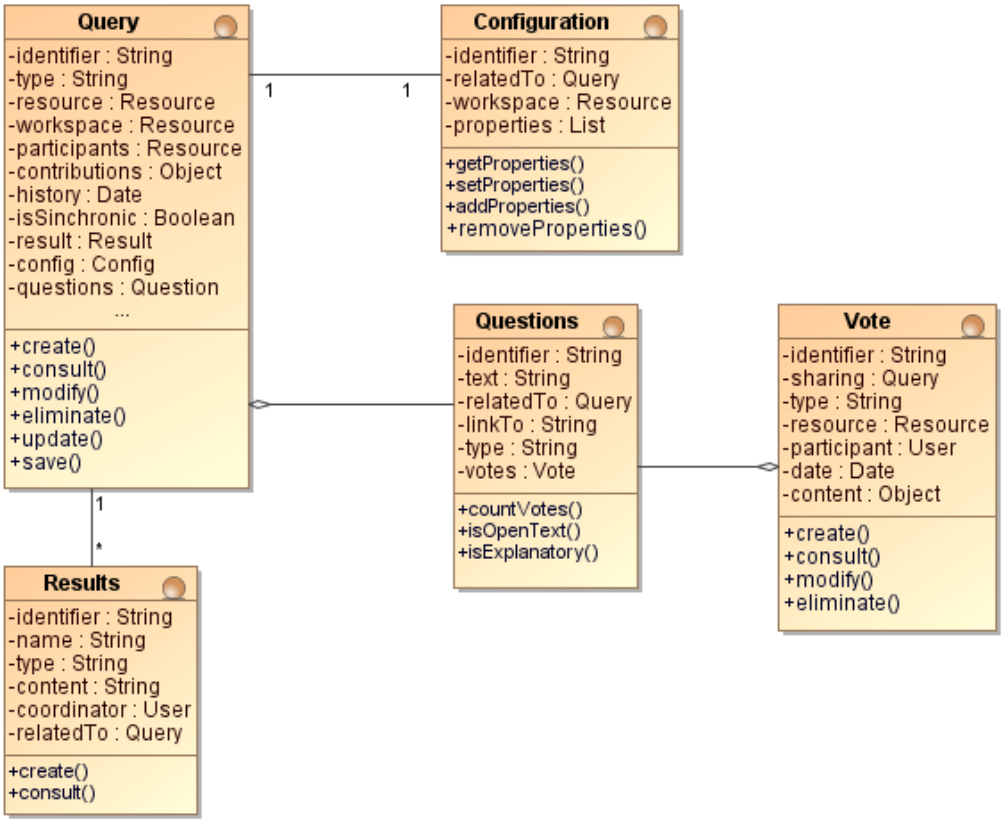
Create results



Consult results



Analysis static diagram



Chapter 2. Design

EVS Knowledge Design

EVS Knowledge Management Design

Note: due to space restrictions we are describing only the core components of our EVS with a little detail. See our CLPL references [1][2] for a detailed explanation of the reutilized components.

Introduction

In designing a computer supported collaborative learning EVS application it is necessary to correctly organize and administer both the resources offered by the system and the users accessing these resources. Furthermore, we must take into consideration that user interaction is crucial in any learning collaborative environment to make it possible for groups of students to communicate with each other and to accomplish common objectives (e.g. a collaborative classroom activity).

All of this user-resource and user-user interaction generates events or logs which are collected in log files and represent the information basis for the performance of statistical processes addressed at obtaining a useful knowledge of the system. This will facilitate the collaborative learning process by keeping users aware of what is going on in the system (e.g. others' contributions, the new queries created, etc.) and controlling the general users' behavior in order to provide support to them. Although the user interaction is the most important point to be managed in general CSCL applications, it is also important to be able to monitor and control the performance and general functioning of the system. This will enable the administrator to continuously track the critical parts of the system and act if necessary. Furthermore, this adds an implicit security layer to that explicitly provided by the *EVS Security* component (e.g. controlling the users' habits making it possible to detect fraudulent use of the system by unauthorized users).

The *EVS Knowledge* component will manage all the specific and large user events in order to record user interaction data as information which is crucial for the correct control and administration of the EVS systems. The final objective of this component is to extract valuable information from the events processed for later analysis with the aim of revealing useful knowledge.

This component will manage as well the voting results and other counting data from the EVS process.

We will define stats/logs/criterions to collect specific data over the following:

About the query: count students in the query workspace, count votes for each query option, count votes for a query, count blank votes for a query, count abstentions for a query, count open answers for a query, log first student vote & last student vote days for a query.

About the student: log student vote history, counting abstentions, counting open answers, counting blank votes, counting days to vote after the query started.

This component will also care about *awareness* and *feedback*. We deliver information back to the user in order to keep it aware and engaged. Appropiate stats/logs/criterions will be created in order to provide support to the following requirements:

Awareness: new query alert, final query results alert, closed query, not yet voted queries and days remaining to close the query if applicable.

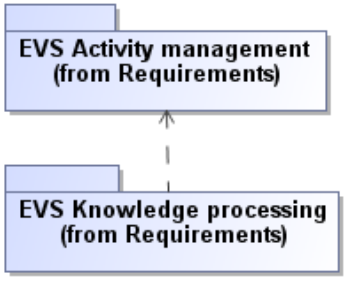
Feedback: show self vote options for each query, self activity, other's activity.

Note: see *Chapter 1 - Specification* for a justification about why we select to keep trace over this data.

The component is made up of the following subsystems:
• **EVS Activity management** subsystem that manages the log files making up of all the events occurring in the system during a period of time. This represents the source of information for the creation of the pertinent statistics.

• **EVS Knowledge processing** subsystem carrying out the management and maintenance of statistical studies through the generated events stored in log files. These studies, and their posterior analysis, will provide the necessary knowledge to control and facilitate the collaborative learning process and improve the general performance of the system.

Graphical representation of the subsystems:



The **user interface**, through which it is possible to manage and maintain log files, criteria and statistics, will be generically focused so as to make particularization in graphic and text modes possible. Even though the user interface will usually be in graphic mode, it is necessary to consider this abstraction in order to make the logical part of the application independent from the specific design of the graphic user interface and its posterior updating. In doing so, we guarantee a minimum level of consistency between the presentation and logical layers in the specific application.

EVS Knowledge processing subsystem

Introduction

The process of correctly managing the events generated in any computer system in general and a CSCL application in particular during its functioning involves several steps: the collection of information, the analysis, communication and presentation. The **EVS Knowledge Processing** subsystem carries out the second step by analyzing the information collected in the log files by the *EVS Activity Management* subsystem as seen before.

The S*tatistics* entity will provide us with the potential for extracting useful information from the log files which in turn will give us detailed knowledge of those parts of the system or the users' behavior in which we are interested. To that end, a statistics will contain the log file data as the information basis along with an EVS criterion with the aim of extracting the desired information from the log files in a collaborative learning environment. This *EVSCriterion* entity can be multiple and can affect several attributes or fields of the log files so as to make the construction of complex consultations possible which provide us with the necessary detailed knowledge.

A hierarchy of criteria is designed to classify and predefine the most usual requests for information in CSCL environments (e.g. "how many users were in the system during a certain period of time?", "Which users vote which query?" etc.) and thus making it possible to reuse

them in as many statistics as possible. Thereby, the root of this hierarchy is represented by the above-mentioned *EVSCriterion* entity as the most abstract form possible. At the first level in the hierarchy, we split all the possible consultations into four general aspects or options, namely system, users, objects and spaces. At a second level, for each aspect there will be the specific type of consultation, so, for example, through the *system* option of consultation, we can ask for the absolute data about the users, objects and sessions related to the system (such as the number of users or the sessions in the system). This hierarchy is left open with the aim of making it possible for the developers to specify their own consultation criteria.

These predefined criteria are driven by the specific EVS resources concerned (such as time information from the system's clock for consultations in which time is involved, a query, a group workspace, and so on). These resources are aggregated to the predefined criteria as parameters in order to dynamically provide the specific resource concerned in the criterion if necessary and, thus, these predefined criteria are parameterized. The basic operations with the criteria (i.e. creation, consultation, modification and elimination) will exclusively be carried out by the system administrator who is the only user with enough privileges to do so and is responsible for the correct and efficient functioning of the system. The criteria management realizes the *Inquiry management* use case seen before during the analysis of this subsystem.

Once a statistics is created, it is executed in order to generate some *results* which will be stored in the *Statistics* entity as a field and will be able to be consulted both during the execution of statistics and for posterior reports. The result of a statistics will be offered by different formats of representation such as text, graphs (bar and pie charts, etc.), html and others where available. These results will be the basis to elaborate full reports which will provide the necessary documentation justifying the conclusions reached.

In order to take advantage of the current highly powerful statistics systems on the market, this subsystem itself does not support any statistical execution process but rather exploits these popular statistical packages as external tools. To this end, a generic interface is provided to make it possible both to export our statistics format to most of these tools and to import the results obtained.

This open architecture design ensures great flexibility which allows the exportation methods to be totally automated by scripts. Furthermore, the data files can be fully customizable in the most popular statistical file formats (such as .XLS, .DBF, .WKS, ASCII and so on). This allows the data to be used easily in other statistical packages.

We are considering two kinds of statistics:

- One kind of statistics is complex and will compromise the system performance. This kind of statistics will not be generated on-demand and will be restricted in creation, modification and elimination to the EVS Administrator or the EVS Tutor. Only consultation will be granted to general users with their restrictions.

- The other kind of statistics will be conducted by simple lightweight operations without major inconvenient for the system performance. This kind of statistics will be granted to any EVS User, included EVS Students, without restrictions about creation, modification or consultation. We are thinking about vote counting, query results or similar lightweight statistics.

With regards to the basic data managed and maintained in each statistics, there will be an identifier, the creation date, a text description explaining the objectives, the criteria involved, the log files from which the information basis is obtained and the results. The statistics

management realizes the *Knowledge revelation* use case seen in the CLPL analysis of this subsystem.
The **user interface**, through which it is possible to manage and maintain both the statistics and their criteria, will also be focused generically so as to make its particularization in graphic and text modes possible.

With regards to **persistence**, this is also generic and permits particularization in both ordinary text files and the specific DBMS used during the particularization.

CLPL Platform reutilization and new EVS classes

EVS Knowledge processing subsystem

- Reused classes from the SystemControlManagement component which can be found in the gpl.control package of the General Purpose Library (GPL):

- Reused business classes (also from EVSUsersManagement component, EVSAdministration component and EVSActivityManagement subsystem): EVSUser, Identity, Interface, ErrorInterface, ProfileMenuInterface, WarningInterface, InterfaceException, ProfileMenuManagement, GenericDiskManager, ProfileDiskManager, ProfileException, GenericDiskManagerException, MenuException, Exception. PerformanceControlDiskManager, PerformanceControlMenuManager, ResultConsultation, ResultConsultationInterface, Statistics, HistoryStatistics, StatisticsConsultation, StatisticsConsultationInterface, StatisticsCreation, StatisticsCreationInterface, StatisticsElimination, StatisticsEliminationInterface, StatisticsExecution, StatisticsExecutionInterface, StatisticsInterface, StatisticsManagement, StatisticsModification, StatisticsModificationInterface, Criterion, LogElement, EVSResource, EVSSpaceResource, EVSUserResource, EVSObjectResource.

- Reused auxiliar classes: Object, String, Integer, Float, Character, Boolean, Date, Container, Iterator, SortedContainer, StringException, IntegerException, CharacterException, FloatException, BooleanException.

- New classes for this subsystem: EVSStatistics, EVSCriterion, EVSStatisticsCreation, EVSStatisticsConsultation, EVSStatisticsModification, EVSStatisticsElimination, EVSStatisticsExecution, EVSStatisticsConsultResults, EVSCriterionManagement, EVSCriterionCreation, EVSCriterionConsultation, EVSCriterionModification, EVSCriterionElimination, EVSCriterionInterface, CreateEVSCriterionInterface, ConsultEVSCriterionInterface, ModifyEVSCriterionInterface, EliminateEVSCriterionInterface, CriterionException, CriterionDiskManager, EVSCriterion, UserCriterion, SystemCriterion, ObjectCriterion, SpaceCriterion, UserData, UserSessions, UserEvents, SystemUsers, SystemObjects, SystemSessions, ObjectActions, ObjectUsers, ObjectAccess, SpaceDocuments, SpaceObjects, SpaceUsers, EVSTimeResource, BlankVoteCriterion, AbstentionCriterion, VoteCountLog, OpenAnswerLog, VoteCountStatistics, OpenAnswersStatistics.

Observations: all our new classes are created upon the CLPL base classes. At the moment of writing the design we don't have any specific implementation in mind and it's important to design our system extending the base system instead of directly using the CLPL classes in order to keep independence from further CLPL platform modifications.

Due to document space limitations we are showing here only a couple of CRC cards as an example of what the extension of the base system it's done.

CRC

Preliminary notes:

- The collaborations of the subclasses are the same as those of the super classes (besides others possible) and they are not shown. Even so, it has been considered the subclass as a compound class though it only inherits the collaborations.
- By default, all classes inherit from Object.
- The class name in italics means abstract class as some of their methods are abstract too.
- The *extends* keyword means the current class inherits the properties of a super class. On the other hand, *implements* keyword means the affected class is specific enough to definite the abstract methods inherited from a abstract super class.
- The visibility properties are:: + means public visibility (for everyone); - means private visibility (for the same class only); # means protected visibility (for the same class and its subclasses).

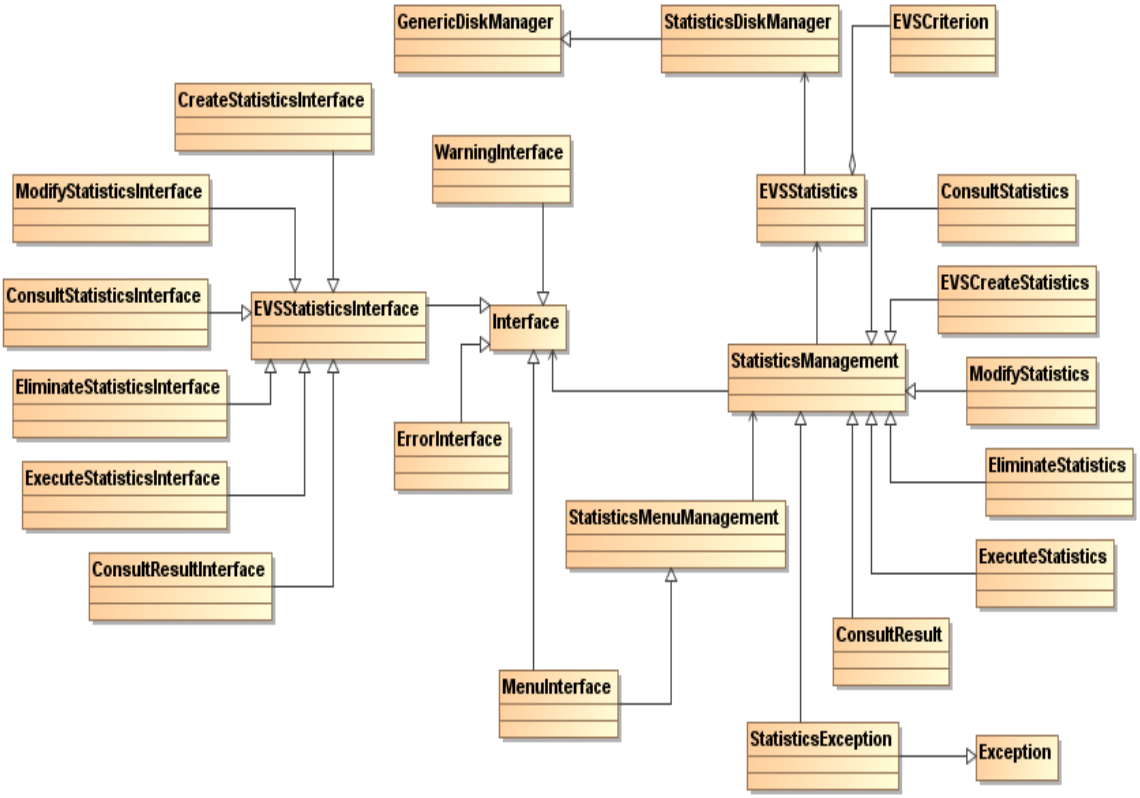
Class	**EVSCriterion extends Criterion**
Class description	Create a criterion in a EVS environment through the selection of certain attributes of a log file
Class type	Property: main class
Class features	Concrete, compound, persistent
Responsabilitats	**Col·laboracions**
Create a criterion in an EVS environment	String, EVSLogFile, SortedContainer
EVSCriterion extends Criterion	
-logFiles: SortedContainer //a LogFile-element container to know the attributes to be selected.	

//stores an empty EVS criterion with an identifier.
+EVSCriterion(pIdentifier: String)
//stores a EVS criterion with an identifier, a container of log files, the attributes of these log
//files to be selected and a text description
+EVSCriterion(pIdentifier: String, pLogFiles: SortedContainer, pAttributes: SortedContainer, pDescription: String)
//get the log files where the attributes to be selected are.
+getLogFiles(): SortedContainer
//set the log files where the attributes to be selected are.
+setLogFiles(pLogFiles: SortedContainer)	

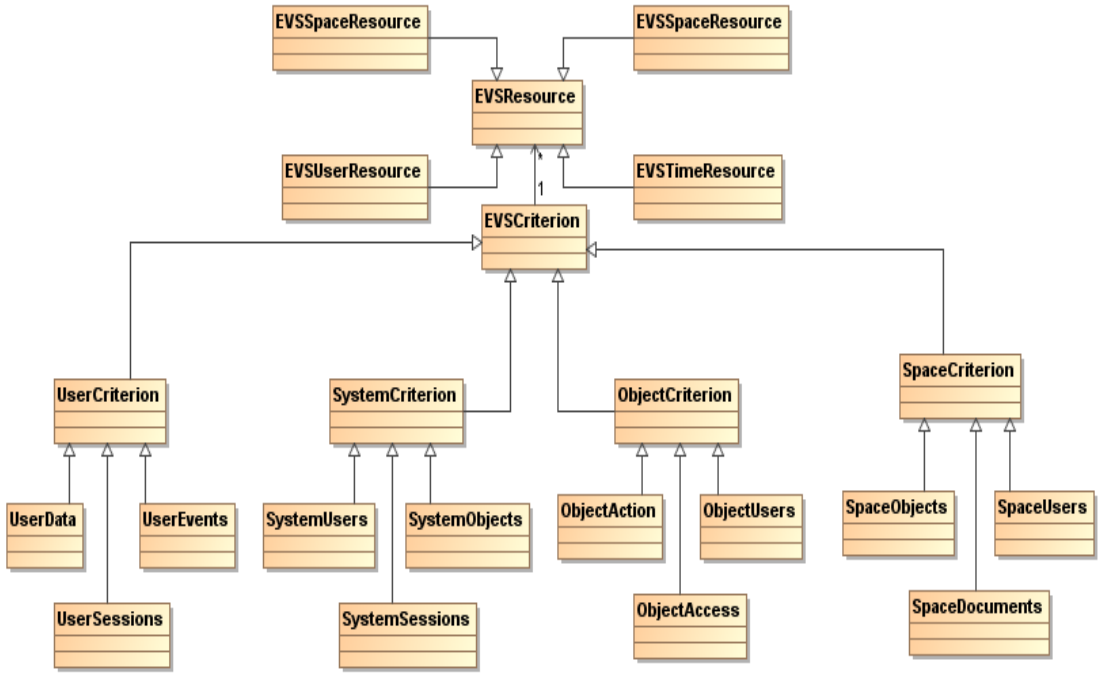
Class	***EVSStatisticsCreation* extends *StatisticsCreation***
Class description	Carries out the logic part or the statistics creation
Class type	Property: main class
Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Receives and send information to the statistics administer. Get and send information to the persistent objects	StatisticsException, StatisticsCreationInterface EVSLogFile, EVSCriterion
EVSStatisticsCreation* extends *StatisticsCreation	
#sci: StatisticsCreationInterface **#evsStatistics: EVSStatistics**	
//creates **+EVSStatisticsCreation()**	

Class diagrams

EVSStatistics class diagram

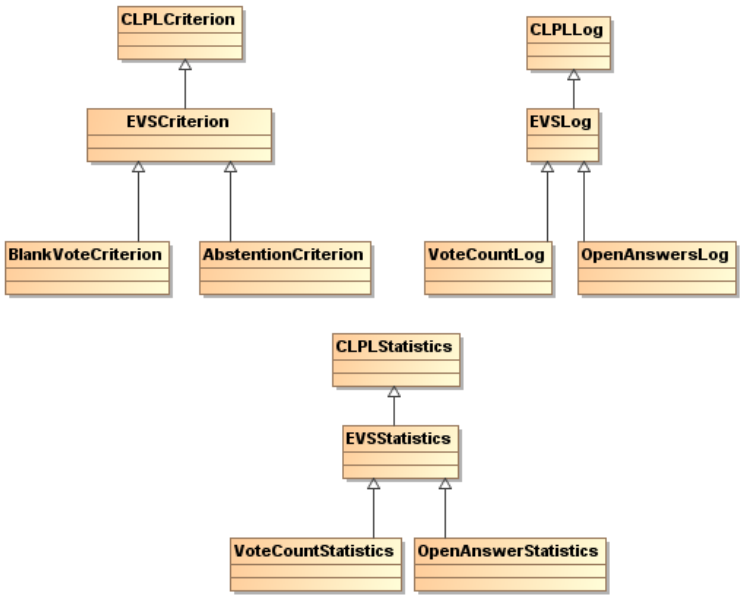


EVSCriterion hierarchy class diagram



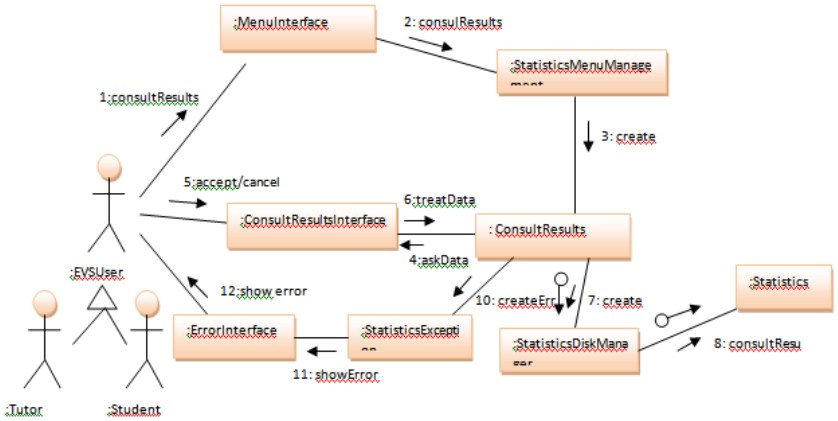
Vote system extensions over the core reused CLPL components.

That's barely a showcase on how we'll interact with the framework from our concrete purposes. Further classes will be added when needed.



Collaboration diagrams

Consult EVS statistics results:



Persistence design

Introduction

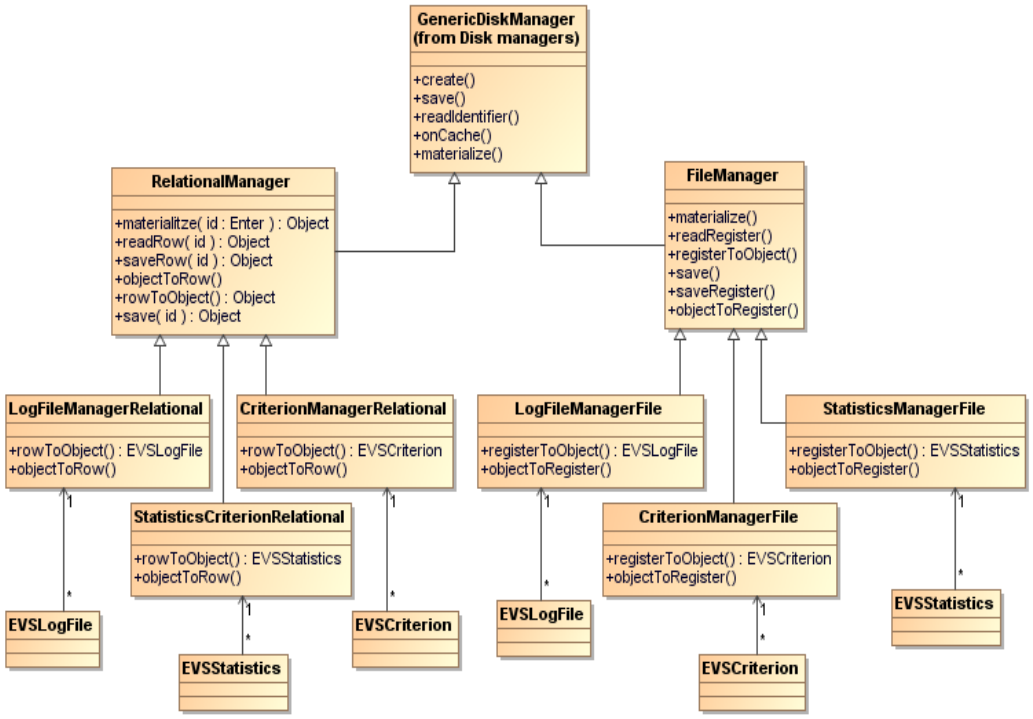
The *generic programming* paradigm will be applied to the design of the persistence as those sessions with lifespan exceeding that of the process that created them requires permanent storage and a management of the persistence which is independent from the management of the mentioned storage.

Furthermore, the persistence implementation can be done by both database (relational and object oriented) and ordinary text files.

In order to achieve this objective, we will base the persistence management in the *disk manager* which permits independence as it enables access database or text file, so, letting the *disk manager* communicate the specific DBMS or file system. Hence, if a specific data manager is substituted by another, the only thing which needs changing will be the *manager disk* that maintains the object persistence in the same form.

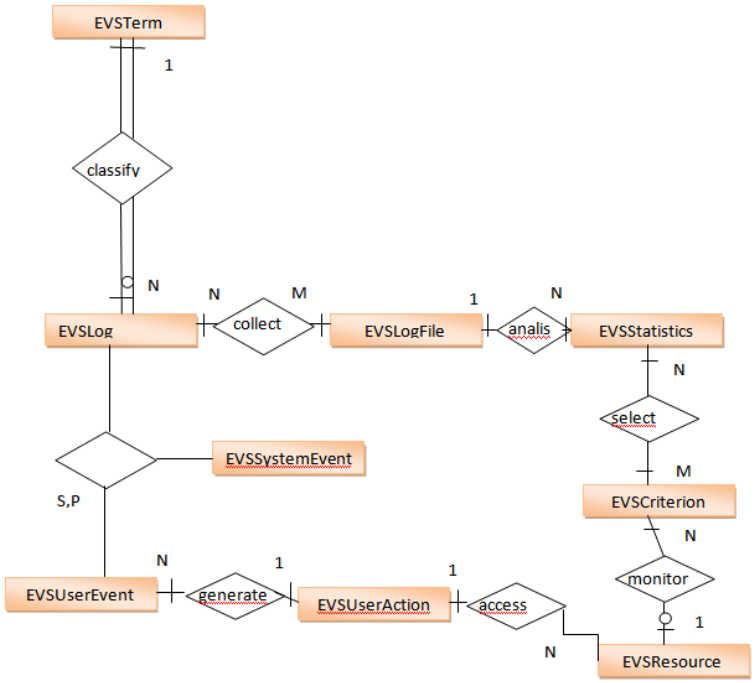
A generic data manager will be created of those operations which are common to both the database and file models. For each one of the two technologies a specific manager disk will contain the specific operations required to access data according to the current storage model.

The complete disk manager hierarchy is as follows:



Conceptual and logic design

Conceptual design (ER Diagram)



Attribute description

*The outlined attributes represent the primary keys

EVSSTATISTICS

evsstatistics-code, log-file-code, date, description, results

EVSCRITERION
evscriterion-code, description, attributes, evsresource
EVSLOGFILE
evslogfile-code, name, description, type, start-date
EVSTERM
evsterm-code, name, start-date, end-date
EVSLOG
evslog-code, evsterm-code, name, date, description, data
EVSUSERACTION
evsuseraction-code, name, evsresorce
EVSRESOURCE
evsresource-code, name, type, location

EVS Functionality Design

This component will mainly deal with the query creation and getting it voted.

Collaborative learning applications (CSCL) need to create virtual environments where students, teachers, tutors, etc., are able to cooperate with each other in order to accomplish a common learning goal. To that end, our EVS application must provide support to the three essential aspects existing in any collaborative application, namely coordination, collaboration and communication being communication essential to reach coordination and collaboration. Furthermore, in order to efficiently communicate the knowledge achieved from group activity to users, EVS application must provide full support to the presentation of this information to users in terms of awareness and feedback.

The final objective of this component is to provide functional support to the EVS application. Moreover, this component implements the last stage of the process of embedding information and knowledge into the EVS application (i.e. presentation of the knowledge generated) by providing the users with immediate awareness and constant feedback of what is going on in the system.

This component is made up of the following 6 subsystems:

EVS Coordination. This manages both members and resources within a collaborative group so as to make it possible for members to accomplish their objectives.

EVS Collaboration. The main purpose of this subsystem is to let participants share resources such as files and applications in a collaborative learning environment. Resource sharing may be in asynchronous mode for the moment.

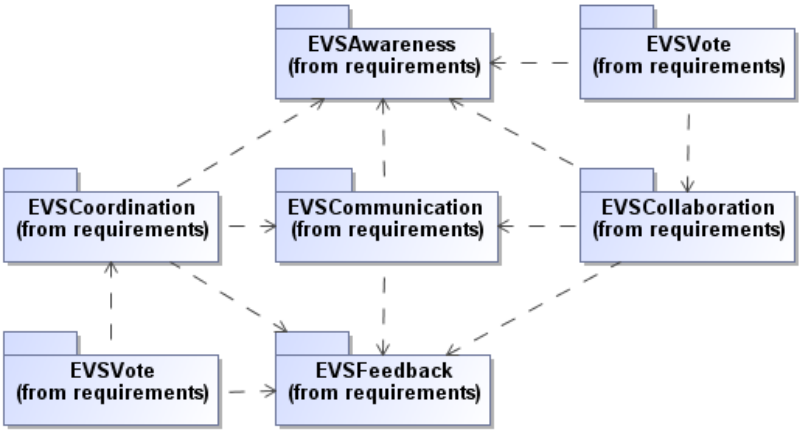
EVS Communication. This manages all the low-level interactions between two or more participants within a collaborative learning group in asynchronous modes.

EVS Awareness. All the events generated by the previous subsystems during a session will be captured and managed by the *EVS Knowledge Management* component and will form the awareness information which will be immediately notified to system users.

EVS Feedback. This aims to influence group participants in a positive manner by means of a steady tracking of parameters outside the task itself and by giving constant feedback of the information related to these parameters to the group.

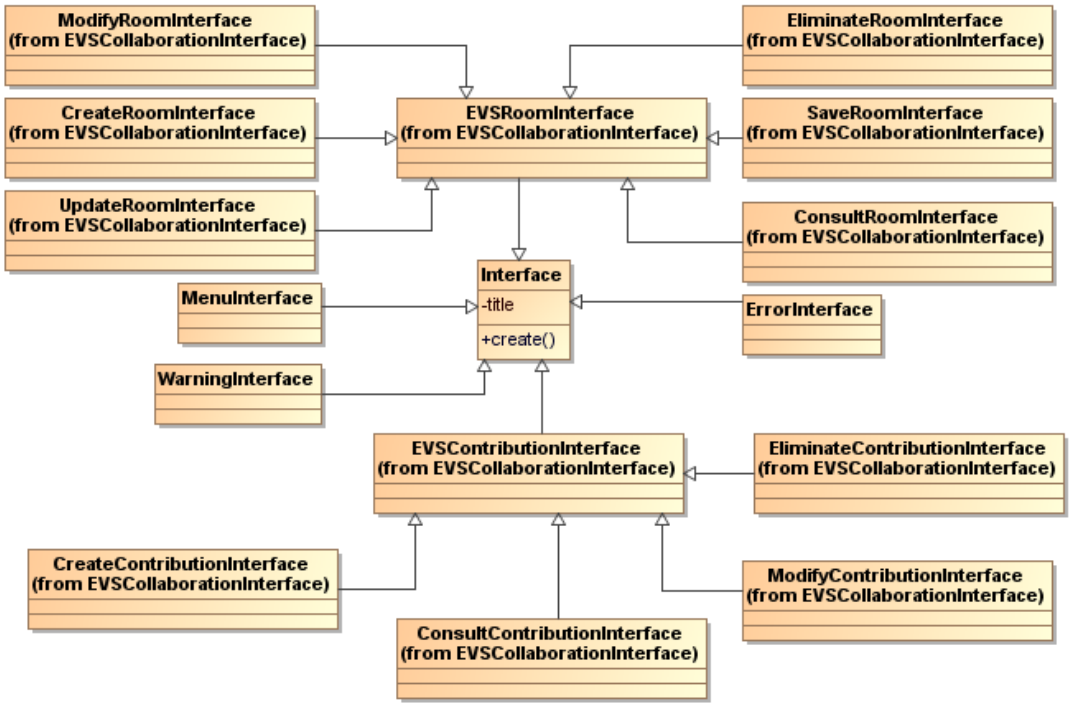
EVS Vote. All the functionality related to the voting system itself instead of the general CSCL support functionalities will be included in this component. Query and vote creation will be managed inside this component. Vote counting, feedback and awareness are meanwhile related to the EVS Knowledge and others functionality subsystems.

Package diagram of this component:

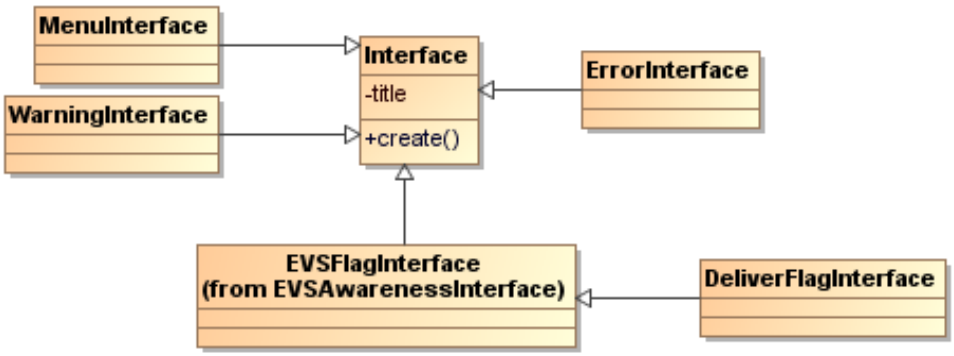


The **user interface**, through which it is possible to manage and maintain the main entities participating in this component, will be generically focused so as to make particularization in graphic and text modes possible. Even though the user interface will usually be in graphic mode, it is necessary to consider this abstraction in order to make the logical part of the application independent from the specific design of the graphic user interface and its posterior updating. In doing so, we guarantee a minimum level of consistency between the presentation and logical layers in the specific application.

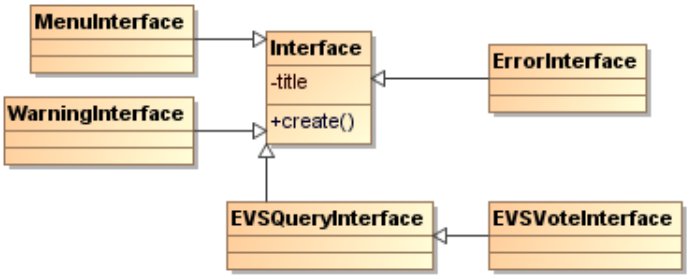
The **EVS Collaboration** subsystem deals with *rooms*, so the user interface hierarchy is as shown in next picture:



The **EVS Awareness** subsystem deals with *states* and *flags*, so the user interface hierarchy is as shown in next picture:



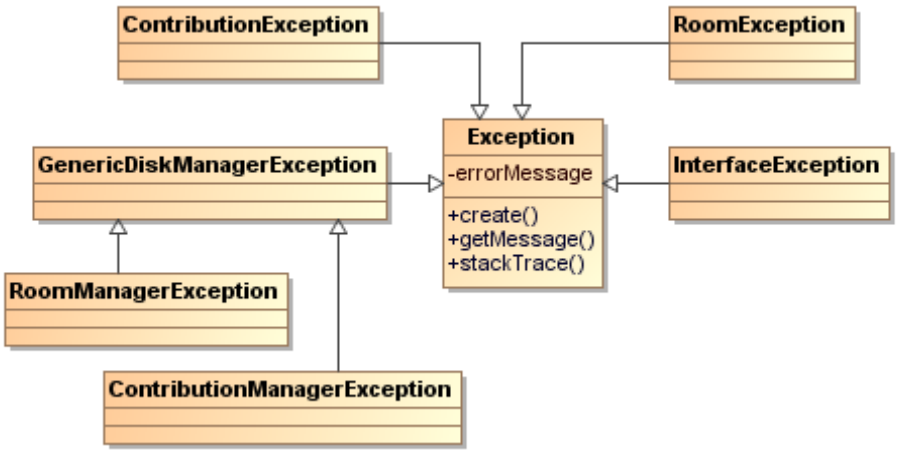
The **EVS Vote** subsystem deals with *queries* and *votes*, so the user interface hierarchy is as shown in next page:



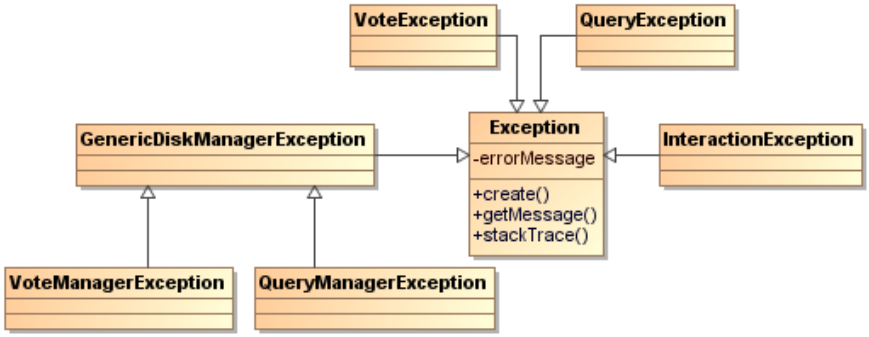
With regards to **persistence**, an abstract data store has been created with which, and through the generic disk manager, it will be possible to model a bridge between the future application and its data in order to make the design of the persistence independent from the specific technology that will manage the data. It will allow the treatment of both ordinary text files and the specific DBMS used during the particularization. Considering the generic design feature and to make the programming language used during the implementation independent from this generic design, it will also be possible to find the classes which create and manipulate the primitive data types with which it will be possible to work with the business classes. On the other hand, generic *Container* and *Iterator* classes have been designed to cross a set of elements given sequentially. We will extend this point later during the persistence design.

Finally, a complete hierarchy of **exception** has been created to catch and treat both the errors produced and the unforeseen anomalous situations in the specification during the work with this component. These exceptions will offer a high degree of reliability to the specific system that is constructed with this component without depending on the error treatment of the specific platform supporting the software. Genericity and independence will be achieved in the same way as is done for error treatment. This treatment will be carried out by the error manager from a single point to facilitate its management. The main objective of this mechanism is to guarantee the robustness of the component.

- EVS Collaboration:



EVS Vote



To summarize, the main objective of this design is to make it independent from any specific technology during the implementation and, in particular, the programming language (in this case, Java will be used) and at the same time to respect the basic principles of the *generic programming (GP)* paradigm as much as possible. For the implementation in Java to be possible, this application has been developed separately with OO methodology and, in order to maintain the ideas of GP design intact, we interpose an implicit logical layer that creates a correspondence between GP and OO design.

EVS Collaboration subsystem

Introduction

Collaboration relies on participants sharing all kind of resources. In a sharing environment, students, as participants in a group, share resources amongst themselves in order to achieve their learning objectives. Sharing resources means those applications that are shared by the participants as the voting tools. To this end, each participant collaborates by incorporating their contributions into the sharing resource so as to elaborate the problem by proposing solutions, discussing the contributions of others and so on. The sharing of resources amongst several participants is therefore a central functionality of the EVS application.

Sharing between different users may be synchronous, with different participants accessing the same resource at the same time (seeing and working on the same copy of the resource such as multi-user editors), and asynchronous, with several users accessing the same resource at different times (each of them working on a different copy of the same document with the possibility of changes of participants in the shared resources being flagged to the others).

In our EVS application we will focus only in the asynchronous functionality. The current scenario is a single user creating a query and sharing them with others users In order to get

vote results. Actually it is not the only possible scenario and synchronous coordination will be focused in further projects or EVS application revisions.

This subsystem aims to provide the basic functionality to share any kind of resource within the collaborative group activity. To this end, the most fundamental abstraction here is a *room*, which is defined in this subsystem to represent all virtual spaces (i.e. sharing applications) where any kind of sharing is performed. Users share a room by making *contributions* all the time. The result of the interaction is then updated in the sharing application to other participants by the mechanisms provided by another subsystem called *EVSCommunication*.

In a collaboration context it is also important to take into account session moderation in order for a collaborative session to be more productive. In our platform these needs are met by using the coordinator features of the *EVS Coordination* subsystem.

All the events generated during sharing will be handled by the *EVS Awareness* subsystem, which will immediately notify users of the current state of the application (e.g. a new query, query closure, etc.). This subsystem will in turn use the *EVS Knowledge Management* component where these events will have been previously handled.

Finally, low-level communication support, which is necessary in the performance of all sharing activities (e.g. creating a query, etc), is provided by the *EVS Communication* subsystem.

The creation and elimination operations of the sharing (i.e. open and close a shared application such as a query) will be carried out by the query owner or the tutor who is responsible for the correct and efficient functioning of the system. With regards to the consultation and modification operations, they refer to the contributions made to the shared application and they are usually done by the group members. Thus, making a new query contribution results as a modification of the shared application.

The **user interface**, through which it is possible to manage and maintain the shared applications, will be focused generically so as to make its particularization in graphic and text modes possible.

With regards to **persistence**, this is also generic and permits particularization in both ordinary text files and the specific DBMS used during the particularization.

EVSFunctionality component

EVS Collaboration subsystem

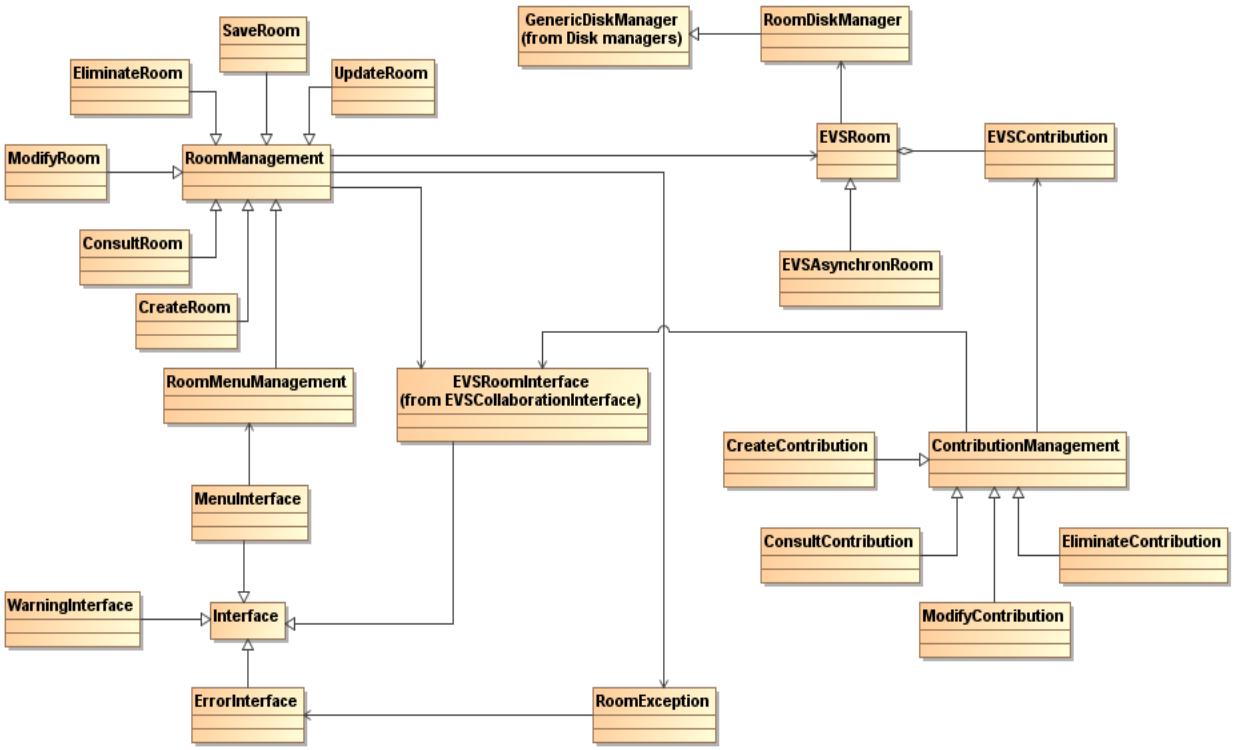
Reused classes from the UsersManagement component which can be found in the *gpl.users* package of the General Purpose Library (*GPL*):

- **Reused business classes** (also from EVSUsersManagement, EVSAdminsitration and EVSKnowledge of the EVS): EVSUser, Interface, ErrorInterface, WarningInterface, MenuInterfece, InterfaceException, GenericDiskManager, GenericDiskManagerException, MenuException, Exception, EVSResource, EVSWorkspace, EVSUsers.

- Reused auxiliar classes: Object, String, Integer, Float, Character, Boolean, Date, Container, Iterator, SortedContainer, StringException, IntegerException, CharacterException, FloatException, BooleanException.

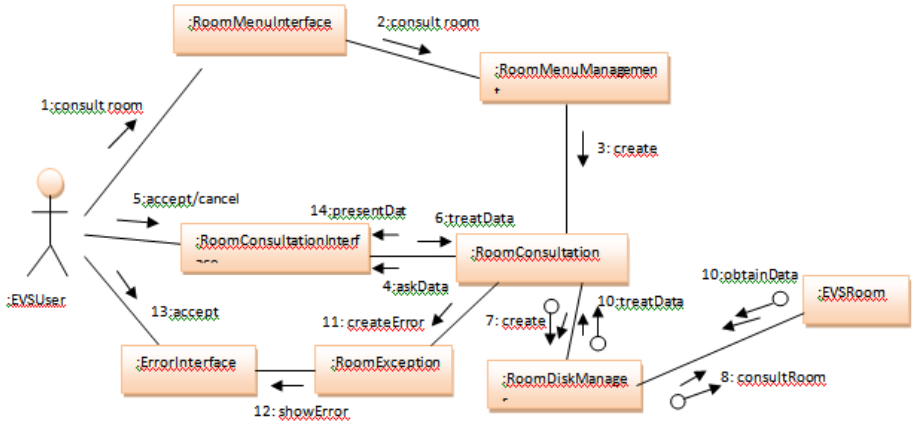
- New classes for this subsystem: EVSRoom, EVSAsynchronicRoom, EVSContribution, EVSRoomManagement, EVSRoomCreation, EVSRoomConsultation, EVSRoomModification, EVSRoomElimination, EVSRoomUpdating, EVSRoomSaving, EVSContributionManagement, EVSContributionCreation, EVSContributionConsultation, EVSContributionModification, EVSContributionElimination, EVSRoomInterface, EVSCreateRoomInterface, EVSConsultRoomInterface, EVSModifyRoomInterface, EVSEliminateRoomInterface, EVSRoomException, EVSRoomMenuManager, EVSRoomDiskManager

Class diagram

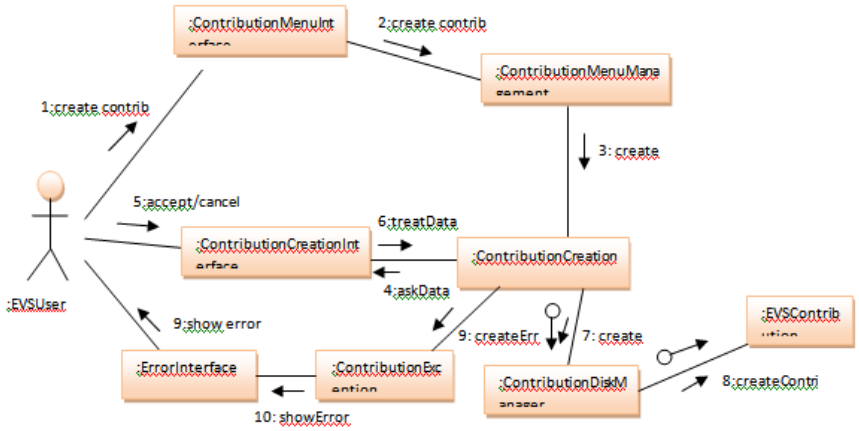


Collaboration diagram

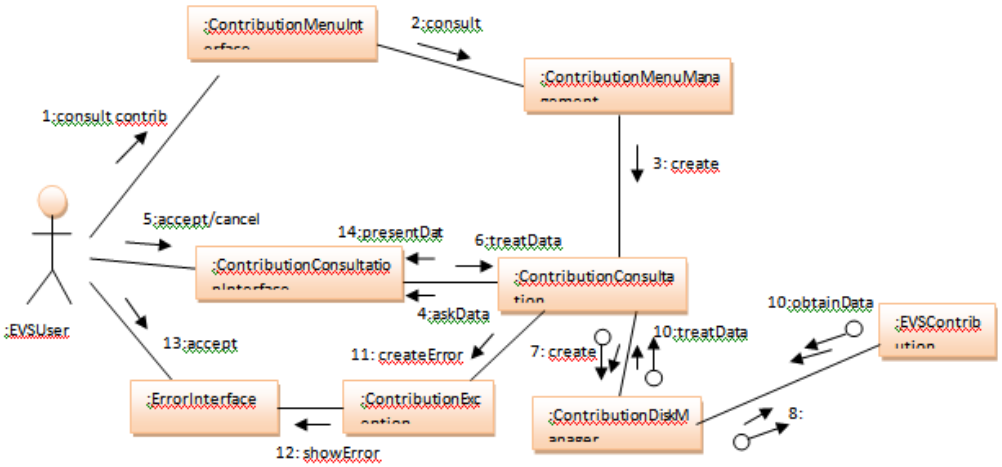
Consult EVSroom



Create EVSContribution:



Consult EVSContribution:



EVS Awareness subsystem

Introduction

Awareness is essential for any of the three forms of cooperation seen, namely coordination, collaboration and communication. It allows implicit coordination of collaborative learning, opportunities for informal, spontaneous communication, and it keeps users informed as to what is happening in the system. Users will receive knowledge of who, when, how and where a query has been created, changed or voted by other users.

Users are continuously interacting with the system and events are generated which, once they have been collected, classified and processed, will then be gathered, managed and formatted by the system as awareness information for distribution to the users involved. This awareness process will use the *EVS Knowledge Management* component where the new events will previously have been managed before communicating them to the users. This process is triggered by the system automatically and so is transparent to non-participating users.

Users need to be aware of the activities performed by others as a basic requirement to solving the task at hand and supporting asynchronous communication (e.g. flagging unvoted queries).

Event redundancy is an essential property during group activity in CSCL applications and, as seen before, it is desired and expected to ensure the provision of users with complete and effective awareness information. Thus, during any interaction for coordination, collaboration, and high-level communication purposes, low-level events from the communication transport mechanism will be generated on certain resources at the same time as other subsystems involved are generating high-level events on the same resources.

As a result, there will be more event information to be managed and thus the resulting awareness information will be more complete. As an example, in making a new query contribution to a thread in the DF group room (collaboration purposes), those low-level events generated by the communication subsystem will only inform about the sender, the recipient (news group server) and the content of the message sent. In the case of high-level events generated by the collaboration subsystem, they will contain information about the type of contribution (e.g. create, help, etc.), the specific group to which the contribution has been made, the topic of the thread, the date and time and so on. Therefore, by unifying these two types of events it is possible to obtain complete awareness information about the contribution made.

In order to provide the essential awareness information to effectively support the three areas seen, this subsystem defines three generic entities, namely *resource state, interaction status* and *group memory*, which support the collaboration, communication and coordination

respectively. Each of these abstractions acts as a vehicle so that awareness information can be classified and presented to users in the correct form depending on the type of activity involved. Thus, being aware of the activities of others is essential for coordination (e.g. consensus-making, group organization, social engagement, etc.).

On the other hand, for the purposes of presentation and notification format, this subsystem defines a *flag* as a single abstraction supporting the presentation of awareness information to users through the user interface by any means: from a visual and simple sign for warning purposes (e.g. a new participant has just created a query) to complex visual and audio effects to keep participants aware of what is happening in the group activity. It may also include different types of short text information provided to report the most recent event history on a specific resource as well as certain statistical analysis results of the group activity.

The ultimate objective of this subsystem is to present awareness information to users in a correct, effective and immediate fashion as the last stage in the process of embedding knowledge and information into CSCL applications.

The **user interface**, through which it is possible to manage and maintain the awareness information, will be focused generically so as to make its particularization in graphic and text modes possible.

With regards to **persistence**, this is also generic and permits particularization in both ordinary text files and the specific DBMS used during the particularization.

EVSFunctionality component

EVS Awareness subsystem

Reused classes from the UsersManagement component which can be found in the *gpl.users* package of the General Purpose Library (GPL):

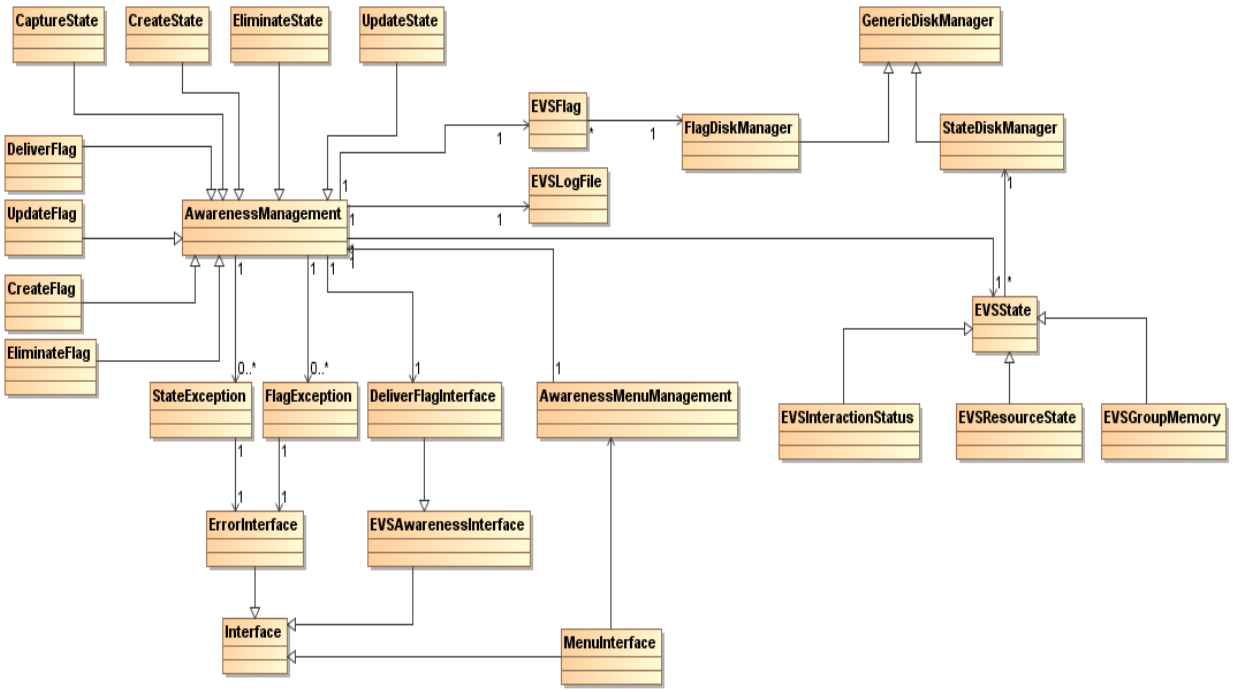
- Reused business classes (also from EVSUsersManagement, EVSAdminsitration and EVSKnowledge of the CLPL): EVSUser, Interface, ErrorInterface, WarningInterface, MenuInterfece, InterfaceException, GenericDiskManager, ProfileDiskManager, GenericDiskManagerException, MenuException, Exception, EVSResource, EVSWorkspace, EVSUsers.

- Reused auxiliar classes: Object, String, Integer, Float, Character, Boolean, Date, Container, Iterator, SortedContainer, StringException, IntegerException, CharacterException, FloatException, BooleanException.

- New classes for this subsystem: EVSInteractionStatu, EVSGroupMemory, EVSFlag, AwarenessManagement, AwarenessInterface, DeliverFlagInterface, AwarenessMenuManager, FlagException, FlagCreation, FlagDelivering, FlagUpdating, FlagElimination

Class diagram

EVS Awareness class diagram



EVS Feedback subsystem

Introduction

Feedback in collaborative learning environments is receiving a lot of attention due to its positive impact on the motivation, emotional state, and problem solving abilities of groups in on-line collaborative learning. It aims to influence group participants in a positive manner by means of a steady tracking of parameters outside the task itself (such as motivation and emotional state) and by giving a constant feedback of these parameters to the group. Therefore, when users participate in an EVS application, they may enhance their abilities by increasing their knowledge about others in terms of motivation, interaction behavior and so on.

Feedback goes one step further than awareness by providing exhaustive information of what is going on in the group over a long period of time (e.g. constantly showing to each group member the absolute or relative amount of the contributions of others). Furthermore, feedback may be obtained about the emotions and motivation of participants through asking them about these states. In all cases, feedback implies receiving information continuously updated.

During the feedback process, all new information communicated to the users will have been previously collected, classified and analyzed by the *EVS Knowledge Management* component. As a consequence of the complex knowledge provided to participants in form of feedback (e.g. group's member relative and absolute amount of contributions, etc) this subsystem makes a strong use of the statistical analysis and need to show the results obtained in complex graphical formats.

In this subsystem we define certain generic entities such as *pool* and *chart* and the necessary functions to correctly manage these entities. Based on these abstractions it is possible to dynamically gather and store great amounts of history data and statistical results from the group activity in order to constantly update and present them to participants in the appropriate diagrammatic form (e.g. pie chart, histograms, etc.).

Concrete data to be collected from a pool in the context of our EVS application:

EVS Student point of view:

- Number of students with access granted to vote the query (census)
- Number of students who voted the query (participation)
- Query usefulness (peer assessment for the query)
- Average query usefulness from all the queries created by the current query creator (peer assessment for the user who created the query)
- Number of contributions created by the current user.
- Number of contributions created by the best contributor.
- Number of queries voted by the current user.
- Number of remaining queries to vote by the current user.

EVS Tutor point of view (in addition to the student view):

- Query mark (tutor assessment for the query)
- Average query mark from all the queries created by the current query creator (tutor assessment for the user who created the query)

This information will be shown as textual and bar charts.

The ultimate objective of this subsystem is to provide group members with constant and elaborated information and knowledge about what is going on in the group activity in order to enhance other members' characteristics different of those related to awareness such as motivation and emotional state. These benefits will provide group activity with an improvement in terms of member's engagement, contribution quality and quantity, task development time and so on.

The **user interface**, through which it is possible to manage and maintain the feedback information, will be focused generically so as to make its particularization in graphic and text modes possible.

With regards to **persistence**, this is also generic and permits particularization in both ordinary text files and the specific DBMS used during the particularization.

EVS Feedback subsystem

Reused classes from the UsersManagement component which can be found in the *gpl.users* package of the General Purpose Library (*GPL*):

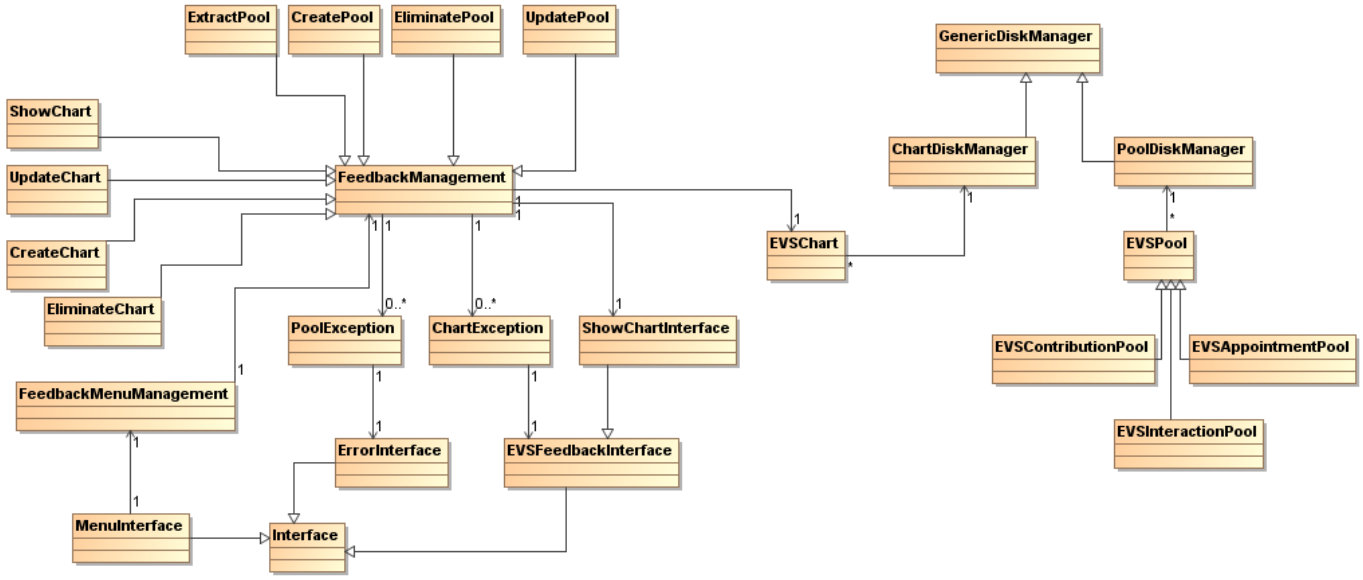
- Reused business classes (also from EVSUsersManagement, EVSAdminsitration and EVSKnowledge of the EVS): EVSUser, Interface, ErrorInterface, WarningInterface, MenuInterfece, InterfaceException, GenericDiskManager, ProfileDiskManager, GenericDiskManagerException, MenuException, Exception, EVSResource, EVSWorkspace, EVSUsers.

- Reused auxiliar classes: Object, String, Integer, Float, Character, Boolean, Date, Container, Iterator, SortedContainer, StringException, IntegerException, CharacterException, FloatException, BooleanException.

- New classes for this subsystem: EVSPool, EVSAppointmentPool, EVSContributionPool, EVSInteractionPool, EVSChart, FeedbackManagement, PoolCreation, PoolExtraction, PoolUpdate, PoolDiskManager, PoolElimination, FeedbackInterface, ShowChartInterface, FeedbackMenuManager, PoolException, ChartException, ChartCreation, ChartDelivering, ChartUpdating, ChartElimination, VotePool, VoteChart.

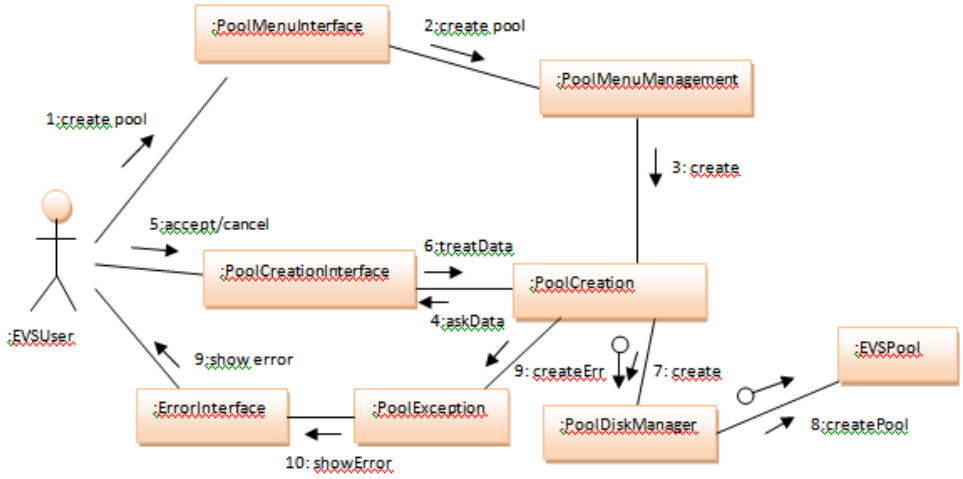
Class diagram

Feedback class diagram



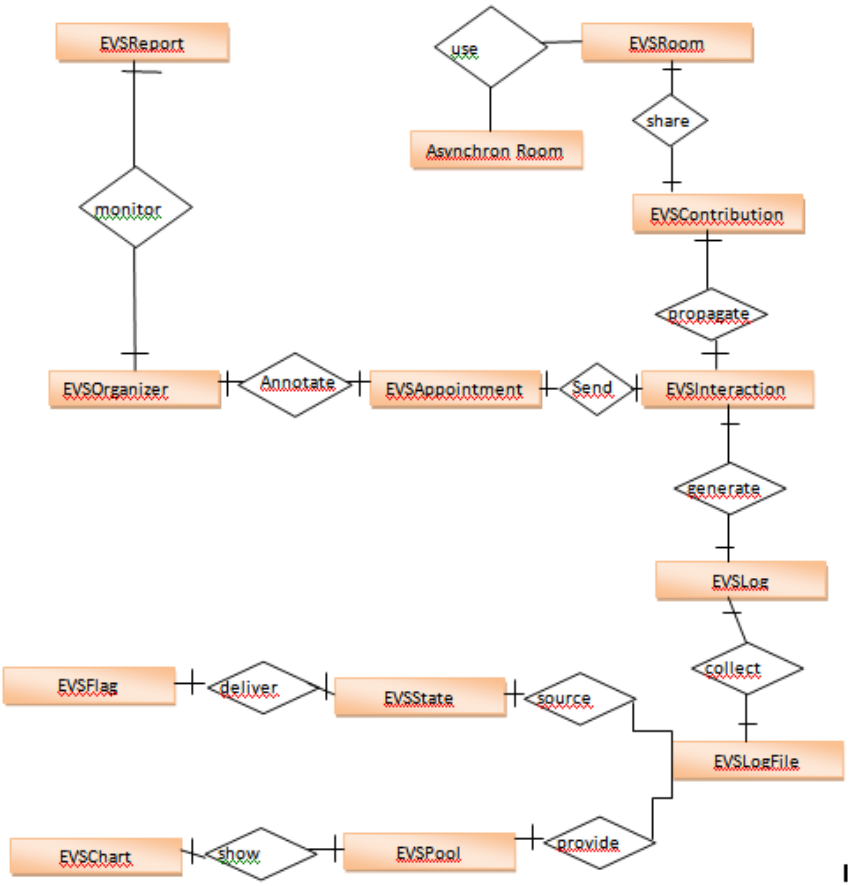
Collaboration Diagrams

Create pool



Conceptual and logic design

Conceptual design (ER Diagram)



Attribute description

*The outlined attributes represent the primary keys

EVSREPORT
csclreport-code, name, date, coordinator, content

EVSORGANIZER
csclorganizer-code, name, group

EVSLAPPOINTMENT
clappointment-code, date, content, organizer

EVSLOG
csclllog-code, name, date, logfile, data

EVSLOGFILE
csclllogfile-code, name, description, data

EVSINTERACTION
csclinteraction-code, sender, receiver, message, channel

EVSROOM
EVSRoom-code, workspace, group, moderator

EVSASYNCHRONROOM
csclasynchronroom-code

EVSCONTRIBUTION
csclcontribution-code, type, author, date, content, room

EVSSTATE
csclstate, name, area, source, content

EVSFLAG
csclresource-code, name, type, state, content, interaction

EVSPOOL
csclpool-code, name, feature, source, content

EVSCHART
EVSUseraction-code, name, type, pool, content, interaction

EVSSYNCHRONROOM
csclsynchronroom-code

EVS Vote Subsystem

Introduction

In order to keep our concrete functionality one step away from the framework we have decided to include all our concrete voting classes in a new subsystem.

This subsystem relies on the other five subsystems and in the knowledge component for many things as feedback, awareness, charting and so. We will reuse as many features on the CLPL framework as we can.

Diving into concrete decisions, we have decided to design four query types:

- Select only one answer. *SimpleQuery*
- Select many answers as you need. *MultipleVote*.
- Rank your choices. *RankingCount*.
- Rank your choices giving points. *Weighted RankingCount*.

Other decisions related to vote the vote system and already seen in the knowledge component are:

- Allow blank vote.
- Allow to show the results before the end of the query.
- Allow open answers.
- Allow explanatory text answers for each answer.

General considerations:

- The query needs an end date.
- The query can only be edited or removed before getting any vote.
- A query will be created from an existing closed query. Duplicate queries are allowed but not at the same period of time in the same classroom.
- Vote will be anonymous to improve participation.
- Each user can vote each query only one time. There is no way to edit a vote when it is emitted. User will receive a confirmation warning before sending the vote to avoid unintentionally voting.

Special discussion forum features

As a discussion and consensus tool it makes sense to imagine queries related to discussion forum threads. If we want to discuss over the query, a DF is a must have.

A requirement made by the advisor is that our EVS will be related to the DF to easily create queries from the existing posts in the DF. To solve this problem an action element –e.g. a button- will be added in each discussion with some text –i.e. “query proposal”- to indicate there is a good idea to discuss in a voting system. These “proposed to query” post will be eligible when creating a new query in the main thread context.

Features related to DF are:

- Make a query for this existing thread.
- Show query proposals for this existing thread.
- Make a query and open a thread related to discuss it.
- Block thread when the voting process is open. This rule will apply when we want to discuss, to vote and to discuss again about the results.

New classes in this subsystem

Note: Due to similarity over many of these classes –e.g. simplequery, multiplechoice, rankcounting and ranking- we are only describing some of them in the CRC cards.

Vote, Query, Question, QueryProposal, SimpleQuery, MultipleChoice, Ranking, RankCounting, EVStoDFtoEVSInterface, VoteException, VoteDiskManager, VoteCreation, VoteUpdating, VoteElimination, VoteConfiguration, VoteConsultation, VoteConfigurationConsultation, VoteConfigurationUpdating, VoteConfigurationElimination, QueryConfigurationConsultation, QueryConfigurationUpdating, QueryConfigurationElimination, QueryConfigurationException, QuestionException, QuestionDiskManager, QuestionCreation, QuestionUpdating, QuestionElimination, QuestionConfiguration, QuestionConsultation, QuestionConfigurationConsultation, QuestionConfigurationUpdating, QuestionConfigurationElimination, QueryProposalException, QueryProposalDiskManager, QueryProposalCreation, QueryProposalUpdating, QueryProposalElimination, QueryProposalConfiguration, QueryProposalConsultation, QueryProposalConfigurationConsultation, QueryProposalConfigurationUpdating, QueryProposalConfigurationElimination, SimpleQueryException, SimpleQueryDiskManager, SimpleQueryCreation, SimpleQueryUpdating, SimpleQueryElimination, SimpleQueryConfiguration, SimpleQueryConsultation, SimpleQueryConfigurationConsultation, SimpleQueryConfigurationUpdating, SimpleQueryConfigurationElimination, MultipleChoiceException, MultipleChoiceDiskManager, MultipleChoiceCreation, MultipleChoiceUpdating, MultipleChoiceElimination, MultipleChoiceConfiguration, MultipleChoiceConsultation, MultipleChoiceConfigurationConsultation, MultipleChoiceConfigurationUpdating, MultipleChoiceConfigurationElimination, RankingException, RankingDiskManager, RankingCreation, RankingUpdating, RankingElimination, RankingConfiguration, RankingConsultation, RankingConfigurationConsultation, RankingConfigurationUpdating, RankingConfigurationElimination, RankCountingException, RankCountingDiskManager, RankCountingCreation, RankCountingUpdating, RankCountingElimination, RankCountingConfiguration, RankCountingConsultation, RankCountingConfigurationConsultation, RankCountingConfigurationUpdating, RankCountingConfigurationElimination.

Class	***Vote***
Class description	Creates a generic vote representing any vote related to a query question.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic vote in an EVS system	String, Date, Object, Query, Question
Vote	
-identifier: String //identifier of this vote **-sharing: Query** //the query which this vote belongs to **-answer: Question** //The question which this vote is going to **-type: String** //the type of this vote **-text: String** //the text of this vote **-resource: Resource** **-participant: User** **-date: Date** //the date when the appointment is done **-content: Object**	
//stores a generic vote with an identifier.	

+Vote (pIdentifier: String, pQuery: Query)
//get the identifier
+getIdentifier(): String
//set the identifier
+setIdentifier(pIdentifier: String)
//get the question which this vote belongs to.
+getQuery(): Query
//set the question which this vote belongs to.
+setQuery(pQuery: Query)
//set the text of this vote
+getText(): String
//get the text of this vote
+setText(pText: String)
//get the date
+getIDate(): Date
//set the date
+setIDate(pDate: Date)

Class	***Query***
Class description	Creates a generic query representing any query related to a DF or EVS subject.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic query in an EVS system	String, Date, Object, Query, Question
Query	
-identifier: String //identifier of this query **-type: String** **-resource: Resource** **-workspace: Resource** **-participants: Resource** **-contributions: Object** **-history: Date** **-result: Result** **-config: Configuration** **-questions: Question**	

//stores a generic query with an identifier.
+Query (pIdentifier: String)
//get the identifier
+getIdentifier(): String
//set the identifier
+setIdentifier(pIdentifier: String)
//get the query.
+getQuery(): Query
//set the query which this appointment belongs to.
+setQuery(pQuery: Query)
//get the text of this appointment
+getText(): String
//set the text of this vote
+setText(pText: String)
//get the date
+getDate(): Date
//set the date
+setDate(pDate: Date)	

Class	***Question***
Class description	Creates a generic question for any query related to a DF or EVS subject.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic question in an EVS system	String, Date, Object, Query, Question
Question	
-identifier: String //identifier of this question **-text: String** **-relatedTo: Query** **-linkTo: String** **-type: String** **-votes: Vote**	
//stores a generic question with an identifier and query. **+Question (pIdentifier: String, pQuery: Query)**	

Class	***SimpleQuery***
Class description	Creates a generic simple query..
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic simple query in an EVS system	String, Date, Object, Query, Question
SimpleQuery extends Query	
-identifier: String //identifier of this query **-type: String** **-resource: Resource** **-workspace: Resource** **-participants: Resource** **-contributions: Object** **-history: Date** **-result: Result** **-config: Configuration** **-questions: Question**	
//stores a generic simpleQuery with an identifier. **+SimpleQuery (pIdentifier: String)**	

Class	***MultipleChoice extends Query***
Class description	Creates a generic multiple choice query representing any query related to a DF or EVS subject.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic multiple choice query in an EVS system	String, Date, Object, Query, Question
MultipleChoice extends Query	
-identifier: String //identifier of this query **-type: String**	

-resource: Resource **-workspace: Resource** **-participants: Resource** **-contributions: Object** **-history: Date** **-result: Result** **-config: Configuration** **-questions: Question**
//stores a generic appointment with an identifier and organizer. **+MultipleChoice (pIdentifier: String)**

Class	***Ranking extends Query***
Class description	Creates a generic ranking query representing any query related to a DF or EVS subject.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic ranking query in an EVS system	String, Date, Object, Query, Question
Ranking extends Query	
-identifier: String //identifier of this query **-type: String** **-resource: Resource** **-workspace: Resource** **-participants: Resource** **-contributions: Object** **-history: Date** **-isSynchronic: Boolean** **-result: Result** **-config: Configuration** **-questions: Question**	
//stores a generic ranking query with an identifier. **+ranking (pIdentifier: String)**	

Class	***RankCounting extends Query***
Class description	Creates a generic query representing any query related to a DF or EVS subject.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic vote in an EVS system	String, Date, Object, Query, Question
RankCounting extends Query	
-identifier: String //identifier of this query **-type: String** **-resource: Resource** **-workspace: Resource** **-participants: Resource** **-contributions: Object** **-history: Date** **-isSynchronic: Boolean** **-result: Result** **-config: Configuration** **-questions: Question**	

//stores a generic rank counting with an identifier. **+rankCounting (pIdentifier: String)**

Class	***EVStoDFInterface extends Interface***
Class description	Creates a generic interface to communicate between the DF and the EVS. Mainly for blocking discussion between voting days and to communicate results.
Class type	Property: main class
Class features	Abstract, compound, persistent
Responsabilitats	**Col·laboracions**
Model a generic DF to EVS interface to block or create discussions from a query.	String, Date, Object, Query, Question, QueryProposal
EVStoDFInterface extends Interface	
-identifier: String //identifier of this vote **-resource: Resource** **-workspace: Resource** **-participants: Resource** **-contributions: Object**	
//creates. **+creates ()**	

Class	***QuestionCreation***
Class description	Carries out the logic part of the creation of a question
Class type	Property: main class
Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Creates a question object. Get and send information to the persistent objects	QuestionException, QuestionCreationInterface, Question
QuestionCreation	
#qci: QuestionCreationInterface	
//creates **+QuestionCreation()**	

Class	***QuestionConsultation***
Class description	Carries out the logic part of the consultation of a question
Class type	Property: main class
Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Consults a question object. Get information from the persistent objects	QuestionException, QuestionConsultationInterface, Question
QuestionConsultation	
#qci: QuestionConsultationInterface	
//creates **+QuestionConsultation()**	

Class	***QuestionUpdating***
Class description	Carries out the logic part of the updating of a question
Class type	Property: main class

Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Updates a question object. Get and send information to the persistent objects	QuestionException, QuestionUpdateInterface, Question
QuestionUpdating	
#qui: QuestionUpdateInterface	
//creates **+QuestionUpdate()**	

Class	***QuestionElimination***
Class description	Carries out the logic part of the elimination of a question
Class type	Property: main class
Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Eliminates a question object. Get and send information to the persistent objects.	QuestionException, QuestionEliminationInterface, Question
QuestionElimination	
#qei: QuestionEliminationInterface	
//creates **+QuestionElimination()**	

Class	**QuestionException extends Exception**
Class description	Class managing the errors generated by the question management
Class type	Property: main class
Class features	Concrete, compound
Responsabilitats	**Col·laboracions**
Manages the errors during the question management	
QuestionException extends Exception	
//creates an exception provoked for the unforeseen situations during the question management **+QuestionException()**	

Class	**QuestionDiskManager extends GenericDiskManager**
Class description	Class managing the bridge between the EVS application and the persistent data of the question
Class type	Property: main class
Class features	Concrete, compound
Responsabilitats	**Col·laboracions**
Manages the bridge between the EVS application and the persistent data of the question.	
QuestionDiskManager extends GenericDiskManager	
//creates a bridge between the EVS application and the persistent data of the question. **+QuestionDiskManager()**	

Class	***VoteCreation***

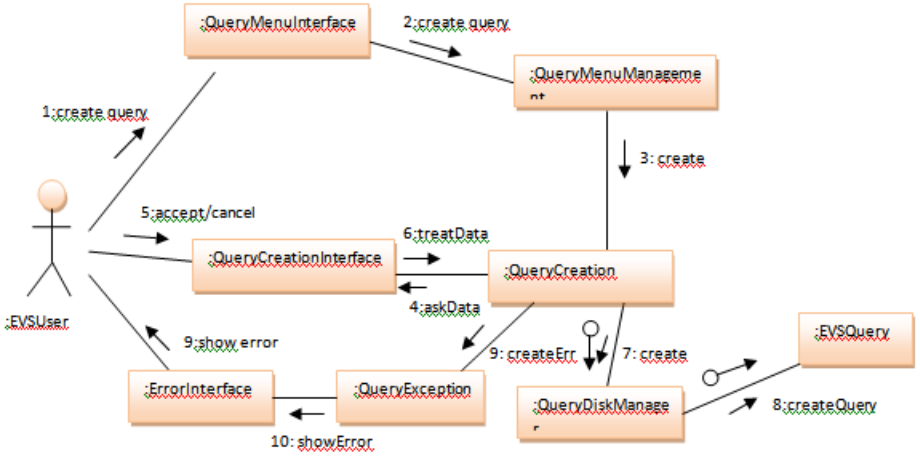
Class description	Carries out the logic part of the creation of a vote
Class type	Property: main class
Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Creates a vote object. Get and send information to the persistent objects	VoteException, VoteCreationInterface, EVSVote
VoteCreation	
#vci: VoteCreationInterface	
//creates **+VoteCreation()**	

Class	***VoteConsultation***
Class description	Carries out the logic part of the extraction of a vote
Class type	Property: main class
Class features	Abstract, compound
Responsabilitats	**Col·laboracions**
Consults a vote object. Get and send information to the persistent objects.	VoteException, VoteConsultationInterface, EVSVote
VoteConsultation	
#vci: VoteConsultationInterface	
//creates **+VoteConsultation()**	

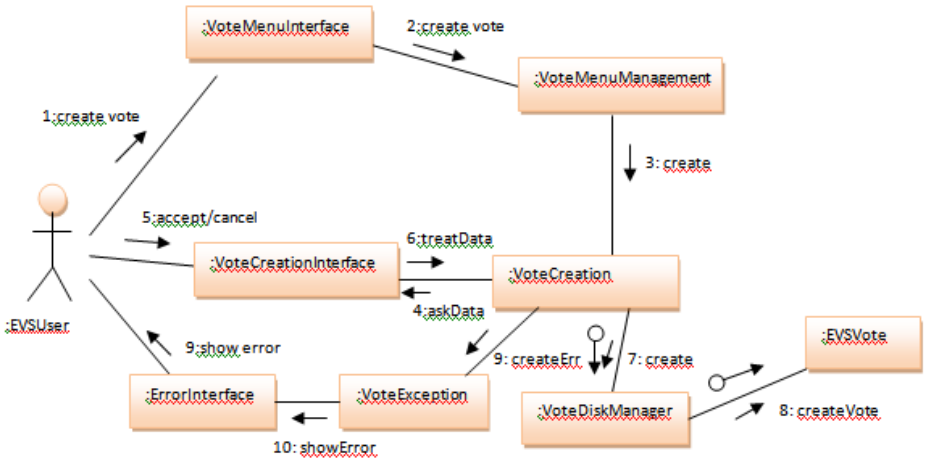
Class	**VoteException extends Exception**
Class description	Class managing the errors generated by the vote management
Class type	Property: main class
Class features	Concrete, compound
Responsabilitats	**Col·laboracions**
Manages the errors during the vote management	
VoteException extends Exception	
//creates an exception provoked for the unforeseen situations during the vote management **+VoteException()**	
Class	**VoteDiskManager extends GenericDiskManager**
Class description	Class managing the bridge between the EVS application and the persistent data of the vote
Class type	Property: main class
Class features	Concrete, compound
Responsabilitats	**Col·laboracions**
Manages the bridge between the EVS application and the persistent data of the vote.	
VoteDiskManager extends GenericDiskManager	
//creates a bridge between the EVS application and the persistent data of the pool. **+VoteDiskManager()**	

Collaboration diagrams

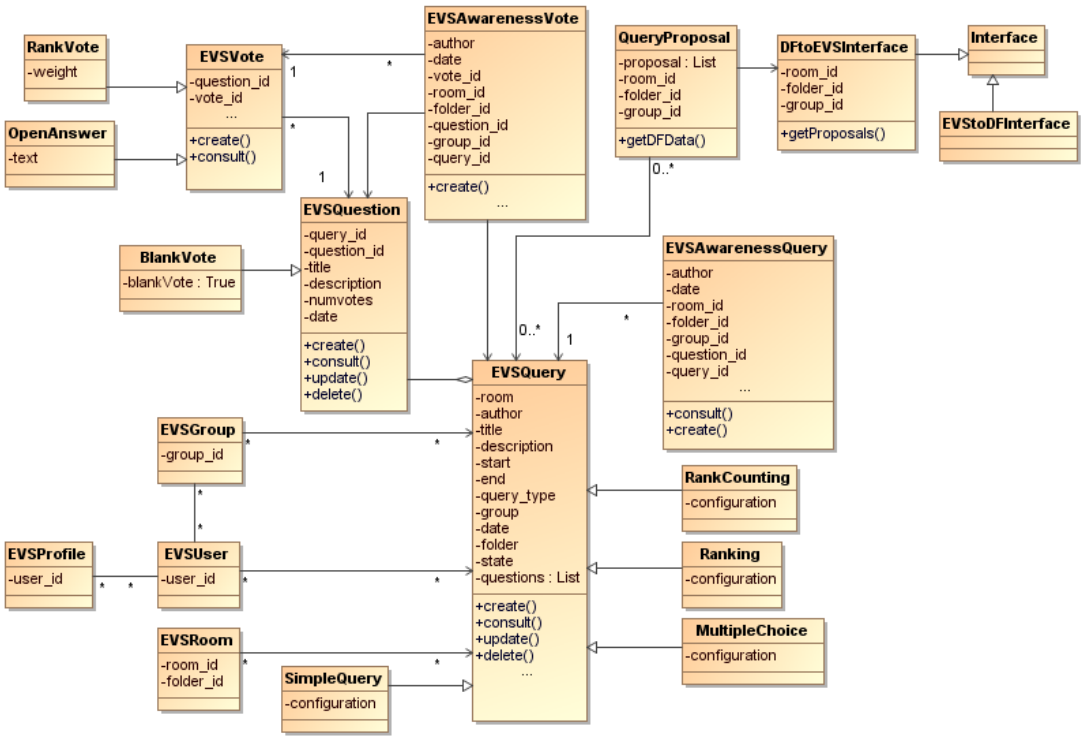
Create query



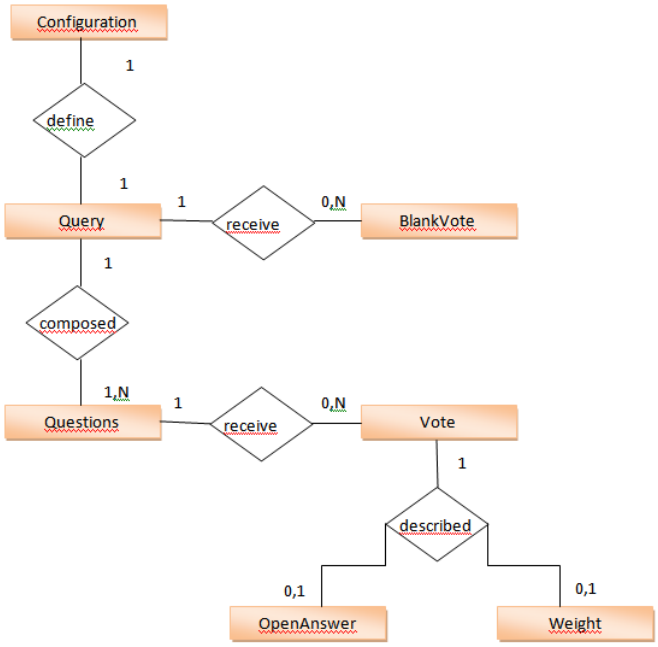
Create vote



Class diagram



Persistence diagram



Attribute description

*The outlined attributes represent the primary keys

CONFIGURATION

configuration-id, query-type, allow-blank-vote, allow-multiple-choice, allow-open-answers, allow-explained-answers, block-discussion, show-results-before-end, start-date, end-date

QUERY

query-id, title, description, configuration-id, date

QUESTION

question-id, title, description, query-id

VOTE

Vote-id, date, question-id
BLANKVOTE
Blankvote-id, date, question-id
OPENANSWER
answer, vote-id
WEIGHT
value, vote-id

Note: Interaction with common EVS entities as USER (who votes), ROOM (the query belongs to) or GROUP (related with room access) are intentionally avoid from the above diagram. For an improved view of the relations see the entity class diagram above or the complete attribute description in the topic *entities to be generated* below.

Persistence design

Introduction

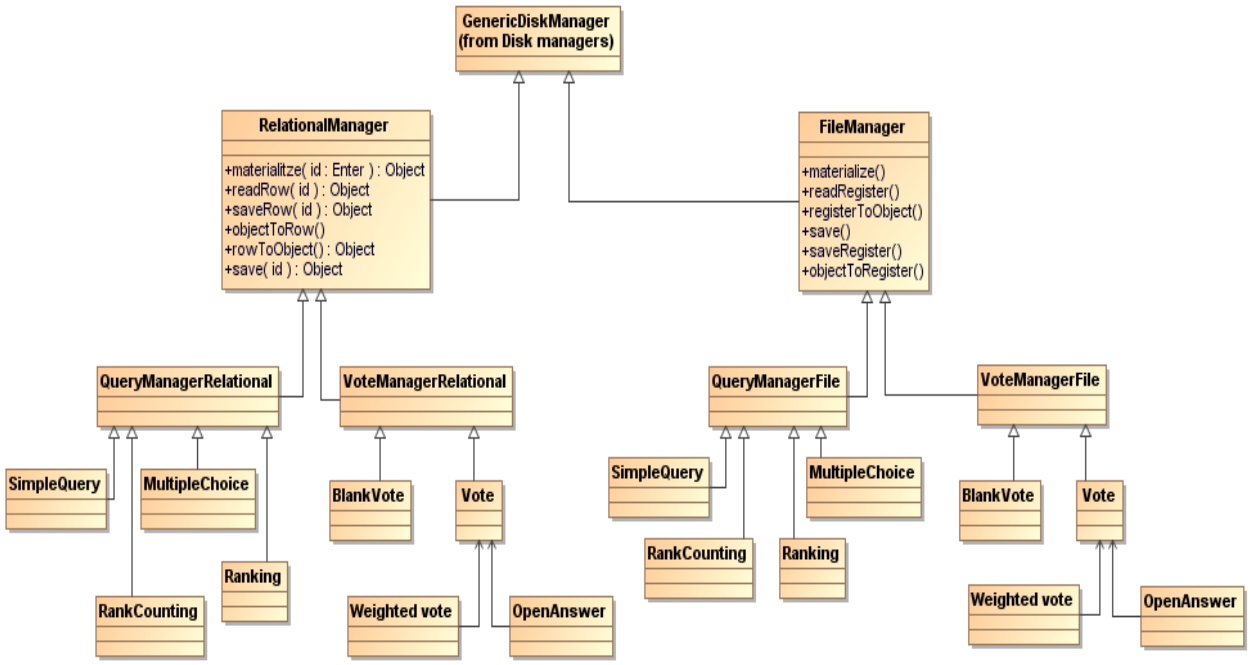
The *generic programming* paradigm will be applied to the design of the persistence as those sessions with lifespan exceeding that of the process that created them requires permanent storage and a management of the persistence which is independent from the management of the mentioned storage.

Furthermore, the persistence implementation can be done by both database (relational and object oriented) and ordinary text files. In order to achieve this objective, we will base the persistence management in the *disk manager* which permits independence as it enables access database or text file, so, letting the *disk manager* communicate the specific DBMS or file system. Hence, if a specific data manager is substituted by another, the only thing which needs changing will be the *manager disk* that maintains the object persistence in the same form.

A generic data manager will be created of those operations which are common to both the database and file models. For each one of the two technologies a specific manager disk will contain the specific operations required to access data according to the current storage model.

The complete disk manager hierarchies for each subsystem are as follows:

EVS Vote



Entities to be generated

Assuming that the project will be implemented over the relational database paradigm we can convert the ER diagram above in the next relational tables.

There are only four tables we need to create specifically for this project; all the other tables are already created by the framework.

We are generating persistence for the query header, the query lines –questions- and the vote. Finally we need another table to save awareness data about our specific query contributions. This kind of specific awareness would be managed by the CSCLAWARENESS entity, but instead of extending the base entity with a new "awareness type" we are using a new entity to keep the framework reusable and not particularized during the further implementation.

The *blank vote* and the *open answer* will be managed as a special type of question in the query instead of creating weak entities. That brings the design even more clear and congruent.

An especial field called *vote_count* would be created on the EVSQUESTION entity to avoid unnecessary counting operations.

EVSQUERY (
query_id INT not null PRIMARY KEY,
room_id int not null FOREIGN KEY REFERENCES CSCLROOM(room_id),
author_id int not null FOREIGN KEY REFERENCES CSCLUSER(user_id),
group_id int not null FOREIGN KEY REFERENCES CSCLGROUP(group_id),
title VARCHAR(250) not null,
description varchar(2500),
start_date DATE not null,
end_date DATE not null,
query_type DOMAIN('simple', 'multiple', 'ranking', 'counting'),
allow_blank_vote BOOLEAN not null DEFAULT(FALSE),
allow_multiple_choice BOOLEAN not null DEFAULT(FALSE),
allow_open_answers BOOLEAN not null DEFAULT(FALSE),
allow_explained_answers BOOLEAN not null DEFAULT(FALSE),
mark varchar(50) DEFAULT -1,
nuseful int4 DEFAULT 0,
useful int4 DEFAULT -1,
state DOMAIN('open', 'close'),
date varchar
)

EVSQUESTION(
question_id INT not null PRIMARY KEY,
query_id int not null FOREIGN KEY REFERENCES EVSQUERY(query_id),
title varchar(100),
description varchar(2500),
question_type DOMAIN('simple','blank_vote, 'open_answer'),
vote_count INT not null DEFAULT(0))

EVSVOTE(
vote_id INT not null PRIMARY KEY,
question_id INT not null FOREIGN KEY REFERENCES EVSQUESTION(question_id),
user_id int not null FOREIGN KEY REFERENCES CSCLUSER(user_id),
group_id int not null FOREIGN KEY REFERENCES CSCLGROUP(group_id),
room_id int not null FOREIGN KEY REFERENCES CSCLROOM(room_id),
folder_room_id int not null FOREIGN KEY REFERENCES CSCLROOM (folder_room_id),

date varchar not null,
explanation TINYTEXT null,
weight INT null)

EVSAWARENESS (
evsawareness_id INT not null PRIMARY KEY,
query_id INT not null FOREIGN KEY REFERENCES EVSQUERY(query_id),
user_id int not null FOREIGN KEY REFERENCES CSCLUSER(user_id),
group_id int not null FOREIGN KEY REFERENCES CSCLGROUP(group_id),
room_id int not null FOREIGN KEY REFERENCES CSCLROOM(room_id),
folder_room_id int not null FOREIGN KEY REFERENCES CSCLROOM (folder_room_id),
date varchar not null)

Chapter 3. Implementation

Preliminary notes

Our project will be developed under *Web Services* [20][39][40]. We'll use Apache Axis [21] as the SOAP server. *Web services* run over standard protocols and produces loosely coupled applications. All the functionality from the *CLPL framework* is accessible through *web services*. Our choice for the client side is *PHP* and our *RDBMS* will be *Postgresql* [22].

In order to install the application please refer to the *Annex I. EVS Installation*

Requirements

Local development has been done in an: Intel Core2 Duo – 1,7GHz - 2 GB RAM (128 shared) – HDD 150GB SATA1.

It is important to say that this configuration is minimal and not suitable for a comfortable development. See our *Annex III. Workstation Issues*.

Software requirements

In addition to our df*evs-web.zip* client application and our df*evs.zip* server application we need the following software as a requirement. The following versions are from our local development workstation. It's important to get exactly our version number. There are many issues due to version changes.

- Apache 2.2.15 (web server), Apache Tomcat 6.0.26 (web services server)
- Java JDK 1.6.0_12
- PHP 5.2.13
  - SOAP 0.9.4 (from PHP PEAR included in df-web.zip for commodity reasons)
- Apache Axis 1.4 (with the following *.jar in the lib)
  - commons-discovery-0.2, commons-logging-1.0.4, jaxrpc.jar, saaj.jar, log4j-1.2.8, wsdl4j-1.5.1,axis.jar

  Optional
  - javamail-1.4.3, xml-security-bin-1_4_0, xml-apis.jar, axis-ant.jar, xerces-2.9.1, jaf-1.0.2
- PostgresQL.8.1 (database server) [23][24][28]
  - Postgresql.jar (Java database driver)

Other software

Even if we can do all our work with a simple text editor we used some tools in order to facilitate our work.

- *Notepad++ 5.8.7* (PHP code editor): A lightweight code editor with many features regarding search, and helpers to format the code. There is only a lack in class or functions explorer.
- *tIDE 2.32* (Java IDE): Our development workstation couldn't carry out with other Java IDE like Eclipse or NetBeans but a class explorer is a must when working with large number of files, packages and methods. *tIDE* is a 4Mb IDE that brings a good syntax colorize and a great class explorer.
- *Google Chrome* 11.0.696.71 (web browser): Light-weight, multi-threaded, full standard compliant web browser.
- *Python 2.7.1* (scripting tasks): We use python in the localization process. To extract the translation strings from our web client files and generating the translation file.
- *Jacobe 7.3.14 (JAVA code beautifier)*: We used Jacobe in order to bring our code more readable and easy to debug. We used the Sun code standard [32].
- *PHPBeatufier (PHP code beautifier)*: We used a PHP code beautifier trying to get our code easier to read and debug. We used the PEAR code standard [33].

We used batch (MS-DOS) files to facilitate the compilation and deployment.

Introduction

Specifying and designing we have proposes a complex project without renouncing to any features. If we were experienced with the technology all the features would be done in time, but we weren't experienced with the technology and understanding the DF internals and getting all working was a long, difficult work.

However due to the CSCL [1][2] framework performance and high reusability we can offer a complete but not exhaustive EVS solution.

The following table will describe what is implemented and what is not from our previous project expectative.

Yes	Create, delete and consult queries.	No	Update query
Yes	Select only one answer.	No	Select many answers as you need.
Yes	Use contributions from the DF to create a query.	No	Rank your choices.
Yes	Total blank votes	No	Rank your choices giving points.
Yes	Allow to show the results before the end of the query.	No	Allow open answers.
Yes	Allow blank vote.	No	Allow explanatory text answers for each answer.
Yes	The query can only be removed before getting any voted.	No	A query will be created from an existing closed query.
Yes	Vote will be secret –i.e. anonymous- to improve participation.	No	Block thread when the voting process is open. [Almost done]
Yes	Each student can vote each query only one time. There is no way to edit a vote when it is emitted.	No	Make a query and open a thread related to discuss it.
Yes	Make a query for this existing thread.	No	Total open answers [Almost done]
Yes	Show query proposals for this existing thread.	Yes	The user will be able to remember their vote.
Yes	Total students.	Yes	The most active users will be announced to encourage participation and recognition.
Yes	Total votes for each option.	Yes	Tutor extended statistics and marks.
Yes	Last vote day.	Yes	Total abstentions
Yes	Total votes	Yes	The user will know how many queries he not yet voted.
Yes	The user will get notifications when a new query is open.	Yes	The user will easily access to results of closed queries.

The following table describes some project improvements not detected in previous phases but included to improve the final project result. The contact with the DF implementation helped to do a better educative tool.

Yes	Peer query usefulness evaluation
Yes	New localization implementation

There're also user interface improvements from our previous design proposal.

About the features selection

The entire EVS project until now was done without a real approach into the DF application. Actually we aren't developing a standalone application but an application that works into the DF and acts as a helper in the discussion, consensus and learning process.

From our daily use of the EVS prototype we found that more query types weren't as useful as a better stats or peer/tutor assessment will be [4][9][10][13]. We decided to work a little more in the educational features even if some expected features will be lost due to time restrictions.

Some unimplemented features appear with a label *almost done*. This label indicates that the required data is collected, even stored in database but the feature related to this data is not fully implemented.

Even with a lack on some features from the design the project offers a complete educational tool ready to be improved in the future with the unimplemented and new features coming from the user's experience.

Implementation CSCL reusability

Server side

The CSCL reusability is the key feature of the framework. All the objects are already implemented and we have only to fit our requirements into the *CSCL components*.

The standard attributes in each *CSCL component* acts as a *pattern* about the minimal data to be collected. The *extraData* generic attribute is a powerful commodity in order to get our customized data into the generic *CSCL* component. It makes the framework truly reusable and extensible.

Looking at the DF and the EVS sources we find that almost all is already done by the CSCL framework. In many occasions we reuse methods and objects from the DF. The DF database connection manager or the DF logger is reutilized in the EVS as many other features like the login and user session management.

More exactly all the EVS components: User, Security, Administration and Knowledge were already implemented in the DF from the CSCL framework. As for the functionality component only: *Pool*, *Awareness*, *Room* and *Contribution* needed new classes and methods to carry our application logic. All the others components were already implemented by the DF or the CLPL framework and were reused without changes.

Once we're familiar with the framework the development process is really improved and it make possible to create a complete project in a couple of weeks.

On the other hand it is difficult to become familiar with the framework and learning how to use it consumes almost 50% of the time dedicated to the implementation.

EVS implementation decisions

CLPL defines itself as a *generic framework* [2]. During the specification and design phases we were thinking about how to fit our EVS requirements into the CLPL objects.

It's the vote an *action*, a *contribution* or an *awareness*? From a design point of view it will be an *action*. But under the implementation point of view, the answer is: all of them should be fine.

During the implementation some doubts disappeared. Finally we took a pragmatic approach inspired by the DF source code. In our example we finally decided that the *query* is a contribution and the *vote* is an awareness related to the *query* contribution. Dealing with the implementation all the objects found their natural place. Even if a *vote* would be an *action*, the complexity of implementing the *vote* as an *action* compared with the complexity of

implementing it as *awareness* indicates that the *Action* object should be used by another kind of actions more complexes. Even if we do a deep study of the CLPL documentation during our specification and design, it was during the implementation when we really got the CLPL know-how.

Design implementation

During the design phase we design a complete standalone EVS application. All the CSCL classes were extended in order to keep clean the *CSCL framework* and all the new methods were created into different classes in order to maintain the code elegant, efficient and easy to extend or debug.

In the implementation phase we took another way. We implemented the EVS following the same implementation decisions as the DF does. Reusing the CSCL framework without any class extension to finally code all the new methods in a monolithic class as the DF does.

During the design we thought about a *query proposal* button to promote threads to the query space. Working with the EVS prototype we understood that the usefulness peer evaluation [4][10] already existed for each DF post and was used to give relevance to the contributions. We decided to use the post contribution usefulness as an indicator for the query proposal instead of creating the *query proposal* button. We are giving a new use from an existing value incentivizing from our EVS the usefulness evaluation in the DF.

Client side

The client side is implemented with PHP. PHP is an OOP language [25]. However it can be also used in terms of the structured paradigm. The DF implementation uses the structured paradigm and according to that we implemented the EVS client using structured programming instead of OOP.

The user interface is implemented in HTML. There is no database access in the client side. All the communication with the server is done by web services. The data is processed with PHP and presented to the user by HTML formatting capabilities.

There are some features in the OOP PHP that would be useful in our project. Actually we are working with PHP scripting, but we suggest considering an entire client revision and rewriting the client side from the OOP perspective.

Client - server communication

Note: See our *Annex II. SOAP Services Description* in order to know how to interact with our web services from any SOAP client.

As we told the generality is one of the CLPL key features and the *extraData* attribute is one of the most useful attributes in the framework. But there is an unexpected problem: *extraData* uses to be an array of multiple-type unnamed variables and all the data from *extraData* has to be accessed by numerical indexes.

We take a look of this code from the EVS implementation before to discuss the problem in detail.

This is part of the "*Create Query*" process. All the query creation parameters are put in an array in the client-side before being sent to the web service.

```
 $parameters_query = array('session'=>$session, 'op'=>$op, 'contributiondata'
=> array('room' => $room, 'idauthor' => $idUser, 'idgroup' => $idGroup,
folder' => $folder, 'qtitle' => $qtitle, 'qdescription' => $qdescription,
'qtype' => $qtype, 'qstart' => $qstart,  'qend' => $qend,  'query_1' =>
$query_1,  'query_2' => $query_2,  'query_3' => $query_3,  'query_4' =>
$query_4,  'query_5' => $query_5, 'qstate' => $qstate, 'qblock' => $qblock,
```

```
'qallowresults' => $qallowresults, 'qallowblank' => $qallowblank,
'qallowexplained' => $qallowexplained  ), 'z'=>array('area' => $area));

//The following line calls the web service.

$ret = $client_create_contribution->call('pickEVSContributionManagementData',
$parameters_query, $soapoptions_contribution);
```

Java service implementation signature for the UI class.

```
public java.lang.Object pickEVSContributionManagementData(java.lang.String
userSessionId, java.lang.String operationType, java.lang.String[] queryData,
java.lang.Object[] queryExtraData)
```

The **queryData** attribute manages all the *CSCLContribution* common data. All the specific data related to the query contribution goes to **queryExtraData**.

The call to the business layer service *queryCreation*

```
//identifier, interaction, location, room, author, date, description:
//title_query, content: description_query, QUERY_EXTRADATA_ARRAY

ret = port.queryCreation(userSessionId, null,   null, null, queryData[0],
queryData[1],  now, queryData[4], queryData[5], new Object[] { queryData[6],
queryData[2],   queryData[3],   queryData[7],   queryData[8],   queryData[9],
queryData[10],  queryData[11],  queryData[12],  queryData[13],  queryData[14],
queryData[15], queryData[16], queryData[17], queryData[18]  });
```

The following is the *queryCreation* signature

```
public java.lang.Object queryCreation(java.lang.String userSessionId,
java.lang.String identifier, java.lang.String csclInteractionId,
java.lang.String location, java.lang.String csclRoomId, java.lang.String
authorCsclUserId,  java.lang.String date,  java.lang.String description,
java.lang.Object content, java.lang.Object[] contributionExtraData)
```

Call for the data layer from the business layer.

```
saveQuery(userSessionId, contribution, contributionExtraData)
```

The *saveQuery* signature

```
public java.lang.String saveQuery(java.lang.String userSessionId,
dfws.clplws.functionality.contribution.data.CSCLContribution contribution,
java.lang.Object[] contributionExtraData)
```

And finally assigning the array data into named variables to facilitate the SQL INSERTs

```
String title = contribution.getDescription();
String author = contribution.getCsclUserAuthorId();
String description = (String)contribution.getContent();
String room = contribution.getCsclRoomId();
String date = contribution.getDate();
//extradata:
String querytype = (String)contributionExtraData[0];
String groupId = (String)contributionExtraData[1];
String folder = (String)contributionExtraData[2];
String start_date = (String)contributionExtraData[3];
String end_date = (String)contributionExtraData[4];
```

```
String query_1 = (String)contributionExtraData[5];
String query_2 = (String)contributionExtraData[6];
String query_3 = (String)contributionExtraData[7];
String query_4 = (String)contributionExtraData[8];
String query_5 = (String)contributionExtraData[9];
String state = (String)contributionExtraData[10];
String block_discussion = (String)contributionExtraData[11];
String allow_results = (String)contributionExtraData[12];
String allow_blank_vote = (String)contributionExtraData[13];
String allow_explained_answer= (String)contributionExtraData[14];
```

We are describing a recipe for disaster. Even if our client is totally independent from our web services our implementation is dependent to the attributes number and order in the client side.

What will happen if we send less than 13 extra data attributes? A *Java OutOfBound Fatal Exception* because the *allow_blank_vote* is in the position 13 of the *extraData*.

Something worse will happen if we change accidentally the order of some attributes in the client side implementation. We will send the right data to the wrong database field without noticing it.

OOP programming in the server side will be helpful but will not suffice if we don't use OOP in the client side. All the data has to be sent trough SOAP and SOAP can't manage objects so use of the OOP programming to keep the programmer free from remember attributes number and order is a must.

Proposed PHP OOP approach

One way to avoid this issue is creating a *Query* class in *OOP PHP5.* Use setters and getters for each attribute and finally a method to create the array before sending the data trough SOAP.

```
$query = new Query();
$query->session=$session;
$query->setIdentifier($idUser);
$query->setTitle($title);
$query->setDescription($descrption);
$query->addQuestion($question);
$query->setAllowBlankVotes($allow_blank);
$parameters_query = $query->getSOAPParams("create");
```

For sure, this may imply a performance effort but it will signifies an improvement in the use of the web services by further project collaborators. In our example the creation of the parameters is encapsulated in a method and is not the programmer responsibility to place each variable in the right place or to control the exact number of variables to send.

It would be verbose but useful under developer's point of view.

User Guide

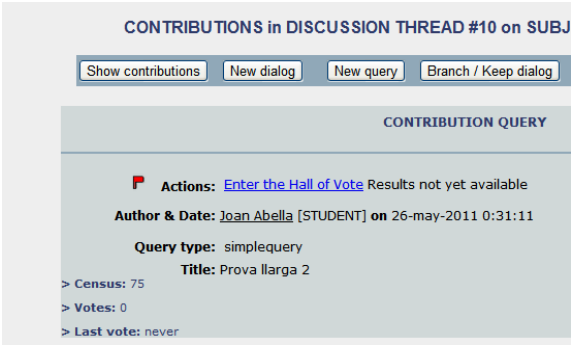
To gain access to the EVS we have to use a DF account. The DF is the responsible of users, groups, shared spaces and folders. Our EVS is a tool in the contribution forum; it's a helper, not a stand-alone application.

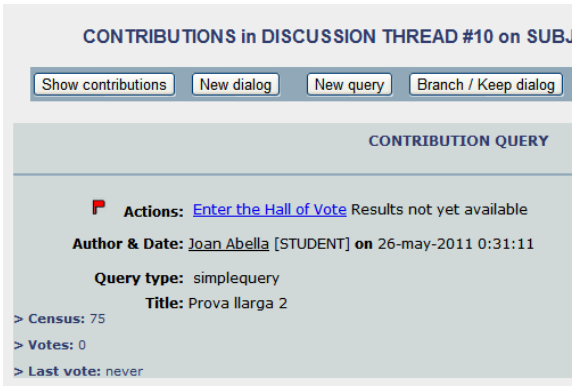
Once validated in the DF and after choosing the group and the course we enter into the application. The first screen is the ***Start*** screen.

In our start page we have the following information

1. A list of the queries in each folder.
2. Each query show the following information: Thread, author [role], title and number of votes in relation to the census.
3. The awareness flag shows the following:
    a. Green – The user voted the query. Not attention required.
    b. Red – The query remains pending to vote by the current user. Attention required.

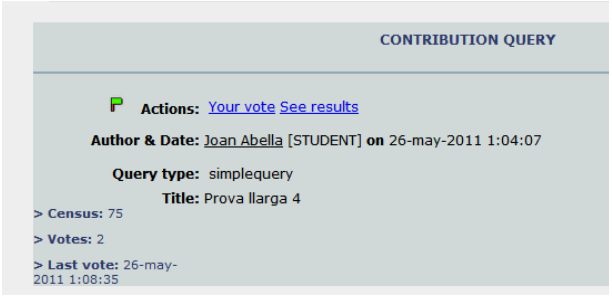
When clicking the link we are redirected to the DF containing the query. It is supposed that we will read the discussion before to vote and that's why we aren't redirect directly to the "*Hall of Vote"* from this link.





If the query is not already voted a red flag mark the query indicating attention. We have a link to "*Enter the Hall of Vote*" if we want to vote. If the query doesn't allow results before the end a message "Results not yet available" will communicate that fact.

It's needed to access the *Hall of Vote* if we want to know the options to vote. From this view we only give some general data as: author, creation data, query type, title, and description and statistics data about participation: census, actual number of votes and last-vote timestamp.



If the query is already voted we can see our vote. If the query is closed or the results are allowed before the query ending we can access to the actual query results.

In the *Start page* and in the *Voting System* page there is some awareness data in the side bar.

The last 5 new queries are shown to get a quick access to the last trend topics in discussion. This link redirects directly to the "Hall of query" to facilitate a quick response.

A list of the 5 last closed queries is also provided as a feedback in order to be aware of the queries results. This link redirects to the results page.

The Voting System menu link

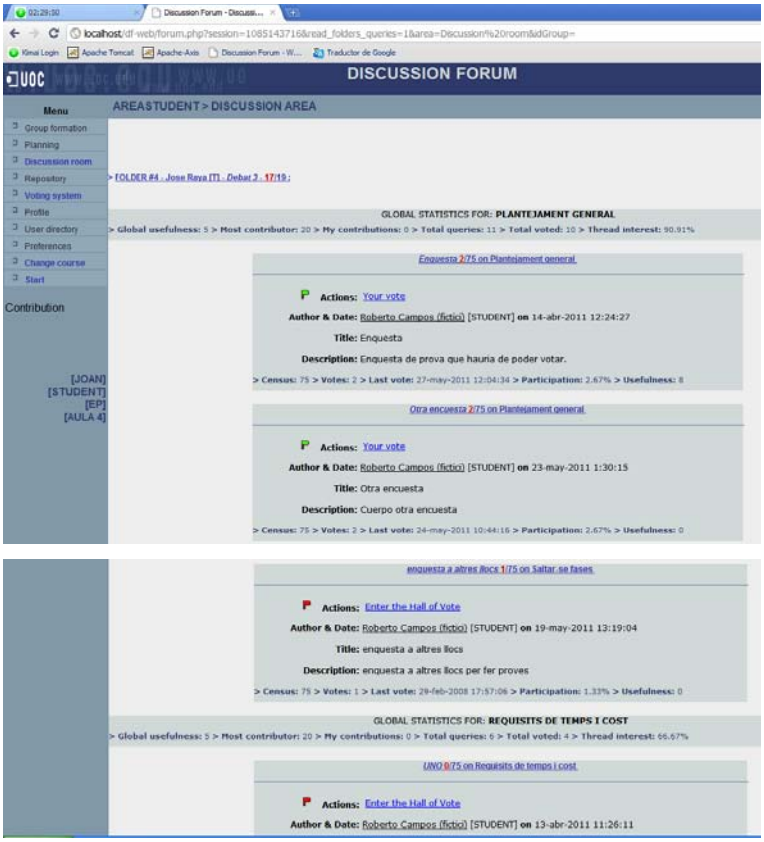
This is the EVS main page.

There is an exhaustive list of all the queries in the course classified by Folder and Thread.

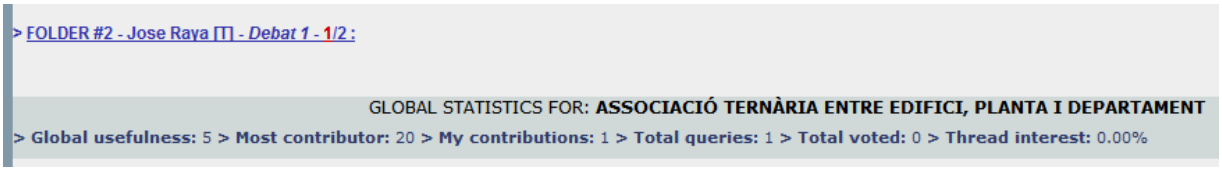
Each thread has a global statistics about the queries in the thread.

Finally each query has an action list including: *delete query, close query, consult your own vote* or *see the results*.

We discuss all that in detail below.

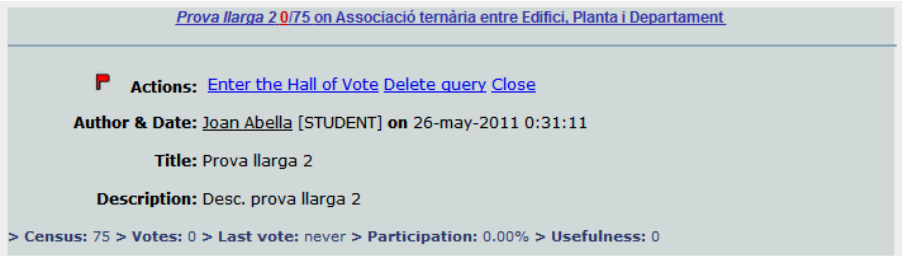


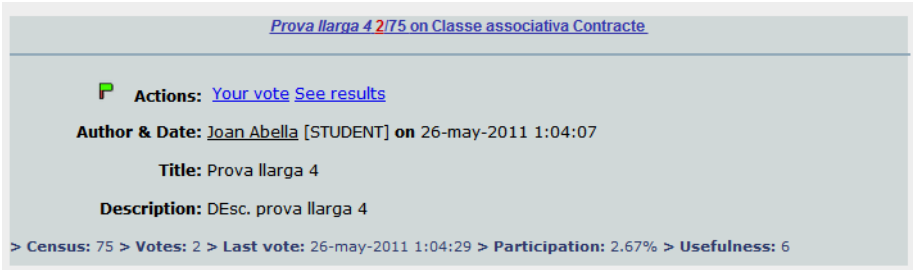
Global thread statistics



- The data *1/2* in the folder link description indicates that there are 2 queries in the thread but 1 remains with 0 votes.
- **Global usefulness**: is the result of the average peer assessment from all the queries in this thread.
- **Most contributors**: is the number of queries created by the user most active in the course.
- **My contributions**: is the number of queries the current user has created.
- **Total queries**: is the number of total queries in the thread.
- **Total voted**: is the number of queries in the thread voted by the current user.
- **Thread interest**: is the average of the index of participation from all the thread queries.

Query header description



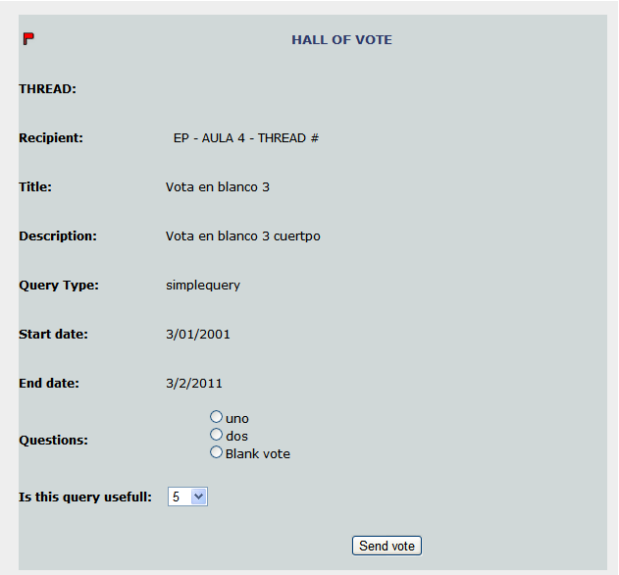


- The title link includes the name of the query, the number of current votes, the census and the name of the thread containing the query. The link redirect to the thread.
- The awareness flag indicates if the query was voted by the current user or not. Red = not yet voted. Green = Already voted.
- The possible actions are different and change under some conditions.
- To enter the hall of vote the query must remain not voted by the current user.
- To delete a query the query must have 0 votes.
- To close a query the current user must be the query creator.
- To see the self user vote, a vote must exist.
- To see the results the query must be close or the *allow results flag* must be defined in the query creation.

Hall of vote

The *Hall of vote* is the place to *vote* and to do a *usefulness* evaluation of the query. The vote is not erasable or editable. When emitted there is no way to change it or to vote again. The vote is only seen by the user who emitted it as a reminder.

If you want to *blank vote* on a query who allow this kind of vote there is an option to do it.

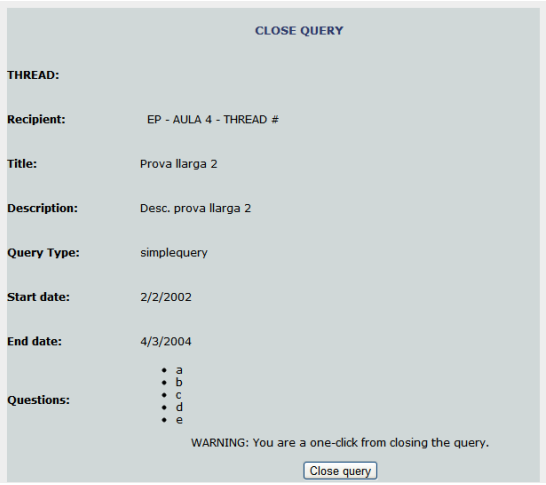


Delete query

Only the creator will delete a query. But there is no way to delete an already voted query. This action is provided as an "undo" action just in case we were posting accidentally an incorrect or incomplete query.

There is no way to delete a single question or to add a new question to an existing query.

We suggest double-checking before posting the query. Trying to never use the **delete query** would be a good practice.



Close query

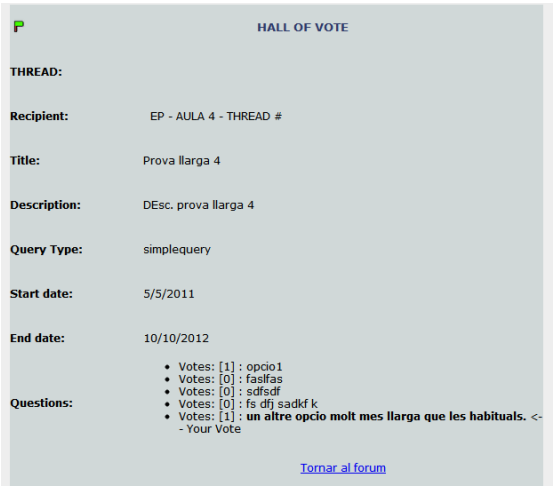
Actually even if the start and end date are recorded *there is not any automated function to close the query*. The query must be manually and intentionally closed by the author.

- A closed query doesn't allow voting it.
- A closed query allows seeing the results.

Your vote [vote reminder]

What we voted for a concrete query is shown with the *Your vote* action. It's also an indicator of how we changed our mind from the vote day until the final discussion day.

If we're voting another option nowadays the discussion is working. If you were voting the "winner" option, then the consensus is maybe not achieved, but certainly improved.



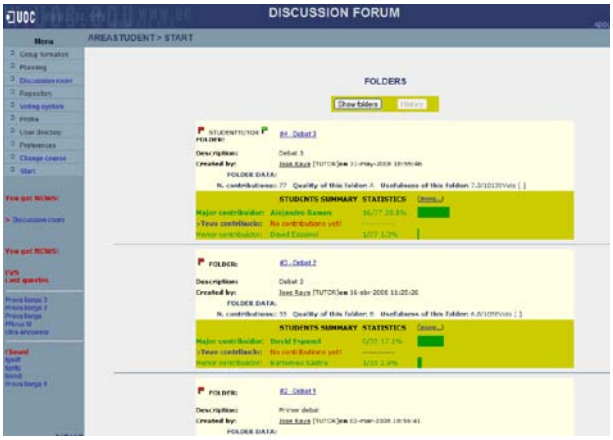
Query results

The query results focus on the key concepts from the student point of view.

- How many votes had every option?
- What do I voted?
- How many people were in the census?
- How many people voted the query?

EVS and DF integration

The DF stills working like before the EVS integration. Queries and votes will not appear under the DF stats. There are folders in the room, threads in the folder and contributions and now queries in the threads.



The thread view still unchanged.

The integration comes from inside the thread. A ***new query*** button allows the query creation in the current thread.

Queries will appear on the *top* of the discussion and a "*please start a new one*" will appear if there is not any query for the current thread.

New query form

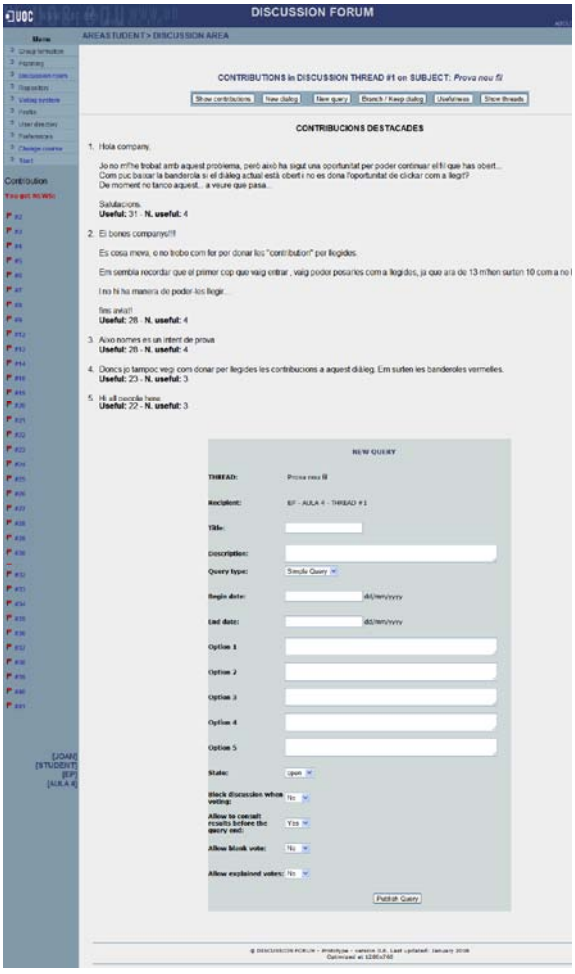
The new query form allows the user to easily introduce all the data related to the query in a single step. Each option is self descriptive. The query will be publicly accessible on the top of the current thread.

Before creating a new query a selection of the five better –usefulness- contributions will help the user to know what is happening in the thread and which the trend topics are.

As a trend or interest indicator, we show the usefulness and number of usefulness votes for each query proposal.

This way, the student can use an existent contribution and copy it to the query form. Or it can rewrite the contribution argument in another way or maybe decide a new query without any relation to the proposed contributions.

On the other hand, the usefulness activity and peer assessment got a new unexpected utility in the discussion forum being part of the EVS contributions selection process.



Form particularities

- If we want to create queries with less than 5 options is enough to leave the text area empty for other options.
- If we allow to blank vote a new option to send a blank vote will appear in the *Hall of vote*. It is not possible to send a true empty vote.
- If we will not vote the usefulness at the voting moment we can't do it later.
- We have a combo box with many different query types but only the simple query is implemented.

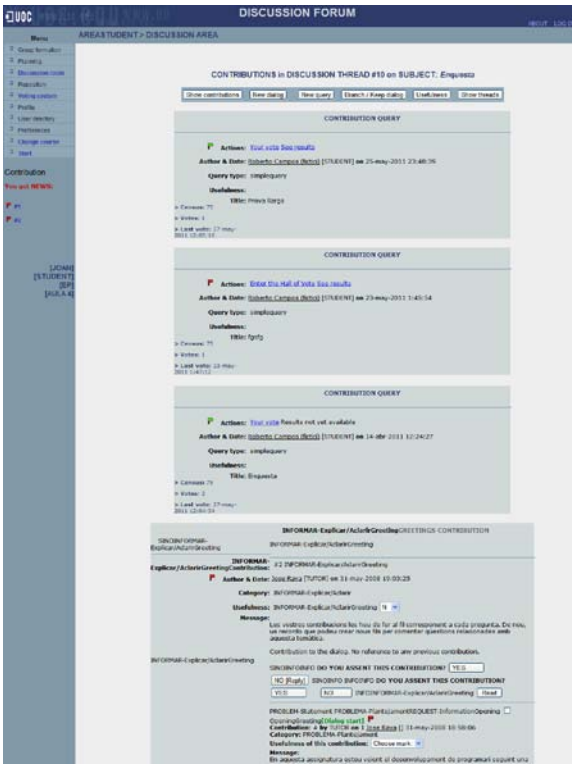
- The start and end data are informative and will not be used in any calculation at this moment.
- There is not any input validation implemented at this point.

There is the result of sending the new query to the DF thread.

The new query appears in the top of the DF thread's page.

If there are many queries, all of them are shown on the top of the thread. Mixing the queries and the DF contributions resulted in a mess of information in our local testing. Separating the EVS and the DF in two clear blocks we allow the user to keep attention in one thing each time.

However, some self-organization is needed to keep the thread comfortable.



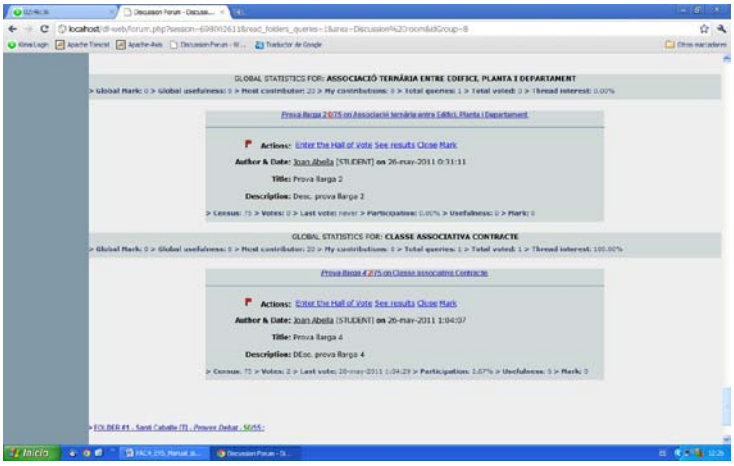
Tutor-only features.

The tutor shares all the features with the student. However, the tutor has got some special features in order to do a better evaluation.

In the start page the tutor has the same view as the student.

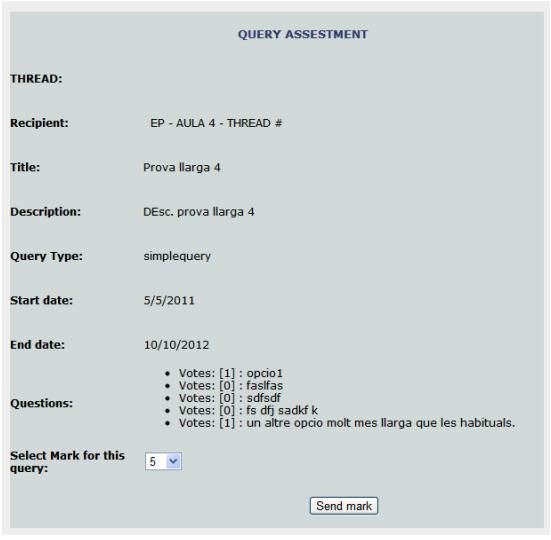
The *Voting system* menu link, shows the same view as the student one but there are two major differences.

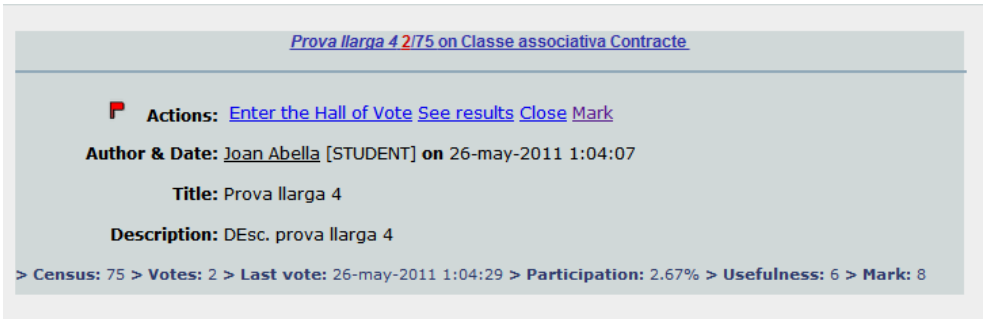
- The tutor can close any query.
- The tutor can assert the query.
- The assessment mark appears now in the threads and query statistics.



New assessment [Query mark]

In any moment the tutor will do an assessment on the query quality. If we want to change a mark another mark should be emitted and the previous mark will be overwritten.

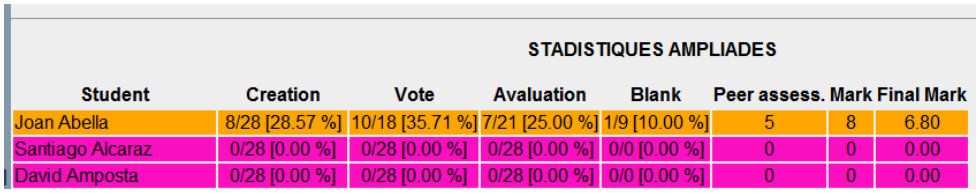




Extended tutor stats

The results from any query will be significantly extended in the tutor view. A complete list of all the students in the census and their respective stats –all but the vote itself- will be accessible for the tutor.

Tutor stats description



STADISTIQUES AMPLIADES

Student	Creation	Vote	Avaluation	Blank	Peer assess. Mark	Final Mark
Joan Abella	8/28 [28.57 %]	10/18 [35.71 %]	7/21 [25.00 %]	1/9 [10.00 %]	5	8	6.80
Santiago Alcaraz	0/28 [0.00 %]	0/28 [0.00 %]	0/28 [0.00 %]	0/0 [0.00 %]	0	0	0.00
David Amposta	0/28 [0.00 %]	0/28 [0.00 %]	0/28 [0.00 %]	0/0 [0.00 %]	0	0	0.00

Student is the complete name of the student. If the student has access to the course it should be in the list. The tutor is avoided from that list even if he can create queries, vote and receive peer evaluation.
Creation: number of queries created by the user/number of total queries created in the group room. The percentage appears as a commodity to further evaluate the user.
Vote: number of votes emitted by the user/remaining votes to emit. The percentage is about how many queries he has voted over to total existing queries.
Evaluation: number of peer evaluation by the user/remaining usefulness votes to emit. The percentage is about how many queries he has evaluated over to total existing queries.
Blank: how many blank votes the user has used/number of total votes the user emitted. The percentage is about how often he uses to blank vote instead of select a concrete option.
Peer assessment: Average usefulness from all the queries created by the user.
Mark: Average tutor mark from all the queries the user has created.
Final mark: 40% Peer assessment + 60% Tutor assessment. This final mark is nonbinding for the tutor.
Color code [based on final mark over 10**]**: Green >=8 ; 5<=Orange<8; Pink<5

Conclusions and further work

Even if e-learning is something common in these times there are not applications focused in the new e-learning paradigm in the marketplace. There are studies about how to adapt current internet applications to an e-learning context, but not studies about how to adapt current internet applications to fit the e-learning context.

We can find EVS systems, and we can find Virtual Leaning Environment (VLE) supporting EVS systems but we can't find any VLE doing the use of the EVS we have done in this project.

We are working from a new point of view, creating new tools for a new learning experience instead of adapting old fashion methodologies for a distance educational process.

Our study for an EVS application integrated with a DF in order to improve the whole educative discussion process is quite complete, exhaustive and well sustained with a complete list of selected references. Our implementation is a proof of concept and would need a new user interface design in order to offer a better experience to the user. HTML5, AJAX, DHTML and new web technologies will improve in the future our web experiences providing users with better and innovative ways to interact in an Internet context.

There are also some features suitable to include in the project but unfinished due to time limitations as more query types, open answers or discussion blocking during consultations.

Further EVS improvements will need a better client implementation and feedback from the students and tutors point of view. Even if it makes sense to continue improving and completing our project implementation trying to improve the EVS4CSCL project without user's feedback would be pointless. At this moment the application is quite usable and getting feedback from their users would indicate where more work is needed.

Beyond our EVS and DF and beyond any usability improvements further projects should examine other common web applications and create an innovative e-learning approach about them [36].

We have organizers and calendars, but how to know who is organized and who's not from an educational point of view. We have chats and personal interaction between our students but, how to know whose social and friendly and who is selfish.

How to improve organization and sociability in an e-learning environment? It's possible to offer in an e-learning environment the same guidelines and direct observation assessment we are bringing to our face-to-face students.

It's possible to effectively evaluate competences in an e-learning context. From our knowledge there are not public developments or studies working in this approach of how computers will support the collaborative learning in the future.

Helpfully we have the CLPL framework to investigate all kind of new applications providing master thesis students the facility to do a complete investigation and a quick proof of concept implementation of the application they want to create.

Even told these applications would not be fully implemented or deployed in production servers due to time limitations, the proof of concept elaborated will act as a starting point for more elaborated implementations based on a deep investigation and some testing already done.

From this point of view the CLPL framework it's an amazing tool in order to improve innovation, research and development in the e-learning area. It also offers to the UOC the key of the next-generation e-learning platforms.

Glossary and notes

EVS: Electronic Voting System. System allowing their users to vote using hardware or software tools in order to simulate a real vote process with electronic commodities as: automated vote counting and instant results and stats.
CSCL: Computer Supported Collaborative Learning. This term references any application facilitating collaborative learning in on-line, collaborative, environments. Discussion forum, collaborative electronic voting systems, wikis, calendars, etc.
CLPL: Collaborative Learning Platform. It's a framework for rapid CSCL application development created in the UOC. This framework acts as a base system for our entire EVS project.
Discussion phases: There are 3 main phases in a learning discussion process: specification, elaboration and consensus.
Peer evaluation: This evaluation comes from students evaluating other students' contributions.
Poll: It's a type of EVS allowing their users to make a choice between two or more options in order to make a winner option and know user's preferences about the topic. A poll doesn't imply discussion or consensus.
Survey: It's a type of EVS with the focus on the user itself. The survey intention is to know the user opinion about some topic or product. A survey doesn't cares about consensus or discussion.
Assessment Test: An assessment test is also an EVS application but there the focus is on the right answer. There is not discussion or consensus in the assessment test context.
Query: Is a consultation in the context of an educational discussion forum where the topics can be discussed and voted. From our educational point of view the query has 3 basic purposes: initiate the discussion, test for consensus and test for educational lacks.
Audience response systems (ARS): this term refers specifically to hardware components to do EVS with a face-to-face audience.
Virtual Learning Environment (VLE): is a system designed to support teaching and learning in an educational setting.
Collaboration: All those functional aspects that support information sharing between several users.
Communication: All those functional aspects that support communication between two or more human participants.
Component library: set of software resources related to each other addressed at the programmer who makes a great saving in terms of effort and cost during the construction of a computer system. The library shows an interface that enables the programmer to find, understand and use the components.
Contribution: In a shared environment, collaboration in which a group member takes part providing some kind of useful information (ideas, references, etc.) so as to improve the group task at hand.
Coordination: The organization and monitoring of well-defined tasks and users as resources to accomplish a final goal.
Discussion Forum (DF): is an internet application created to support discussion.
EVS action: each of the possible operations on an EVS resource (e.g. to create or eliminate a query, vote, etc.).
EVS administrator: EVS user carrying out the computer system management so as to control, update and improve the system performance.
EVS component (of software): software piece created with the clear aim of being reusable during the construction of an EVS computer system. Its qualities are reliability, robustness, efficiency and the independence provided by its interface. If the purpose of the component is to be generic then it has a much greater impact on software construction given that it can

form part of the foundations of a huge number of EVS applications which share this generic behavior.

EVS coordinator: group member as an EVS user who acts as the EVS group leader full- or part-time during the accomplishment of a task or phase. His/her task is to organize the group (e.g. to divide the tasks to be made among the group members) and the resources belonging to the group.

EVS group: cooperative learning group made up of students aiming to accomplish a specific learning objective.

EVS participant: student as a member of a collaborative learning group.

EVS peer: Each of the participants in a collaborative learning group. This term also represents other system users to whom the participant is personally related.

EVS resource: all the objects participating in a collaborative learning environment such as voting systems, calendars, bulletin boards, folders, workspaces, users and so on.

EVS subsystem: software piece as a result of the component decomposition into smaller pieces that are sufficiently autonomous as to be designed separately but sufficiently dependent as to satisfy an interface with other subsystems.

EVS tutor: person as a EVS user (e.g. a teacher, lecturer, etc.) who organizes the groups (e.g. splitting them up, distributing skills, etc.) and plans the tasks, monitors the workspaces, provides help, support and encouragement to the groups and tracks and assesses the groups' work as well as the learning outcomes of the participants.

EVS user: person or group of people who interact with the EVS application. Examples of users are: a student as a group member, a group coordinator, the teacher/lecturer who monitors the group, the system administrator, a group-entity representing the *room* abstraction (i.e. workspace) where the group members meet.

Feedback: Capability provided by the system with which is possible to influence online learning groups providing them with detailed knowledge about members' interactions, problem-solving processes and scaffolding. The aim of providing these selected feedback mechanisms is to have a positive impact in terms of motivation, interaction, problem-solving abilities, and learning.

Interaction: An interaction is an abstraction of communication representing any kind of piece of information exchanged between a sender and one o more receivers. An interaction can be a text message, a spoken communication, a gesture in a videoconference, a brushstroke in a shared drawing editor, etc. In a collaborative learning environment, it is the collaboration process in which a group member attempts to present, explain, or illustrate new knowledge (ideas, references, etc.) to their peers while their peers attempt to understand and assimilate the new information.

Query management: is the set of activities related to the query administration. Create, delete, update, consult, block.

Query results: is a resume of the data generated by the query. Query results include final vote count, but not only vote count, also some feedback and awareness data useful to put the query in context.

Question: is an option to be choose in order to answer a query.

Sharing: Capacity of a system to enable users to exchange among them same or different copies of the same resources.

System's knowledge: All the information of the system provided by the system's monitoring.

User awareness: Capability provided by the system with which it is possible for the user to have knowledge about what is going on in the system.

User state: A system's actor can be found in any state: active, not available, disconnected, etc.

Vote count: is the number of each answer we get over a query.

Vote: is the answer to a query provided by an user.

References

Papers

[1] Caballé, S., Xhafa, F. (2009). Fostering Collaborative Knowledge Building by the Effective Provision of Knowledge about the Discussion Process. Special Issue on: "Advances in Intelligent Information Management Systems and Applications". International Journal of Business Intelligence and Data Mining (IJBIDM), Vol. 4, No. 2, pp. 141-158. Inderscience Publishers. ISSN: 1743-8187.
[2] Caballé, S., Xhafa, F. (2010). *CLP*L: Providing Software Infrastructure for the Systematic and Effective Construction of Complex Collaborative Learning Systems. Journal of Systems and Software (JSS). Vol. 83, No. 11, pp. 2083 - 2097. Elsevier. ISSN: 0164-1212.
[3] Experience of online communication and collaboration from the undergraduate experience of blended e-learning: a review of UK literature and practice. Rhona Sharpe, Greg Benfield, George Roberts, Richard Francis. [63-65]. The Higher Education Academy – October 2006
[4] Formative peer assessment in a CSCL environment: a case study. Frans J. Prins*, Dominique M. A. Sluijsmans, Paul A. Kirschner and Jan-Willem Strijbos. Open University of the Netherlands, The Netherlands (2005).
[5] A.Bruce, University of Edinburgh. January 2005. *EVS: a catalyst for lecture reform*

On-line references

[6] UOC annual report 2009/10 [WWW document]. URL. http://www.uoc.edu/portal/_resources/CA/documents/memories/0910/sintesi-memoria-0910.pdf (visited 2011 March)
[7] Draper,S.W. (2002, Jun 9) Electronically enhaced classroom interaction [WWW document]. URL http://www.psy.gla.ac.uk/~steve/ilig/handsets.html (visited 2011 March)
[8] Draper,S.W. (2008, Set 19) Pedagogical formats for using questions and voting [WWW document]. URL http://www.psy.gla.ac.uk/~steve/ilig/qpurpose.html#Designing (visited 2011 March)
[9] Draper,S.W. (2008, Set 19) Feedback to students [WWW document]. URL http://www.psy.gla.ac.uk/~steve/ilig/feedback.html (visited 2011 March)
[10] Draper,S.W. (2008, Set 19) Peer instruction [WWW document]. URL http://www.psy.gla.ac.uk/~steve/ilig/peertable.html (visited 2011 March)
[11] Alan Aycock, Carla Garnham, Robert Kaleta (March 2002). Lessons Learned from the Hybrid Course Project [WWW document]. URL http://www.uwsa.edu/ttt/articles/garnham2.htm (visited 2011 March)
[12] List of 25 Electronic Voting Systems. URL http://www.dmoz.org/Society/Politics/Campaigns_and_Elections/Electronic_Democracy/Electronic_Voting_Systems/ (visited 2011 March)
[13] Computer Assisted Assessment Center (1998) Objective Test Design [WWW document]. URL http://www.caacentre.ac.uk/resources/objective_tests/index.shtml (visited 2011 March)
[14] Consensus Decision Making. URL http://seedsforchange.org.uk/free/consens (visited 2011 March)
[15] Papers on Cooperative Decision-Making. URL http://www.vernalproject.org/papers/Process.html (visited 2011 March)
[16] Paul Johnson. Voting Systems. University of Kansas. (2005). URL http://pj.freefaculty.org/Ukraine/PJ3_VotingSystemsEssay.pdf (visited 2011 March)
[17] Simon P.Bates, Karen Howie and Alexander St. J. Murphy. University of Edinburg. The use of electronic voting systems in large group lectures: challenges and opportunities. http://www.ph.ed.ac.uk/elearning/publications/evsfinal.doc (visited 2011 March)
[18] Robin H. Kay, Ann LeSage. Examining the benefits and challenges of using audience response systems: A review of the literature. University of Ontario Institute of Technology. January 2009. http://portal.acm.org/citation.cfm?id=1570538.1570686 (visited 2011 March)

[19] URL. http://en.wikipedia.org/wiki/Consensus_decision_making (visited 2011 March)
[20] URL. http://www.roseindia.net/webservices/why_webservices.shtml (visited 2011 March)
[21] URL. http://wiki.apache.org/ws/FrontPage/Axis/AxisGeneral (visited 2011 March)
[22] URL. http://www.postgresql.org/ (visited 2011 March)
[23] Postgresql documentation – Cast.
http://www.postgresql.org/docs/8.4/static/sql-createcast.html (visited 2011 June)
[24] Postgresql documentation – Create and restore backups
http://www.postgresql.org/docs/8.1/static/backup.html (visited 2011 June)
[25] PHP5 on-line manual. OOP in PHP5.
http://php.net/manual/en/language.oop5.php (visited 2011 June)
[26] Apache Tomcat documentation. Timeout how to.
http://tomcat.apache.org/connectors-doc/generic_howto/timeouts.html (visited 2011 June)
[27] Lime survey. The open source survey application.
http://www.limesurvey.org/ (visited 2011 June)
[28] Postgresql security information home.
http://www.postgresql.org/support/security (visited 2011 June)
[29] Use Moodle with and EVS a change in mindset.
http://optivote.wordpress.com/2010/08/19/using-moodle-with-an-evs-a-change-in-mindset/ (visited June 2011)
[30] Optivote hardware EVS devices manufacturer. http://www.optivote.co.uk/ (visited June 2011)
[31] Virtual Learning Environment (VLE).
http://en.wikipedia.org/wiki/Virtual_learning_environment (visited June 2011)
[32] Jacobe JAVA code beautifier.
http://www.tiobe.com/index.php/content/products/jacobe/Jacobe.html (visited June 2011)
[33] PHPFormater. PHP code beautifier.
http://beta.phpformatter.com/Features/

Books

[34] Colleen Garton, Erika McCulloch- Fundamentals of Technology Project Management – 2004- ISBN: 1-58347-053-0
[35] D. Held - Models of Democracy (3rd ed.), Polity Press, Stanford, CA, 2006. ISBN: 0-74563-147-9
[36] Guy Merchant (Sheffield Hallam University, UK). Learning for the Future: Emerging Technologies and Social Participation. Pages: 2239-2251 pp. ISBN: 1-60566-984-9
[37] Justin Clarke - SQL Injection Attacks and Defense – 2009 – ISBN: 9781597494243
[38] Dafydd Stuttard - The Web Application Hacker's Handbook -2009 – ISBN: 9781118026472
[39] Ka Iok 'Kent' Tong - Developing Web Services with Apache Axis –2005– ISBN: 978-1411670327
[40]Tutorial de servicios web basado en Java y Axis – UOC - Ingeniería del software de componentes y de sistemas distribuidos
[41]Ignacio Lamarca, José Ramón Rodríguez -Metodologia de gestió de projectes TIC- UOC classroom materials.
[42]Discussion Forum – User Guide from the DF Project. UOC classroom materials.

Annex I. EVS4CSCL Installation.

Installing the base software in windows is easy due to the facility finding our concrete versions but there are many problems that may appear and it's better to avoid them at the beginning.

Check out our references about Apache Axis [39] [40] for a detailed installation description. We are discussing here all that is not detailed enough in the documentation.

It's usual to find in Internet many one-click "*Apache+Postgres+Tomcat+PHP+MySQL+PEAR*" compilations (*xampp*, *wampp*, *bitnami*, etc.). From our experience it's more difficult to get Axis working in these special installations.

Installing each requirement as stand-alone application, we would need to do some manual configuration to connect all the applications. If we're using a standard windows setup the installer will be helpful and the manual configurations will be significantly reduced.

To avoid problems with long folder names and spaces in folders we suggest installing all in the root of the disk. Another thing to do is to rename the folders by their equivalent without spaces.

There are the environment variables in our system. Adapt it to fit your own configuration.

1. AXIS_HOME = C:\axis
2. AXIS_LIB= %AXIS_HOME%\lib
3. AXISCLASSPATH= %AXIS_LIB%\axis.jar;%AXIS_LIB%\axis-ant.jar; %AXIS_LIB%\commons-discovery-0.2.jar; %AXIS_LIB%\commons-logging-1.0.4.jar; %AXIS_LIB%\jaxrpc.jar; %AXIS_LIB%\saaj.jar;%AXIS_LIB%\log4j-1.2.8.jar; %AXIS_LIB%\xml-apis.jar; %AXIS_LIB%\xercesImpl.jar;%AXIS_LIB%\activation.jar;%AXIS_LIB%\mailapi.jar; %AXIS_LIB%\wsdl4j-1.5.1.jar; **%AXIS_LIB%\postgresql.jar**
4. CATALINA_HOME = C:\Archivos de programa\Apache Software Foundation\Tomcat 6.0
5. JAVA_HOME = C:\Archiv~1\Java\jdk1.6.0_12
6. CLASSPATH= .;%JAVA_HOME%\lib\tools.jar;**%AXISCLASSPATH%;C:\Archivos de programa\Apache Software Foundation\Tomcat 6.0\webapps\axis\WEB-INF**
7. TOMCAT_HOME = C:\Archivos de programa\Apache Software Foundation\Tomcat 6.0
8. PATH = %SystemRoot%\system32;%SystemRoot%; **C:\Archivos de programa\Java\jdk1.6.0_12\bin; C:\php;**

We put special attention to:

- Adding the JAR's to the CLASSPATH or AXISCLASSPATH is not enough. It's needed to copy the JAR's into de AXIS_LIB folder.
- Line 3: Add postgresql.jar to the AXISCLASSPATH and the AXIS_LIB folder.
- Line 6: don't forget to add the dot (.) at the start of the CLASSPATH or you will need to write the complete path in order to compile a class in the current folder.
- Line 6: remember to add the Tomcat webapss\axis\WEB-INF folder to the CLASSPATH
- Line 8: Adding the java bin to the path is needed in order to compile from MS-DOS batch files.
- Line 8: Some Apache installations would need the PHP root folder in the PATH.
- Remember that *php.ini*, *httpd.conf* and *server-minimal.xml* will need some manual configuration relative with our local installation.

EVS application folder structure

Unzip *df-web.zip* into the Apache *htdocs* directory [*C:\htdocs\df-web*] and the *SOAP* directory also into the root of *htdocs* [*C:\htdocs\SOAP*]

Note: it's possible, but we can't recommend it, to use the same directory as Apache and Apache Tomcat Home. We suggest using a different home directory for each server in order to avoid confusion between the client and the server side when we're developing.

The client EVS application should be now accessible trough

```
http:\\localhost\df-web\index.php
```

Unzipping *df.zip* we will get four directories and some batch files: *dist*, *src, wsdl, evsdb and many *.bat* files.

Copy the *dfws* directory from *src* into the Apache Tomcat WEB-INF
[*C:\minipracticaCLPL\home_apache\WEB-INF*]

Copy all the JARs from *dist* into the Apache Tomcat WEB-INF\LIB directory.
[C:\minipracticaCLPL\home_apache\WEB-INF\lib]

Copy the directory *wsdl* into the Apache Tomcat WEB-INF directory.
[C:\minipracticaCLPL\home_apache\WEB-INF]

Copy the *batch* files into the Apache Tomcat WEB-INF directory.
[C:\minipracticaCLPL\home_apache\WEB-INF]

Using the batch files

Each batch file will compile a complete component. Each batch file is supposed to be started from console directly typing the batch name (we will not be able to see the error trace if there is anyone doing a double click from the interface). As for the Tomcat installation path it will be it different in every personal installation. Use a text replace tool to change it in all the batch files before compiling the sources.

Use the .bat files in order. Launch Name1.bat before Name2.bat if we're in a new installation in order to deploy the services in Axis. If you already have a DF installation working using the Name2.bat to compile and copy the JAR's will be enough.

It's not enough to copy the JAR's to the Tomcat directory ["C:\Archivos de programa\Apache Software Foundation\Tomcat 6.0\webapps\axis\WEB-INF\lib"]. The deployment must be done with the right tools and files.

This is an excerpt from the *functionality1.bat*, a bath file to publish the web services trough Axis commodities by using *wsdl* files.

```
java org.apache.axis.wsdl.WSDL2Java --server-side
--skeletonDeploy true -p clplws.functionality.pool.business
wsdl/clpl/functionality/business/CSCLPoolManagement.wsdl

java org.apache.axis.wsdl.WSDL2Java --server-side
--skeletonDeploy true -p clplws.functionality.pool.data
wsdl/clpl/functionality/data/CSCLPoolDiskManager.wsdl
```

This is an excerpt from the *functionality2.bat,* a batch to compile and deploy a functionality pool component.

```
javac .\dfws\clplws\functionality\pool\business\*.java

jar cvf .\lib\EVSCSCLPoolManagement.jar
.\dfws\clplws\functionality\pool\business\*.class

copy .\lib\EVSCSCLPoolManagement.jar "C:\Archivos de programa\Apache
Software Foundation\Tomcat 6.0\webapps\axis\WEB-INF\lib"

java org.apache.axis.client.AdminClient -p 8085
.\dfws\clplws\functionality\pool\business\deploy.wsdd

javac .\dfws\clplws\functionality\pool\data\*.java

jar cvf .\lib\EVSCSCLPoolDiskManager.jar
.\dfws\clplws\functionality\pool\data\*.class

copy .\lib\EVSCSCLPoolDiskManager.jar "C:\Archivos de programa\Apache
Software Foundation\Tomcat 6.0\webapps\axis\WEB-INF\lib"

java org.apache.axis.client.AdminClient -p 8085
.\dfws\clplws\functionality\pool\data\deploy.wsdd
```

Comments

We are working in the directory *c:\minipracticaCLPL\home_apache\WEB-INF* but there is not the Tomcat WEB-INF directory. Actually the Tomcat WEB-INF directory is *C:\Archivos de programa\Apache Software Foundation\Tomcat 6.0\webapps\axis\WEB-INF\lib* and that's why all the JAR's generated are copied to this destination.

We are following this name convention:

- EVSCSCLComponentLayer.jar indicates a new EVS class in the component. It includes also the DF classes for commodity reasons.
- DFCSCLComponentLayer.jar signals a DF class in the component but any EVS class.
- CSCLComponentLayer.jar indicates that this component is not used by DF or EVS.

We can use any name for the JAR files. Actually the JAR filename is not important because all the EVS and DF services are provided under the same CSCL web services interface. There is no

relation between the JAR name and the web service name. The service *urn:name* comes from the *wsdl* file but we leave it unchanged.

As an example EVSPoolManagement.jar implements services from CSCLPoolManagement.wsdl

The *urn:CSCLPoolManagement* ws interface bring to us the following services:

EVS: poolsEVSConsultation

DF: treatPoolData, poolsConsultation

CSCL: startPoolInterface, closePoolInterface, startPoolDiskManager, closePoolDiskManager, poolCreation, poolExtraction, poolsElimination.

Consider naming our EVS new generated JARs with the same name as your existent JAR's or you'll get class duplications from different JARs providing the same classes and it will become difficult to debug some unexpected duplicate services. In example, the *DFPoolManagement.jar* implements the same classes and methods as the *EVSPoolManagement.jar* but the *poolsEVSConsultation*.

The classes from the DF or the CSCL weren't changed by the EVS project, but you may be installing an older version of some classes if we were already improving the DF.

Setting up the database

We can find the database scripts in the folder *evsdb* folder from df.zip

- From scratch: create new DF & EVS installation with dumb data. Some dumb data is not correct and comes from internal test, used only for demonstration purposes.
    - `psql database_name < evs.sql`
- From current DF installation: create only the new tables and sequences without any data.
    - 1st. Copy the content from *sequences_evsdb.txt* to the PostgresQL SQL Manager and execute it.
    - 2nd. Copy the content from tables tables_evsdb.txt to the PostgresQL SQL Manager and execute it.
    - 3rd. Add an empty row with id = 0 in all the new tables. We were told to do that by the advisor as some PostgresQL 8.1 particularity relative to sequences.

Setting up the database connector

The database connector is hard coded. There is a method to read a configuration file and it seems to be perfectly done but in fact we were unable to connect to the database using the configuration file. Probably a Java version issue due to improved security about allowing the JVM to access local files, but we can't figure out the solution and finally we took the decision to do it in the source code.

Consider changing the connection string into *DFAuthenticationManagement*.

Edit the file from Apache Tomcat WEB-INF directory C:\minipracticaCLPL\home_apache\WEB-INF\dfws\clplws\security\autentication\business*DFAuthenticationManagement.java*

Search the following string and adapt it to your current configuration.

```
static final String[] CONNECTION_DATA = new String[]{
"org.postgresql.Driver","jdbc:postgresql:DATABASENAME",
```

```
"DATABASEUSER", "DATABASEPASSWD"};
```

The database user should be “postgres” and the password is your password to access postgres (we were asked for it during the installation process). The database name is the name of the database we’re using for the DF.

Apache2.2 costumization

Edit the apache *httpd.conf* from your apache2 configuration folder

C:\Archivos de programa\Apache Software Foundation\Apache2.2\conf

and define the right *DocumentRoot*. We are using C:\htdocs as our home for the PHP client side.

```
DocumentRoot "C:\htdocs"
```

Remember to copy the SOAP library from PEAR into this directory. A copy of the SOAP library has been provided with the df-client.zip

Restart the server in order to allow the changes to take effect.

Tomcat customization

Our web services (ws) are configurated to call services through the Tomcat *port 8085* instead of the default *port 8080*.

Locate the file Tomcat_HOME\conf and edit the file *server-minimal.xml*

Change the line *<Connector port="8080" />* with this one *<Connector port="8085" />* and **restart the server**.

Note: each time we do any change in configuration or in the JAR files we need to restart the server. Each time the server is restarted all is reloaded, configuration and JAR’s. Code changes take effect after the reload as the same as new services that become public after the server restart.

About localization

We decided to implement another localization implementation in order to add portability to windows and promote an easier way to localize further developments.

The function *gettext()* as an alias *_()*. When we want to add a string to be translated we do that: `_(“Translate this string”);`

In order to remain compatible with *gettext()* we are going to overwrite it with an associative array *$language* were the translate string is the key and the value of the array for that key is the string translated to a desired language.

en.php

```
$language[“Translate this string”]=”Translate this string”;
```

ca.php

```
$language[“Translate this string”]=”Tradueix aquesta cadena”;
```

Final considerations

Getting working the EVS from scratch is not easy. The better approach is to get it step by step.

- There is a tutorial from the UOC courses ”Tutorial de servicios web basado en Java y Axis” [5] who is helpful starting an Axis server configuration.
- In order to prepare our local environment the advisor gave us a reduced version of the DF project called “MinipracticaCLPL” who helped getting all the requirements working. It was only the login service, but if we can login then we have a working installation.
- Even with the help provided by the advisor and the tutorials it was difficult to get the final DF project working. This document includes all the information from the both documents above and all the new information and tips we collected dealing with the project.

Getting working the EVS from a current DF installation should be easy:

- Copy all the new EVS JAR files to Tomcat WEB-INF directory and restart the server. We have only 11 JAR files related to our EVS all of them from the functionality components: *contribution*, *room*, *awareness* and *pool*. Each EVS*.jar is equivalent to the previous DF*.jar but with a new EVS classes inside. It’s pretty safe to delete the equivalent DF*.jar files, redeploy all the services and restarts the server to get the new web services on-line.

  Note: our EVS shares the connection manager with the DF. If we have a working DF there is no reason to do any change in the database configuration. But if we want to test the EVS in a new database then a change is needed.

- Rename the *df-web* to *evs-web* if you don’t want to risk your previous *df-web* installation and copy it to your Apache *htdocs* directory.

  Note: There will be problems with the localization files. Under windows there is only one language: English. Catalan is only partially translated for demonstration purposes. Under Linux the localization would be lost as an undesired side effect. Search in files *.php for our *_()* function and remove it. After that regenerate the translation strings, uncomment the specific LOCALE variable definitions in the .php files and the original *gettext()* should work again. On the other hand if we’re comfortable with the new localization approach we can use our localization in Linux.

Annex II. SOAP services description.

Awareness

http://$wsHost:$wsPort/axis/services/CSCLAwarenessInterface		
UI	**Bussines**	**Data**
askEVSAwarenessData	awarenessEVSUpdate	readEVSAwarenessData
This service retrieves information about awareness according to **type** values. Returns an array of data in format key -> value according to the **type** requirements. All the values from this array are related to awareness and statistics data.		
Parameters to interact with the *askEVSAwarenessData* user interface.	$parameters_awareness = array('session' => $session, 'awarenessdata' => array('type' => $type, 'spaceid' => $spaceVisitedId, 'iduser' => $idUser, 'group' => $idGroup, 'spaceVisited' => $spaceVisited));	
Type may be one of the following values	TUTORSTATS, HALLOFVOTE, MENUFOLDER, START, THREAD, FOLDER, FOLDERMENU	

http://$wsHost:$wsPort/axis/services/CSCLAwarenessInterface		
UI	**Bussines**	**Data**
pickEVSAwarenessManagementData	awarenessCreationEVS	saveAwarenessEVS
This interface is called from the application, not from the client-side. Returns true or false in case of some query has been voted or not by the current user. It's used as a helper by other services-		

Pool

http://$wsHost:$wsPort/axis/services/CSCLPoolInterface		
UI	**Bussines**	**Data**
	poolsEVSConsultation	readEVSPoolData
This service is not called from the client-side. It acts as a helper for other server-side services. *RoomsEVSAllConsultation* relies in this service to get statistical data about each room and folder.		

Contribution

http://$wsHost:$wsPort/axis/services/ CSCLContributionInterface		
UI	**Bussines**	**Data**
pickEVSContributionManagementData	queryCreation, queryModification	saveQuery, modifyQuery
This service stores the data relative to a Query or update data relative to a query according to **type** values		
Parameters expected by the service *pickEVSContributionManagementData* We show two options: the query creation and the	**$create_query** = array('session' => $session, 'op' => $op, 'contributiondata' => array(	

vote creation. This example illustrates the problem of having generic services operated by a "type" argument instead of separated services with well know and concrete parameters. However a larger number of web services would affect the server performance and we do our project this way in spite of the problems derivate from this decision. **$ vote** = array('session' => $session, 'op' => $op, 'contributiondata' => array('room' => $room, 'idauthor' => $idUser, 'idgroup' => $idGroup, 'folder' => $folder, 'qtitle' => $qtitle, 'qdescription' => $qdescription, 'idquestion' => $idquestion, 'quseful' => $quseful), 'z' => array('area' => $area));	'room' => $room, 'idauthor' => $idUser, 'idgroup' => $idGroup, 'folder' => $folder, 'qtitle' => $qtitle, 'qdescription' => $qdescription, 'qtype' => $qtype, 'qstart' => $qstart, 'qend' => $qend, 'query_1' => $query_1, 'query_2' => $query_2, 'query_3' => $query_3, 'query_4' => $query_4, 'query_5' => $query_5, 'qstate' => $qstate, 'qblock' => $qblock, 'qallowresults'=> $qallowresults, 'qallowblank' => $qallowblank, 'qallowexplained'=> $qallowexplained), 'z' => array('area' => $area));
Type may be one of the following values	CREATION, MODIFICATION, DELETE, VOTE, MARK, CLOSE

http://$wsHost:$wsPort/axis/services/CSCLContributionInterface		
UI	**Bussines**	**Data**
askEVSQuery	queriesHallConsultation uses awarenessCreationEVS	readQueryHall
This service retrieves information about all the data to be present in the Hall of Vote. It includes calls to the *awarenessCreationEVS* service in order to know if the query has been already voted or not.		
Parameters to interact with the *askEVSQuery* user interface.	$parameters_read_hall = array('session' => $session, 'contributiondata' => array('type' => $type, 'thread' => $room, 'iduser' => $idUser, 'idgroup' => $idGroup, 'subject' => $subject, 'area' => $area, 'role' => $role, 'contribution' => $contribution));	
Type may be one of the following values	TUTORSTATS, HALLOFVOTE, MENUFOLDER, START, THREAD, FOLDER, FOLDERMENU	

http://$wsHost:$wsPort/axis/services/CSCLContributionInterface		
UI	**Bussines**	**Data**
askEVSContributionData	queriesConsultation	readQueries
This service retrieves the header information about all the queries in the room (thread). It returns an array with all the data for each query in the thread.		
Parameters to interact with the *askEVSContributionData* user interface.	$parameters_read_queries = array('session' => $session, 'contributiondata' => array('type' => $type, 'thread' => $room, 'iduser' => $idUser, 'idgroup' => $idGroup, 'subject' => $subject, 'area' => $area, 'role' => $role, 'contribution' => $contribution));	

Room

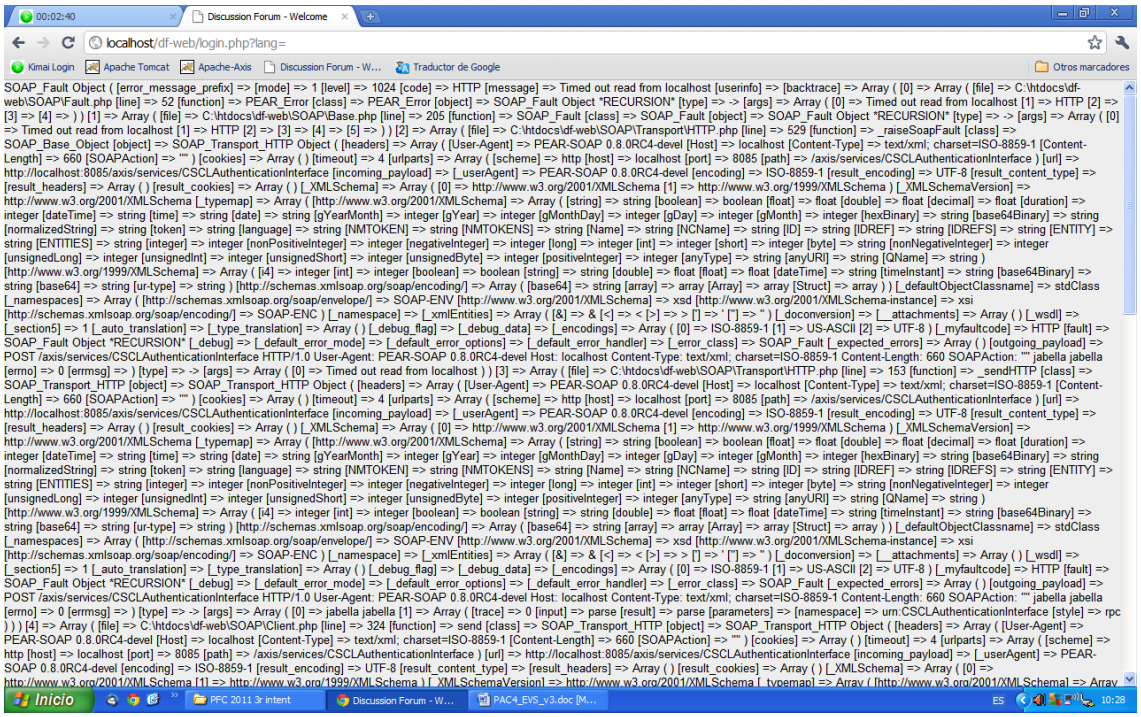
http://$wsHost:$wsPort/axis/services/CSCLRoomInterface		
UI	**Bussines**	**Data**
askEVSRoomData	roomsEVSConsultation	readEVSRooms
This service retrieves statistics, awareness, feedback and pool information to be shown in the *voting system* screen. According to the **type** value we will get values for the folder or for the thread itself. It returns an array with all the data for each folder and thread according to the **type** value.		
Parameters to interact with the *askEVSRoomData* user interface.	$parameters_room_queries = array('session' => $session, 'contributiondata' => array(**'type'** => $type, 'thread' => $room, 'iduser' => $idUser, 'idgroup' => $idGroup, 'subject' => $subject, 'area' => $area, 'role' => $role, 'contribution' => $contribution));	
Type would be	FOLDER, THREAD	

http://$wsHost:$wsPort/axis/services/CSCLRoomInterface		
UI	**Bussines**	**Data**
askEVSRoomAllData	roomsEVSAllConsultation	readEVSAllRooms
This service retrieves statistics, awareness, feedback and pool information to be shown in the *voting system* screen. According to the **type** value we will get values for the folder or for the thread itself. This service is different from the askEVSRoomData because this service retrieves extended data for the tutor. It returns an array with all the data for each folder and thread according to the **type** value.		
Parameters to interact with the	$parameters_read_threads = array(	

askEVSRoomAllData user interface.	'session' => $session, 'roomdata' => array('type' => $type, 'feedback' => $feedback, 'user' => $idUser, 'folder' => $folder_data['idroom'], 'start' => '0', 'end' => $ALL, 'area' => $area, 'group' => $idGroup));
Type would be	FOLDER, THREAD

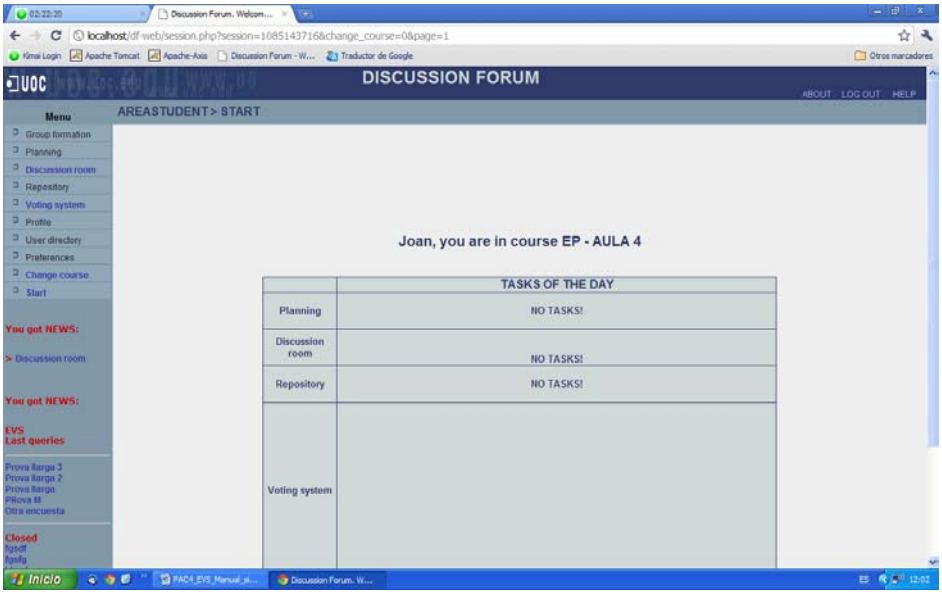
Annex III. Workstation issues

Due to our poor workstation performance we got "*Timed out from localhost*" issues [26]. There we'll show our know issues. After a page reload all these should be solved.

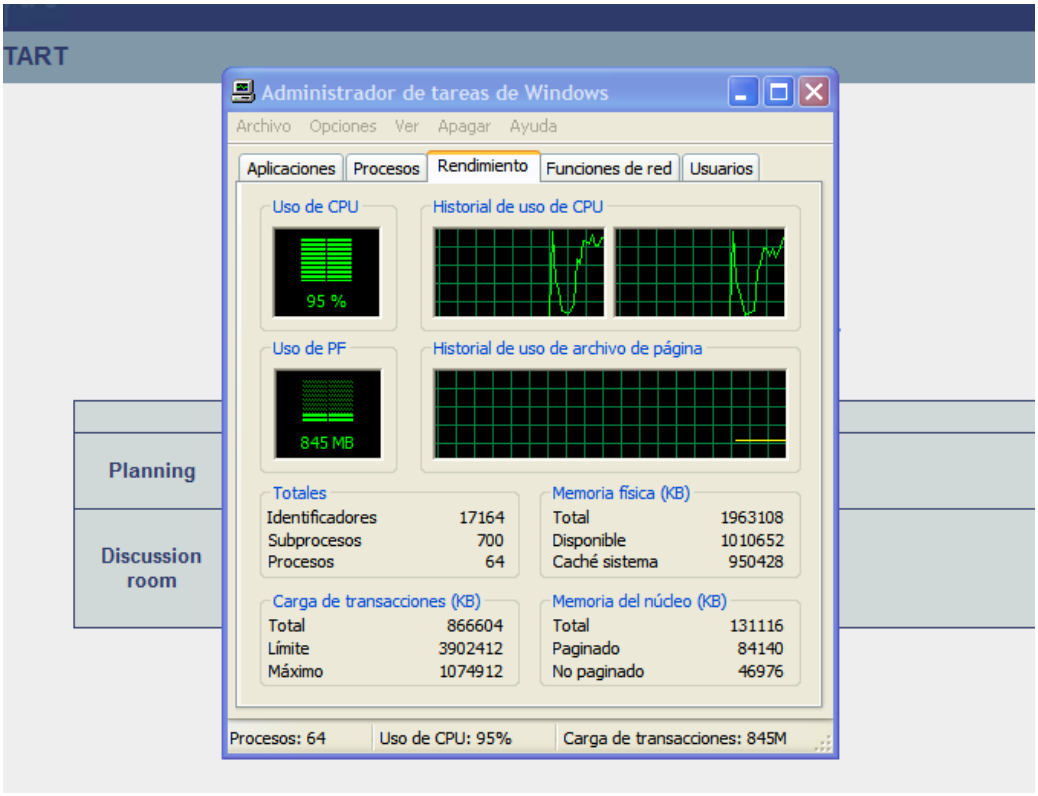


The first time we access the application there is a problem loading the session data and the start page appears empty. This problem is usually solved with a single page reload.

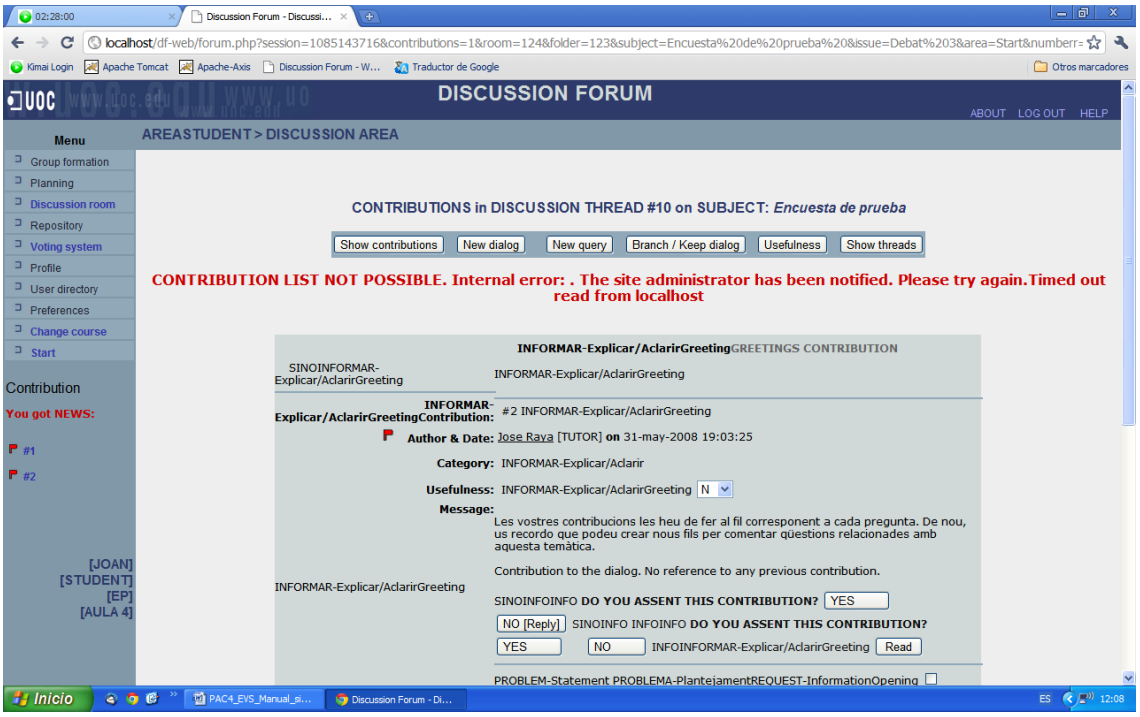
We tried to fix this issue but examining all the code involved in the process we didn't find any code related problem. It would be another *time out* issue.



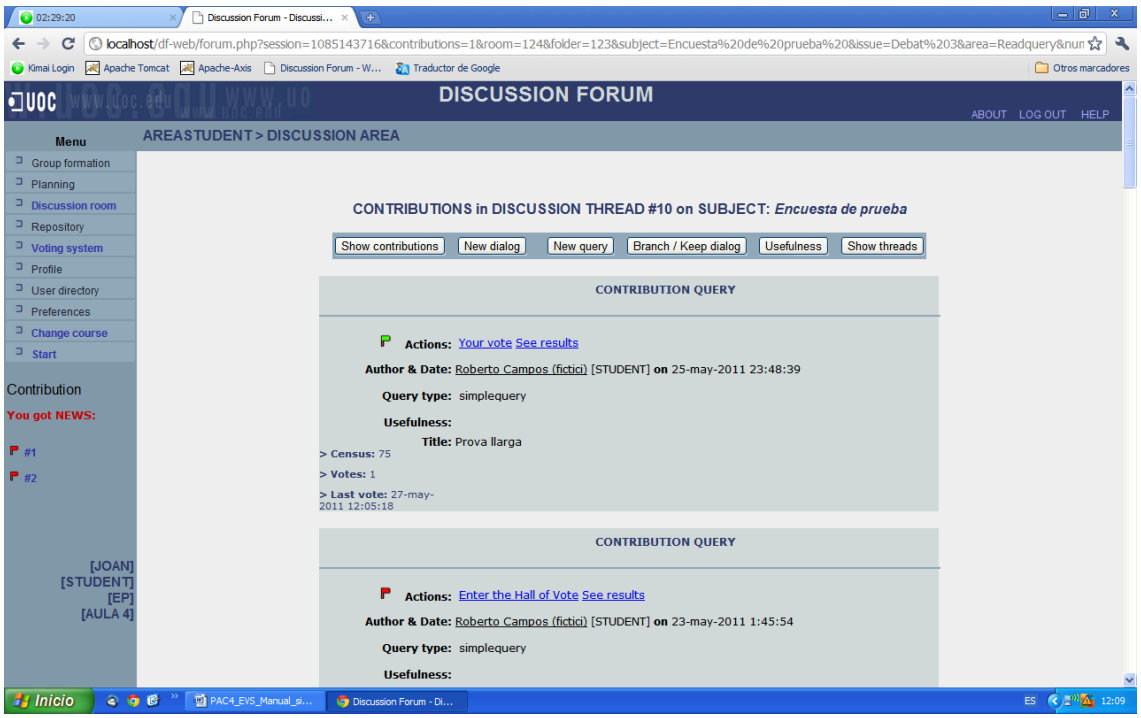
Our workstation goes easily to the 95-100% of CPU consumption executing the EVS application and we experience many issues due to poor system performance.



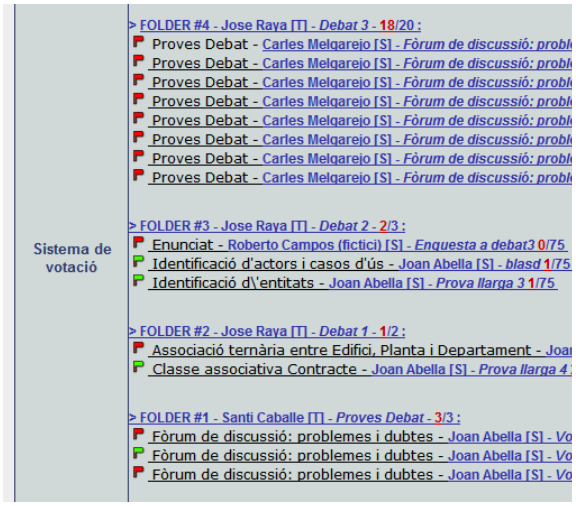
Another common issue is the "Contribution List not possible. Internal error. **Timed out from localhost**". Usually a reload solve the problem.



Below we are showing the same page after a single reload. It's not an application error but a performance issue from our workstation.



The following pictures show another unexpected issue due to our workstation poor performance. There the problem comes from reading unexpected buffered data. Refreshing the page clears the buffer and the data appears correctly. It's again a workstation issue and shouldn't appear in production servers.



Our workstation carries with all the servers under the same machine: *Postgresql, Apache, PHP, Tomcat* and *Axis*. Usually there is an antivirus scanning all the web data transfers, but the antivirus is stopped during the EVS tests. Even stopping other O.S. services the system become more stable but not failsafe.

We apologize for those issues.