Modeling Interactions and Navigation in Web Applications

Natacha Güell[1] natacha@inf.puc-rio.br
Daniel Schwabe[1*] schwabe@inf.puc-rio.br
Patricia Vilain[1, 2] vilain@inf.puc-rio.br
[1]Departamento de Informática, PUC-Rio. Rua M. de São Vicente, 225. Rio de Janeiro, RJ 22453-900
[2]Departamento de Informática e Estatística, UFSC, Campus Universitário Florianópolis, SC 88040-900

Abstract

This work presents a method that bridges the gap between requirements elicitation and conceptual, interaction and navigation design for Web applications.

This method is based on user scenarios, use cases, and a new graphical notation, called User Interaction Diagrams. From these specifications, it is shown how to derive a conceptual model, and then how to derive the navigational structure of a Web application that supports the set of tasks identified in the scenarios.

1. Introduction

Most current hypermedia (and web) design methods such as HDM [6], RMM [9], EORM [11], or even earlier versions of OOHDM [14, 13] provide the designer with models and a corresponding notation to specify the design and implementation of applications. Little guidance is given as to how a designer should interact with all the stakeholders involved, capturing their requirements and eventually developing the actual design.

In this work we present an approach to address this problem, based on a method that employs User Scenarios [1], Use Cases [10] and User Interaction Diagrams (UIDs) [16, 17]. Whereas the first two techniques are well known in the literature, the former is a new primitive specifically designed to capture the externally observable interactions between a user and an interactive application.

The method described in the remainder of the work shows how to gather requirements, how to synthesize a conceptual design and a navigation design of an application from these requirements, as represented in Figure 1.

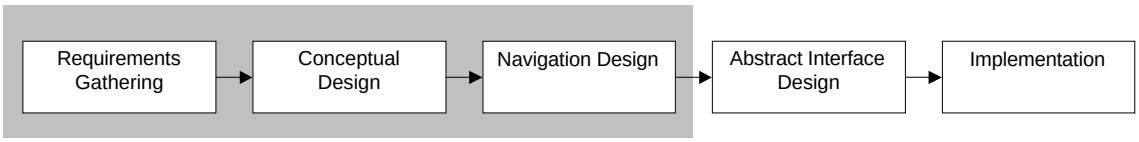


Figure 1 – Phases in the Design and Implementation of Web Applications. The shaded area is addressed by the method discussed in this work.

The presentation in the work will use as an example a website designed for the Intranet of a large corporation, which has offices in several cities geographically spread out over a large territory, focusing specifically on its Training Division. This unit is responsible for offering course and training activities for the entire company, fulfilling demands from all other units, designing new courses, hiring instructors when necessary, providing the infrastructure and actually running classes.

The remainder of the work is organized as follows. Section 2 describes how to first gather and specify requirements through user scenarios, use cases and UIDs, and how to use the collected models to derive a conceptual design for the application. Section 3 shows how to continue the design process by

[*] Daniel Schwabe and Natacha Güell are partially supported by grants from CNPq; Patricia Vilain is partially supported by a grant from CAPES.

synthesizing the navigation design; Section 4 briefly compares the new method with existing work, and draws some conclusions.

2. Requirements Gathering and Conceptual Design

Successful designs star with suitable requirements gathered from users and other stakeholders. Requirements gathering is divided into the following phases (Figure 2): identification of roles and tasks, specification of scenarios, specification of use cases, specification of user interaction diagrams, and validation of use cases and user interaction diagrams. The conceptual design presents a single phase: specification of the conceptual schema.

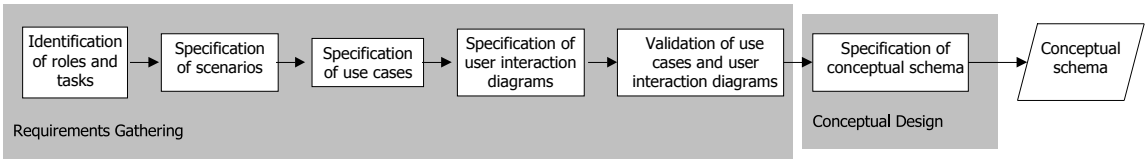


Figure 2 – Phases of the requirements gathering and conceptual design

2.1 Role and Task Identification

In this phase, the designer interacts with the domain to identify the roles[1] users play and tasks the application supports. This interaction is achieved through analysis of documents and interviews of users.

Users can play several roles, in which the user exchanges information with the application. In the example, an initial examination revealed several possible roles in the design of the website for the training division: Student, Manager, Administrative Coordinator, Didactic Coordinator, and Human Resources Coordinator. Later, during the specification of the use cases, we verified that administrative coordinators and didactic coordinators could be unified into single role, Coordinator. We also identified that all other roles had the same tasks, so they were eventually unified into the role Employee.

For each role, we identify the tasks the web application must support, such as (for the role Student): Search information about a course; Search information about an instructor; Get the course material in advance.

2.2 Scenario Specification

In this phase, each user specifies textually or verbally the scenarios that describe his tasks. If a user plays several roles, these roles must be identified. Scenarios are narrative descriptions of how the application may be used [1]. Although the designers also can produce scenarios, we will concentrate on those produced by users.

The tasks identified in the previous phase can guide users in determining relevant scenarios.

In Figure 3, we present some scenarios, specified by students and managers in our case study, describing the search for a course given a subject.

[1]The term “role” used in this work is similar to the term “actor” used in [10].

AL3.4 – Browsing courses given a subject

Courses can be searched by their subjects. I am a software developer. Some subjects are of interest to me, for example, “compilers” and “languages”. For an administrator, subjects such as “operating system” and “tools” are interesting. So, the courses must be classified by the kinds of users.

GA4.4 - Search for a course given a subject

Sometimes when an employee looks for a course, he doesn’t have the name of the course. Therefore, the search by the course name is not sufficient. It is also important to allow the search by subject. Thus, an employee can choose a subject and find all courses related to that subject. In the “Earth Sciences” area, we can find 10 related subjects, “Geophysics”, “Geophysical Computing”, “Inversion”, “Photography”, and so on. Each subject can be associated with two or more courses. There are courses associated to one or more subjects.

GA5.2 – Finding information about a course

To decide if an employee can take a course, we need to get the information about the course. We would like to find the following information in the course page: program, instructor and course material. The course material is important because it can show us the organization and depth of the course.

Figure 3 – Scenarios specified by users in our case study

2.3 Use Case Specification

A use case is a way of using the application [10]. They present the interaction between the user and the application, without considering the internal aspects of the application.

Scenarios describing the same task are grouped into a single use case. The description of a use case has to include the information presented in all related scenarios. The designer must identify which data items shown in the scenarios are relevant. A use case can also be augmented by information from other use cases or by using design patterns [12].

If several types of users perform the same task, the scenarios for these types of user are grouped into a use case, identifying the roles it belongs to.

In Figure 4, we present the use case “Browsing courses given a subject” derived from all scenarios that deal with browsing courses given a subject.

Use Case: Browsing courses given a subject.

Scenarios: ST1.4 / ST2.1 / ST3.2 / MA1.7 / MA3.6 / MA4.4 / MA 5.2 / MA7.1 / MA7.8 / C1.1 / C3.2 / C3.3 / C4.1 / C1.12

Roles: Coordinator, Employee

Description:

1. The user enters the subject, or part of it.
2. The application returns a hierarchical set with the subjects matching the input, and the user selects the subject of interest.
3. The application returns a set with the modules of the selected subject, and the user selects one course in the module.
4. For the selected course, the application shows its name, program, weekly hours, set of subjects, set of teachers, and the set of course materials, if there are such course materials associated with it. For each teacher, the application shows the name, institution, e-mail, photo (optional) and curriculum. For each course material, the application shows the title, summary and table of contents.
5. The user can download the material (download_ftp) or see it (show).
6. If the user wishes, he can enter suggestions about the course.

Figure 4 – Use case “Browsing courses given a subject”

2.4 User Interaction Diagram Specification

For each use case, we define a user interaction diagram (UID) [16]. UIDs graphically represent the interaction between the user and the application, described textually in the use cases. This diagram only describes the exchange of information between the user and the application, without considering specific user interface aspects. UML (*Unified Modeling Language*) [2] does not present similar diagram. UML diagrams representing the interaction, such as the sequence diagrams, collaboration diagrams, statechart diagrams, and activity diagrams, consider the exchange of messages between the objects of the system. So, they are more appropriately used in the design phase since in the requirement gathering we do not have knowledge about the objects yet. Figure 5 shows the UID that has been defined to represent the interaction in the task "Browsing a course given a subject".

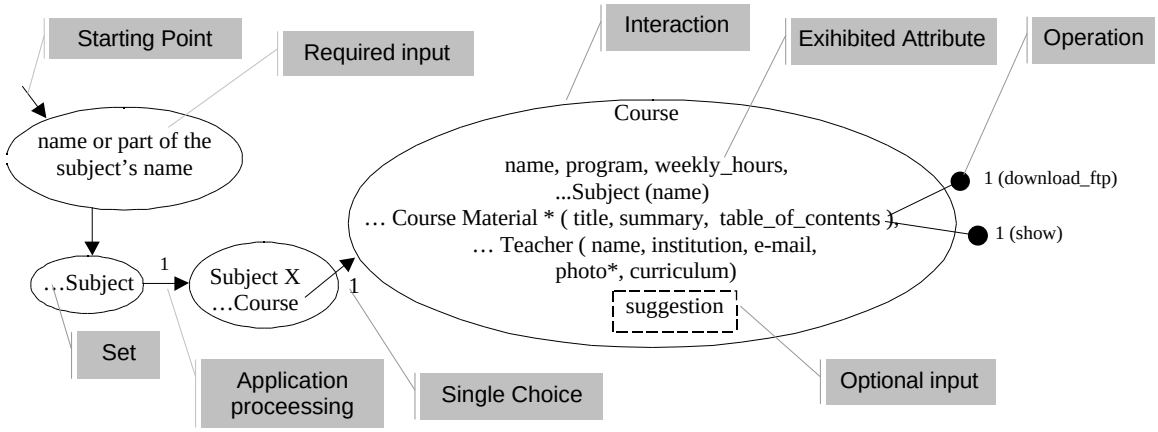


Figure 5 – User interaction diagram, use case "Browsing courses given a subject"

The ellipses represent the interaction between the user and the application, showing the information exchanged between them. Arrows, denoting that processing in the application occurs before the next information is presented, connect the ellipses. An arrow without a source represents the first interaction. A line represents operations that do not require the interaction exchange with a black bullet in one end.

In this example, the user begins the interaction by entering a subject of interest, or part of it. This input is required and is represented by a rectangle with a continuous border. Next, the application presents the set of subjects related to the input, within which the user selects one subject, represented by the number 1 attached to the arrow. Next the application shows the set of courses related to the chosen subject, and the user selects one course. The application shows all the information related to this course, and it is also possible to get the course material via ftp or visualize it (download_ftp() and show()). If the user wishes to, he can also enter suggestions about the course.

2.5 Use Case Validation

In this phase, the designer interacts with each user to validate the use cases and UIDs showing them to the users to ascertain if they agree with them. The user validates only those use cases and diagrams related with the roles he plays. Before beginning the validation with any user, the use cases have been modified according to result of the validation with the previous user, and a table is used to keep track of the successive alterations

2.6 Conceptual Schema Specification

The conceptual design is composed of one single phase, the specification of the conceptual schema. The conceptual schema is obtained from UIDs according to some rules. Some of these rules are just an adaptation of standard schema normalization techniques [4] so that these rules can be applied to UIDs. The most important rules are presented below (other details can be found in [16]).

1. For each UID, define a class for each element, set element and specific element. For each defined class, assume the existence of an identifier attribute OID.
2. For each data item of the set elements that appears in each UID, or data item input by the user, define an attribute according to the following:
   - Verify that the data item is functionally dependent (see [4]) on the attribute OID in each class, i.e., that OID → data items. Verify that the data item is not transitively dependent (see [4]) on the OID. If these conditions are satisfied, then the data item must become an attribute of the class.
   - Verify that the data item is functionally dependent, but not transitively dependent, on the attribute OID of two or more different classes and that the data item is not transitively dependent on the OIDs. If these conditions are satisfied, then the data item must become an attribute of a relationship between these classes.
3. For each data item entered by the user, represented by an single rectangle, define an attribute according to the following:
   - Verify that the data item is functionally dependent (see [4]) on the attribute OID in each class, i.e., that OID → data items. Verify that the data item is not transitively dependent (see [4]) on the OID. If these conditions are satisfied, then the data item must become an attribute of the class.
   - Verify that the data item is functionally dependent, but not transitively dependent, on the attribute OID of two or more different classes and that the data item is not transitively dependent on the OIDs. If these conditions are satisfied, then the data item must become an attribute of a relationship between these classes.
4. For each attribute that appears in a set different from its class, tentatively define a relationship between its class and the class of the set elements. If the attribute class is not related to the class representing the set, then verify that the attribute class is related to the class of another attribute present in the set. Verify that this resulting relationship is semantically correct (i.e. if it makes sense in the domain being modeled).
5. For each interaction flow (represented by an arrow), if there are different classes in the source interaction and the target interaction, define a relationship between these classes. Verify if this relationship is semantically correct (i.e. if it makes sense in the domain being modeled).
6. For each operation that appears in the UIDs, verify that it is an operation of the corresponding class.

At the end of the process, do the necessary adjustments in the resulting class diagram, for instance, identify generalizations and missed relationship cardinalities.

Figure 6 shows the part of the class diagram that results from these steps, for the example:

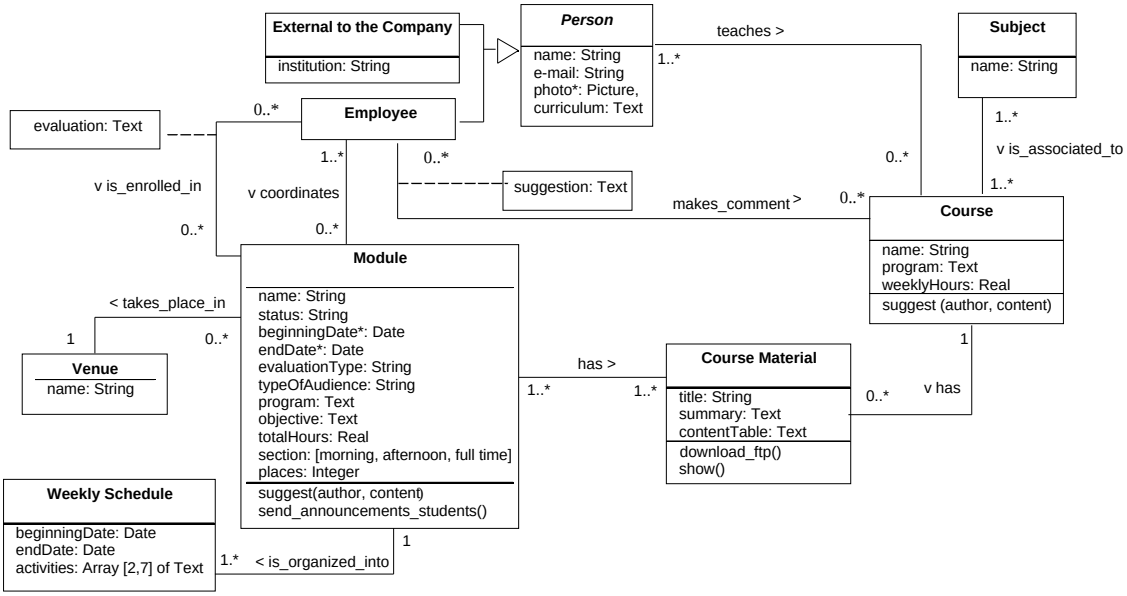


Figure 6 – Part of the conceptual schema of the application

In our study case, the training division offers courses to train company employees. Courses are organized into modules. Each module takes place in a certain period, in some venue. The module activities are organized into a weekly schedule. The employees of the company can enroll in several modules and some employees will be module coordinators. Each course is associated with one or more subjects, and it may have course materials. Course teachers may be company employees or people external to the company.

3. Synthesis of the Navigational Model

3.1 Overview

The previous phases have produced, for each task, several scenarios, a use case and a UID. We describe next a method (first proposed in [7]) to derive the navigational topology of an application that supports each task, allowing the designer to reuse solutions or apply design patterns, according to his own experience. In suite, the navigational solutions of the different tasks are unified into a general solution for the application. The goal is to obtain the OOHDM Navigation Model of the application.

The four fundamental phases of the navigational design are shown in Figure 7 below.

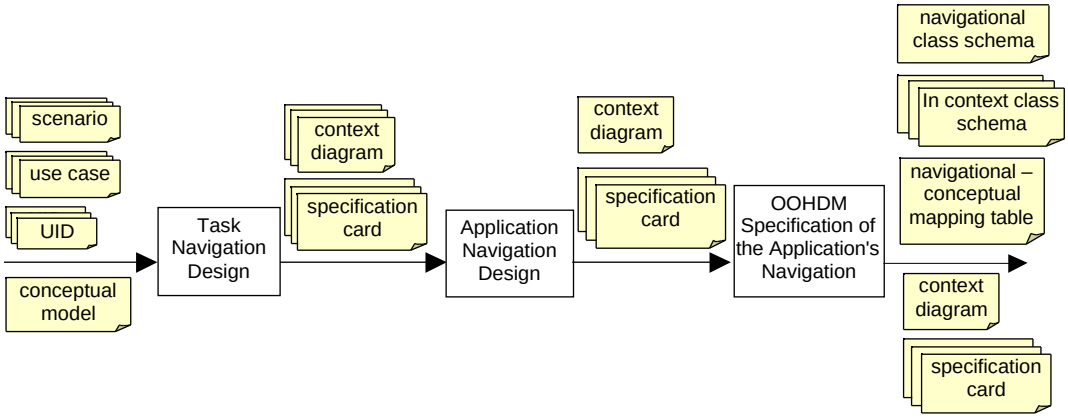


Figure 7 – Main phases of the navigational design

In the first stage, the navigation of each task (represented by a use case) is designed, represented in a context diagram and the corresponding specification cards. The necessary navigational information for this specification can be obtained either by analysis of the scenarios or through the reuse of (previous) project solutions known to the designer.

At this point the designer already possesses a navigational vision of all the individual parts of the system. The navigation design of the whole application is obtained by unifying the solutions for each task. Once the union process is concluded the navigation will be completely specified.

Each of these (sub) tasks are examined in more detailed in the next sub-sections.

3.2. Task Navigation Design

For each task, the most appropriate navigation sequence that supports the user is determined. The designer can base himself on the UID, since it was validated with the users and represents, therefore, an acceptable form of carrying out the task. In spite of this, the designer should not adopt the UID as its only guide, but also consult the scenario descriptions, since these contain the knowledge that users have about the task at hand. The designer can also reuse solutions either based on his own experience or by applying design patterns [12].

There are tasks where the user needs to work with sets of objects, which can be explored in different ways according to the user's objective. OOHDM has a design primitive for such sets, called *navigational context.* The navigational structure of an application is characterized as a group of contexts in which the objects will be accessed. This is specified in the context diagram, as exemplified in Figure 8.

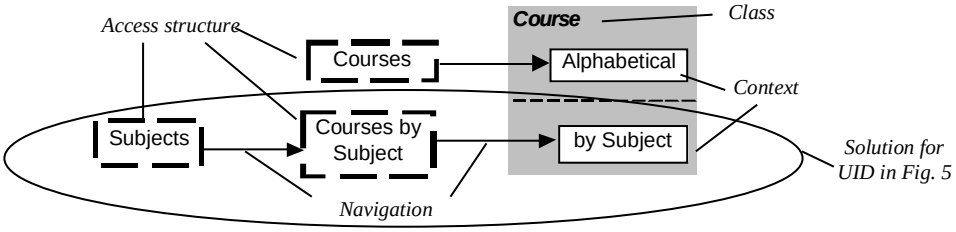


Figure 8 – Example of context diagram

The context diagram in Figure 8 represents three access structures (“Courses”, “Subjects” and “Courses by Subjects”) and two contexts for the class Course (“Alphabetical” and “by Subject”). The sub-diagram composed of “Subjects”, “Courses by Subject” and the context “Course by Subject” represents a possible navigational solution for the task described in the UID of Figure 5.

The main steps in the task navigation design the are represented in the Figure 9:

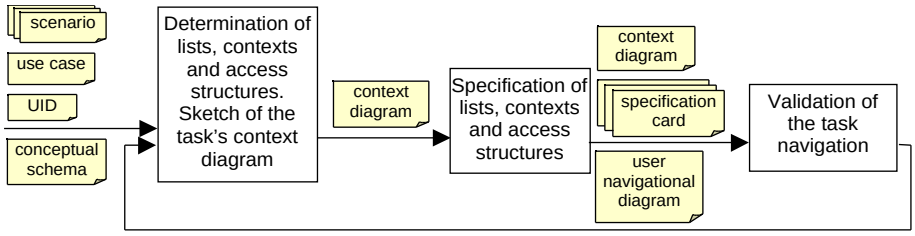


Figure 9 – Steps to design the navigation of a task

3.2.1 Determination of lists, contexts and access structures; sketch of individual task context diagram

Each set represented in the UID is analyzed to determine the type of primitive it will become: an access structure, a context or a list attribute (an ordered set of elements). The solution will be represented in a context diagram.

The following rules can be applied during the analysis of each UID:

Rule 1: Mapping of the interaction that presents an object, into a context of the object class

Every object is navigated within a context. Therefore, any interaction that presents an object should be mapped into the context where the object will be navigated. In the diagram in Figure 10 the third interaction presents an object of the class Module, and the following interaction presents an object of the class Course. Each must then be mapped into contexts: a context of class Module and another of class Course. (Professor, revise o ingles de esta ultima oracion)

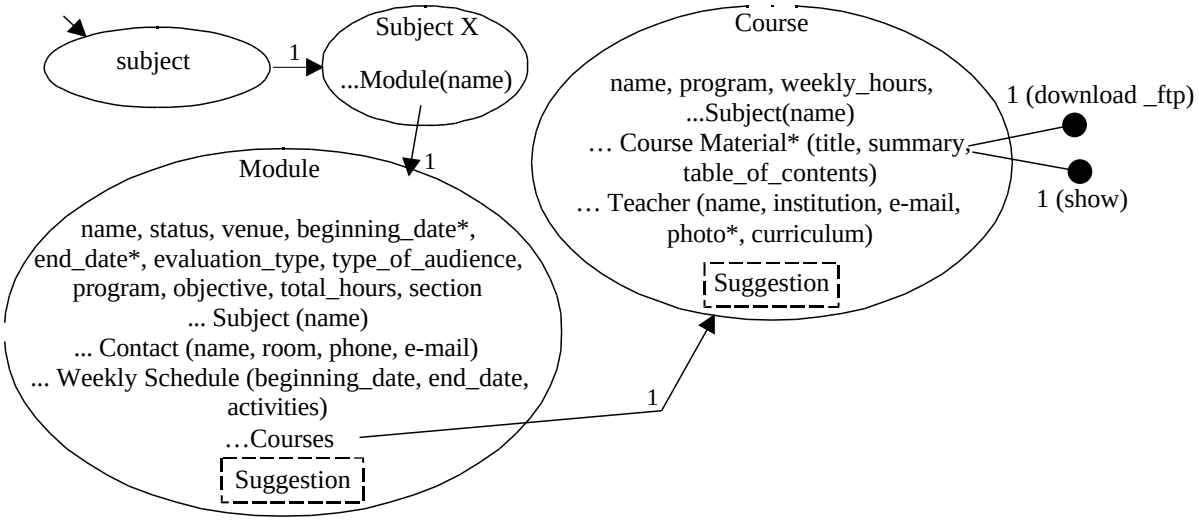


Figure 10 – User interaction diagram of the use case “Browsing modules given a subject”

The next step is to determine the elements of each context. The elements of a context generally have some common characteristic: they belong to the same class, they have attributes with the same value, or they are linked to a given object through the same relationship. Otherwise, the set is arbitrary, so each element must be enumerated.

To determine the elements of a context we analyze the source of the interaction in the UID. The following guidelines can be used:

- If the source of the interaction is within a set of objects that have a common characteristic, the natural elements of the target context are these same objects. In the example, the interaction that presents the object of class Module is accessed from the interaction that presents the set of modules of a certain subject. Therefore, the objects of this context will be the modules with the same subject. Similarly, the objects of the Course context will be the courses in a certain module.
- If the source of the interaction is another object, the elements of the context are those that are linked to the source object through a certain relationship.
- If it is an initial interaction, it is not yet possible to specify the objects of the context. We adopt the convention of labeling these contexts with “<Class name> by ?”.

Figure 11 represents the mapping of the interactions of the diagrams in Figure 5Figure 10 that were analyzed according to rule 1:

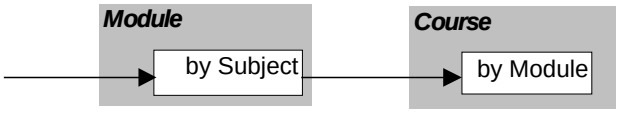


Figure 11 – Part of the context diagram of the use case “Browsing modules given a subject”

Rule 2: Mapping of a set that is part of the information of an object, into an access structure, or into a context, or into a list attribute

In the diagram of interaction of Figure 10, the information about a module includes the following sets: Subject (subjects of the courses in the module), Contact (employees of the training center that coordinate the module), Weekly Schedule (weekly activities of the module) and Courses (courses that will be taught in the module). The information about a course includes the sets: Subject (subjects associated with the course), Course Materials and Teacher (people who teach the course).

To decide the mapping of each set we analyze the importance of its information with respect to the execution of the tasks that will use the object. To determine this degree of importance, it is necessary to analyze the descriptions of the use case and its scenarios, in order to better understand the task and users' goals.

· Very important information should appear integrally as lists that are attributes of the object:

According to the scenarios described by the students in the example, they would not choose to attend a course without previously knowing its subjects, nor who are its teachers. Thus, all the information about the subjects and about its teachers will be shown together with the information about the course.

· Important information may appear as access structures tied to the object. In this case the designer should decide the attributes that will be presented in the structure:

The “Course Material” set was considered important but not essential information, for the task of choosing the course. Some users described that the course material is important because it shows the organization and depth of the course. On the other hand, the students will access the course material when they are about to attend it, i.e., when they are already registered for the module. Therefore, the course material will be accessed through an access structure as part of the course description.

· Complementary information can appear as an anchor, that points to an access structure or to an element in context:

The module’s activity schedule does not constitute decisive information for the choice of a module, so it will be accessed by means of an anchor that points to the object that details it.

Figure 12 represents the mapping of ’s diagram after the module and course sets have been analyzed. In the schema, for each context, a card is included that describes its object vision and its access permissions. It may happen that the same object can be accessed in different contexts and it may be necessary that in each context the object presents the information in different ways, as well as that it allows different navigations. This will be solved through *In Context classes*.

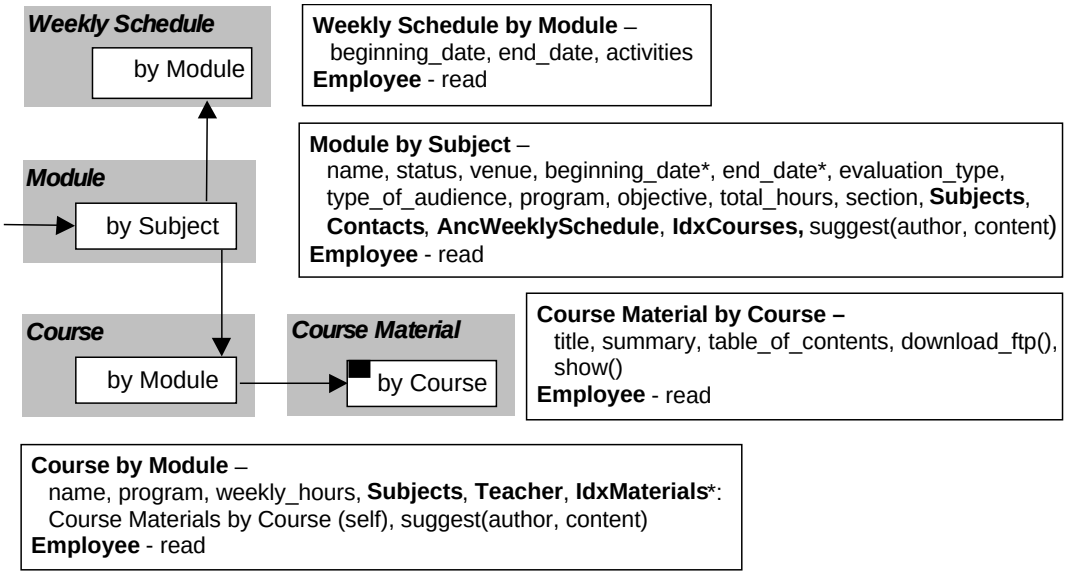


Figure 12 – Context diagram, use case "Browsing modules given a subject"

Rule 3 Mapping of a data entry interaction followed by an interaction that presents a set of objects, into an access structure of the set's objects

The choice of the access structure depends on the possibility of the data entered by the user being generated by the application, and whether this data belong to the objects of the target set or to the other objects linked to them.

The diagram in Figure 13 shows the search of modules given a keyword. The user can enter a keyword, also selecting the attributes where it will be searched; the application returns the modules that match the search.

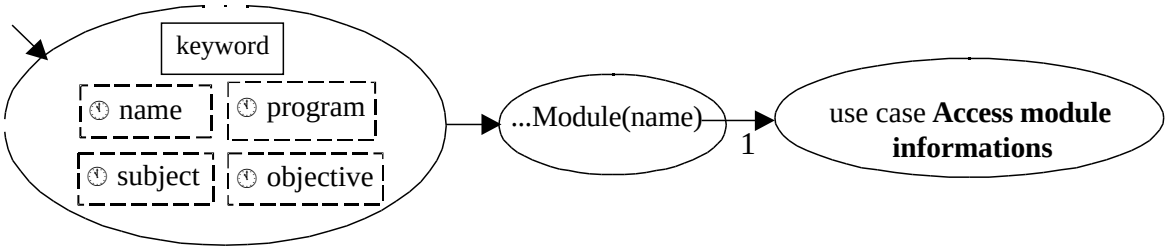


Figure 13 – User interaction diagram, use case "Searching modules by keyword"

In this case, the system is not able to generate the results of the user's entry beforehand. In such cases, the data entry interaction and the resulting set are mapped into an access structure whose elements will be computed from the user's input (*dynamic structure*). Figure 14 represents the mapping of Figure 13's UID:

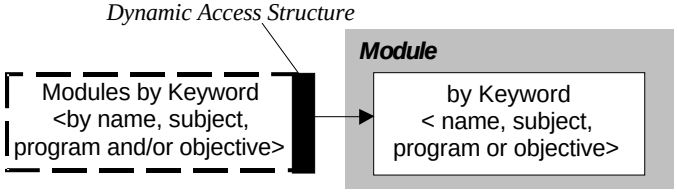


Figure 14 – Context diagram, use case "Searching modules by keyword"

In the case of Figure 10, the user must enter a subject, and the system returns the courses associated with that subject. Since in this case the application knows beforehand all possible valid inputs by the user, it could generate all the subjects stored in the knowledge base for the user to choose one, without

having to actually enter it. Therefore, both interactions are mapped into an access structure. The users explained that they wanted to enter the subject because the number of subjects is large and to select one could be cumbersome. Such feedback will be taken into account during the design of the interface for the access structure.

The type of the access structure, simple or hierarchical, will depend on the data entry and on the objects of the destination set. A hierarchical access structure is a sequence of simple structures where the selection of each structure is the input parameter for the succeeding one [14].

In the diagram of Figure 10, the user enters a subject and the target set is formed of courses linked to this subject. Therefore, both interactions are mapped into a hierarchical access structure. In the first level, the subjects are presented ordered alphabetically and in the second level the courses for each subject selected in the first level are presented. Figure 15 represents the mapping of the initial interactions of the diagram represented in Figure 10:

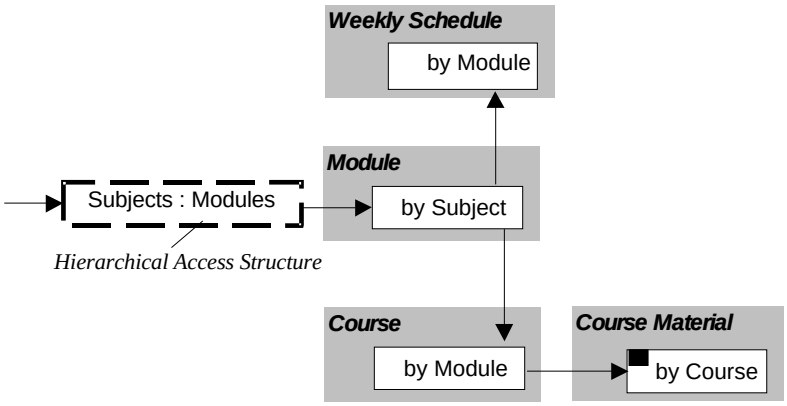


Figure 15 – Context diagram mapped from the “Browsing modules given a subject” UID.(Figure 10)

3.2.2 Specification of lists, contexts and access structures

The construction of the context diagram of the tasks is completed with the specification of the lists, contexts and access structures in their respective cards, presented next. In these cards are recorded data that are not representing in the context diagram: the characteristics of the elements of the set, the ordering, the kind of the context internal navigation, etc.

Is defined a specification card for: each list that is an object attribute, each context represented in the context diagram and each type of access structure used in the context diagram, regardless of the structure being represented graphically or being described as an attribute of a class in some context vision.

The example is based on the context diagram of Figure 15 and the vision cards in Figure 12:

Figure 16 illustrates the specification card for the list “Contacts”, that is an attribute of class Module, in the context "Module by Subject":

List : Module by Subject . Contacts
Attributes e Operations : name, room, phone, e-mail
Conceptual rule: SELECT e.name, e.room, e.phone, e.e-mail FROM e: Employee, m: Module WHERE e ***coordinates*** m ORDER BY e.name, Ascending

Figure 16 – Example of a specification card for lists

Figure 17 illustrates the specification card for the context “Module by Subject”:

Context: Module by Subject
Type: static
Parameters: s: Subject
Elements: m: Module WHERE m ***is_associated_to*** s ***Conceptual rule***: m: Module WHERE (m ***has*** c) AND (c ***is_associated_to*** s)
In context Classes:
Ordering: by name, ascending
Internal navigation: by index (**Modules by Subject**)
Operations:
Users: Employee **Permissions**: read
Comments:

Figure 17 – Example of a specification card for navigational contexts

The selection rule of the context's elements is derived from the context diagram. When this rule does not map directly from the conceptual model, the conceptual mapping rule should also be recorded in the card, in order to facilitate the mapping of the navigational classes from the conceptual model, discussed later on.

An example of that is the selection rule of the context “Module by Subject”. From the contexts diagram (Figure 15) the rule will be:

m: Module FROM s: Subject WHERE m ***is_associated_to*** a.

According to the conceptual model there is no relationship between the classes Module and Subject. The rule derived from the conceptual model is:

m: Module FROM s: Subject, c: Course WHERE (m has c) AND (c ***is_ associated_to*** s).

Figure 18 illustrates the specification card of the class “Course Materials for Course”, that is the type of the access structure of the attribute IdxMaterials, that appears in the class Course, in the context “Course by Module”.

Access structure : Course Materials by Course	
Type: simple	
Parameters: c: Course	
Elements: cm: Course Material WHERE c ***has c***m	
Attributes title..... summary ……..... ……....	**Target** Ctx Course Material by Course (c) download_ftp() show()
Ordering: by title, Ascending	
Users: Employee **Permission**: read	
Comments:	
Trace backward:	**Trace forward:**:

Figure 18 – Example of a specification card for access structure

3.2.3 Validation of each task navigation

Once the task navigations have been designed, the designer can validate them with the users, using context diagrams as a communication tool between the designer and the users. An intermediate notation may also be used: the user navigation diagram. This is an extension of the UID that allows representing navigational information. The notation of this diagram can be found in [16].

During validation the designer will maintain a copy of each solution to record each interview, drawing changes or writing comments. Later on, based on these records, and on his own experience, the designer will opt for a solution, trying to reconcile the changes proposed by the various users. If the validation results in significant changes, the context diagram is sketched again and the process is repeated. Otherwise, the designer can continue on to the next step of the process.

If the users present conflicting opinions about the solutions presented, this may actually indicate that different applications may be needed to accommodate this group of users.

3.3. Application Navigation Design

The application's context diagram will be synthesized through the successive union of the context diagrams of the individual tasks. For each union step, a new context diagram is built, defining its contexts, its access structures and the object visions for each context.

We start by unifying the solutions of each user group, beginning with the group that performs the main tasks. Tasks in which the same navigational object is accessed are unified in sequence. Instead of maintaining the solution for each group separately until the end, we prefer to unify them as soon as each solution is defined.

The same navigational class may appear in contexts originating in different partial diagrams, so we analyze if some of these contexts may be replaced by a single context. We analyze the tasks that each context supports, as well as the object visions and users permissions in these contexts. We will replace them by a single context only if the resulting context supports all such tasks, and if the object visions and the user permissions do not conflict.

We also analyze the possibility of unifying the access paths leading to the original contexts and if it is possible to generalize the original navigations of these contexts within the resulting diagram. Since they are now part of the same diagram, it is necessary to explore the possible navigations between them.

As an example, we will unify the previously discussed tasks, starting with Employee. We will unify the solutions for the use cases "Browsing modules given a subject" (Figure 15) and "Searching modules by keyword"(Figure 14), since class Module is accessed in both.

Contexts, and their superset, when present, may be both replaced by a single context. The context "Module by Keyword" (modules whose name, program, subject or objective contains a certain keyword) includes the context "Module by Subject", therefore "Module by Subject" could be replaced by "Module by Keyword".

After analyzing the tasks, we opt to maintain both contexts (Figure 19). Search based on a keyword is useful for users that know the module's characteristics, while navigation guided by access structures leading to "Module by Subject" is useful for beginners.

We must then analyze how will the joint navigation of these contexts be in the resulting diagram. According to the conceptual model, every module can have activities and courses associated to it (contexts "Weekly Schedule by Module" and "Course by Module"). Evidently, these navigations can be generalized for all contexts of class Module, including "Module by Keyword" as shown in Figure 19. Notice that the union with use case "Browsing courses given a subject" (Figure 8) was also represented, so the diagram in Figure 19 is the union of the all employee's tasks.

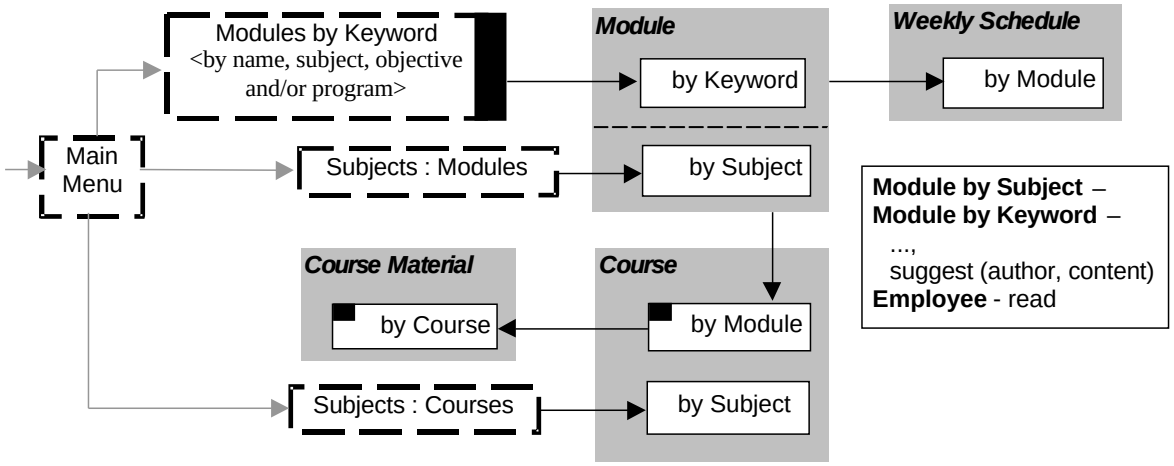


Figure 19– Resulting context diagram, including the employee's tasks.**Error! Reference source not found.**

Next, we must join the sole coordinator's task (Figure 20): "Browsing coordinator's module". None of the contexts represented in this diagram is a subset of some context in the diagram in Figure 19 so once again both diagrams are united. Since there is a new context for class Module ("Module by Coordinator"), we must once again analyze their joint navigation.

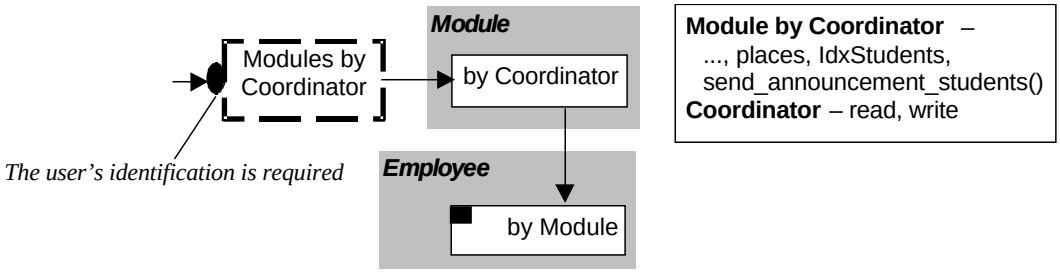
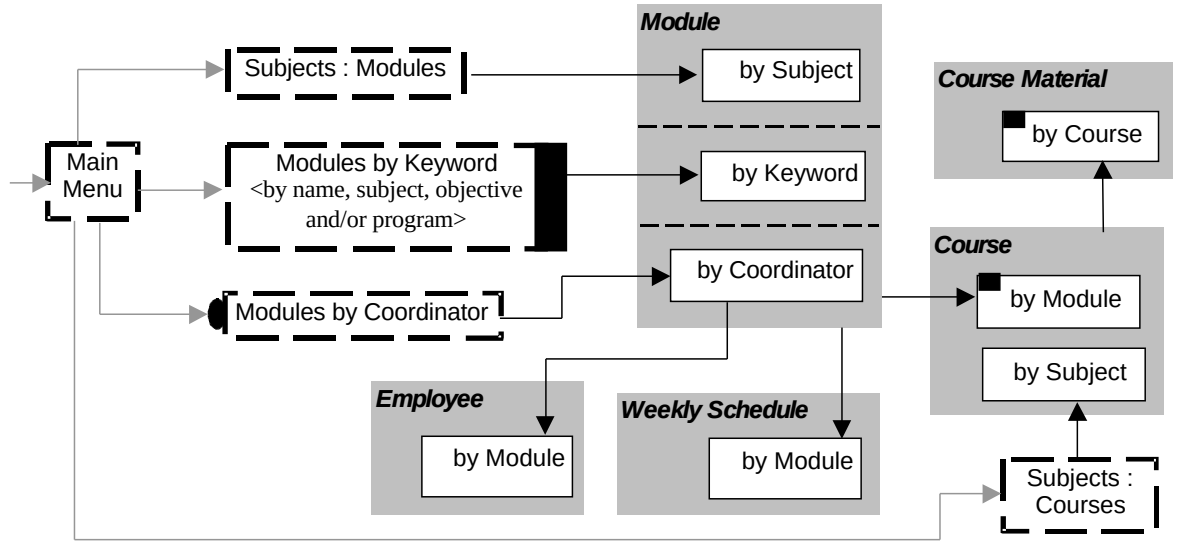


Figure 20 – Context diagram of "Browsing coordinator's module" task

The navigation from "Module by Coordinator" to "Employee by Module" is based on the fact that every module has several enrolled students who are employees. This navigation cannot be generalized due to the class visions and to the user permissions for these contexts. In fact, within "Employee by Module", evaluation data about students (employees) can only be accessed by the coordinator. In addition, in "Module by Coordinator" it is possible to send official announcements to the students.

On the other hand, in "Module by Subject" and "Module by Keyword", students can send suggestions about the course to the coordinator. Therefore, due to the differences of the data and navigations of context "Module by Coordinator" with respect to the others contexts of class Module, the navigation between these contexts is not allowed (represented by dashed line in Figure 21).

In the Main Menu there are options for the different groups of users. Its interface must indicate to each user type, employees or coordinators, what are its options. The context diagram in Figure 21 represents it through the access restriction to the access structure "Modules by Coordinator".



Module by Subject
Module by Keyword–
name, status, venue, beginning_date*, end_date*, evaluation_type, type_of_audience, program, objective, total_hours, section, Subjects, Contacts, AncSchedule: Anchor(Weekly Schedule by Module (self)), IdxCourses: Courses by Module (self), **suggest**(author, content)
Employee - read

Module by Coordinator -
name, status, venue, beginning_date*, end_date*, evaluation_type, type_of_audience, program, objective, total_hours, section, **places,** Subjects, Contacts, AncSchedule: Anchor(Weekly Schedule by Module (self)), IdxCourses: Courses by Module (self), **IdxStudents**: Employees by Module (self), **send_announcements_students**()
Coordinator- read, write

Figure 21 – Final resulting context diagram

The context diagram of Figure 21 supports all tasks that were previously discussed: “Browsing courses given a subject”, “Browsing modules given a subject”, “Searching modules by keyword” and “Browsing coordinator’s module”.

3.4. Navigational Specification of the Application

The OOHDM navigational design specification is a set of models, enumerated here in the same order that the project documentation should be presented: one conceptual-navigational mapping table, one navigational classes schema, several In Context classes schemas, one context diagram, and several specification cards. The context diagram and the cards were obtained in the union process.

3.4.1 Navigational Classes Schema

Navigational classes describe the information content through which the users will navigate. These classes constitute visions of conceptual classes: a navigational class can present some information of a conceptual class or a union of the contents of several conceptual classes. Navigational classes are called nodes; navigational relationships are called links; and node attributes that activate navigations are called anchors.

Every class of the context diagram is a node. The navigations among the context diagram’s classes could be links. We should analyze the selection rule of the target context, especially when it involves navigations between contexts of the same class. A detailed analysis is outside the scope of this work.

In order to exemplify we will derive the navigational classes schema from the Figure 21’s context diagram. In that diagram there are five classes (Module, Course, Weekly Schedule, Course Material and Employee), and several navigations among classes (from Module to "Course by Module",

"Weekly Schedule by Module" and "Employee by Module", and from Course to "Course Material by Course"). The selection rule of the context "Course by Module" (***Parameter***: m: Module | ***Elements***: c: Course WHERE m ***has*** c), indicates that the context is integrated by the courses of the module that is passed as a parameter, therefore there should be a link from Module to Course.

The Figure 22 shows all the nodes and links in the resulting navigational classes schema.

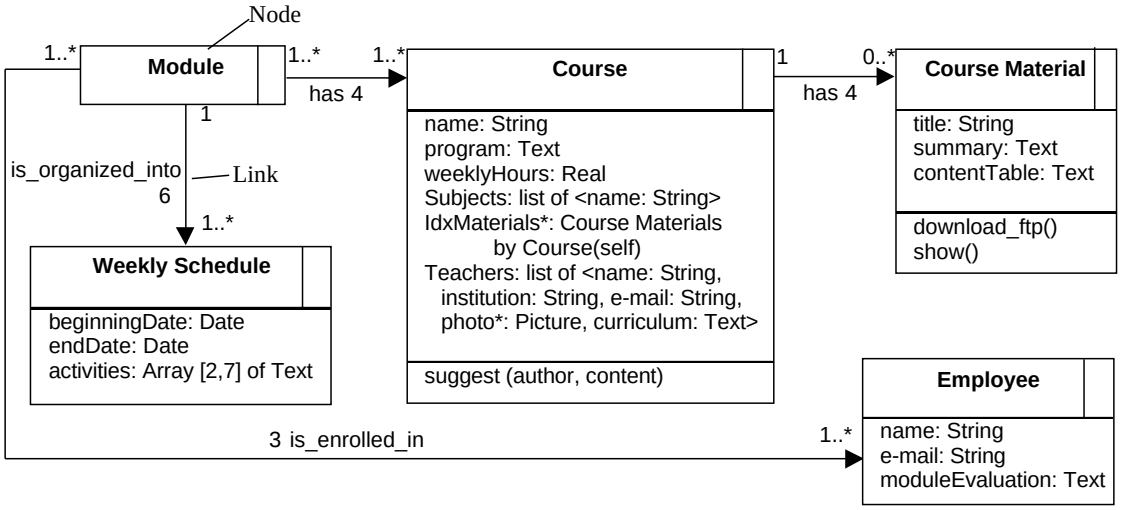


Figure 22 – Part of the navigational classes schema of the application

3.4.2 In Context class schemas

In this step we will build the InContext class schema for each node. We derive the In Context classes from the navigations and the object vision cards of the context diagrams, which record the peculiarities of the object in each context. The object peculiarities must be defined as “decorators” that enrich the object when it is visited within the different contexts, called *InContext classes* in OOHDM. The complete navigation object that the user navigates is the combination of the “pure” navigational object with its InContext Classes, according to each particular context within which the navigation takes place.

In the example, the only class that has several visions is Module: one for the students (contexts “by Subject” and “by Keyword”) and another for the coordinators (context “by Coordinator”). Therefore only the Module ‘s In Context classes schema is defined (Figure 23).

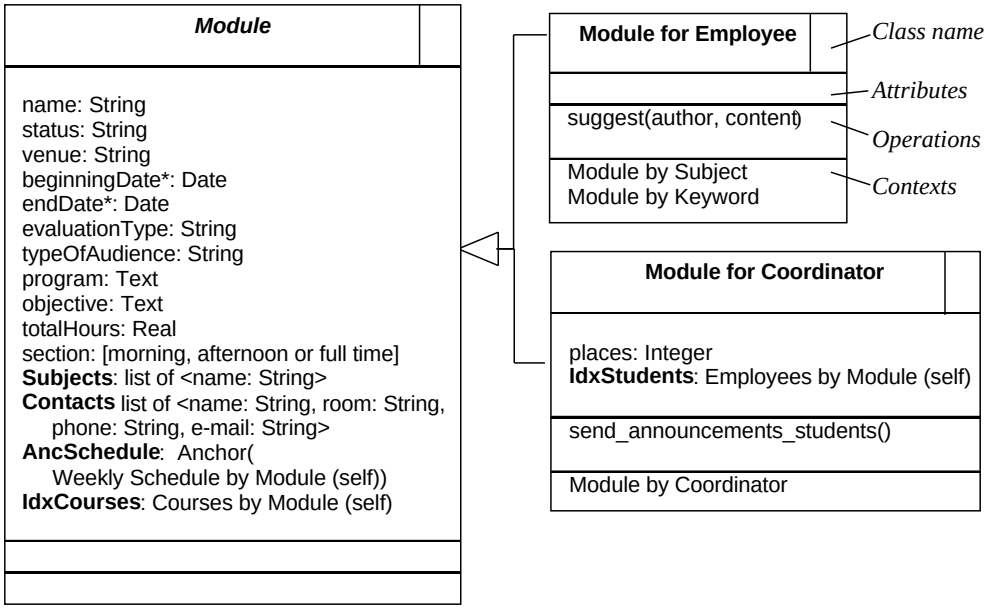


Figure 23– Class Module In Context classes diagram

Nodes that do not have In Context classes are specified only in the navigational class schema. All nodes are represented in the navigational class schema, but only the class name is given, for nodes that have In Context classes (in which the full specification is found).

3.4.2 Table of conceptual - navigational mapping

The next step is to specify the conceptual mapping of each node and link.

The base conceptual class of each node is determined. Next, by comparison, the attributes and operations of the node that don't appear in the base class are identified and a mapping rule for each one is expressed. The base conceptual class of a navigational class is the one derived from the same interaction in the UID during the synthesis of the conceptual class schema.

The attributes and operations of a node may not appear in its base conceptual class for two reasons: they are defined during navigation design, such as anchors, lists and access structures; or they belong to other conceptual classes related to the base class.

As an example, the comparison of the node Module with its base conceptual class Module (Figure 6) reveals some additional navigational attributes: “Subjects”, “Contacts”, “IdxCourse”, “AncSchedule” and “IdxStudents”, created in the navigation design, and “venue” that is functionally dependent of the class Venue. The table in Figure 24 shows (part of) the mapping from the conceptual class Module to the node Module.

Objects	Mapping Rules
Navigational: Module.venue ***Conceptual***: -	SELECT v.name FROM m: Module, v: Venue WHERE m ***takes_place_in*** v
Navigational: Module.Subjects.name ***Conceptual***: -	SELECT s.name FROM s: Subject, m: Module, c: Course WHERE (m ***has c***) AND (c ***is_associated_to*** s)
Navigational: Module.Contacts.name, Module.Contacts.room, Module.Contacts.phone, Module.Contacts.e-mail ***Conceptual***: -	SELECT e.name, e.room, e.phone, e.e-mail FROM e: Employee, m: Module WHERE e ***is_coordinator_of*** m ORDER BY e.name

Figure 24 – Part of the conceptual - navigational mapping table

Most links are directly derived (i.e., one-to-one) from conceptual relations. In some cases, however, certain links correspond to the composition of conceptual relations, and their derivation rules must also be specified in the conceptual-navigational mapping table. In the example in Figure 22**Error! Reference source not found.**, all links are directly derived.

3.5. Final specification of the Application's Navigation

The elements, parameters and selection rules of the contexts and access structures specification cards, should be based on the navigational classes (nodes and links). The conceptual rules do not appear in these cards anymore, since they were created only to aid the conceptual mapping.

In this step, the specification cards are revised with the goal of eliminating any difference with the navigational model. For example, the context “Module by Subject” was specified initially with the node “Subject”, but at the end of the design; “Subject” wasn’t defined as a navigational class. In fact, it was turned into an attribute of node Module, specified in the conceptual-navigational mapping table. The selection rule in the card therefore must be updated to eliminate de differences between the conceptual model and the navigational model (Figure 25).

Context: Module by Subject
Parameter: s: Subject
Elements: m: Module WHERE m ***is_associated_to*** s

specification revision

Context: Modules by Subject
Parameter: subject_name: String
Elements: m: Module WHERE m.Subjects.name ***contains*** subject_name

Figure 25 – Specification revision of the Figure 17’s context card

4. Conclusions

We have presented a method that allows requirements gathering, conceptual design and navigation design of Web applications, based on scenarios, use case and UIDs. This method combines, in a unique way, traditional concepts, such as scenarios and use cases, and new concepts, such as the UIDs, in the navigation of hypermedia application development.

Several approaches also use scenarios and use cases for gathering requirements, but in a different way. In [15], scenarios are used to validate the requirements and are automatically generated from use cases elicited from the users. The scenarios used in [5], different from our scenarios, describe interface and navigation aspects, since they are used in the redesign of an existing web site.

In [8] the requirements gathering is based on use cases derived from user stories. A user story can be seen as a simpler scenario, where an agent performs a physical action on an object. However, in both users stories and use cases, the interactions between the users and the system are specified only as a textual description.

In the Unified Process [10], the requirements are also defined by use cases, but scenarios are not used to specify them. If necessary, user interface prototypes can be used to understand and specify the requirements, but there are no diagrams that focus specifically on the interaction between the user and application.

The method WSDM [3], for designing read-only web sites, proposes a similar approach than the one presented here. In WSDM the navigation for each audience class is designed separately and the navigational model of the web site is the collection of disjoint navigation tracks of the several audience classes, linked by an initial element. Since different user groups could need to navigate through the same track, the navigational structure of a application should not be disjoint. In our approach different user groups can navigate through the same track, because OOHDM models permit specifying applications with complex navigations and behavior and not simply read-only.

Although OOHDM currently allows a simple form of navigation customization according to a user role, a more complete model is needed. We are currently developing an extension to OOHDM that will allow the specification of customized user views, based on user roles and identity.

The approach presented here has been used in several real-life applications being developed, with encouraging results. For the example mentioned in this work, 23 users of six different classes were interviewed, eliciting 169 tasks in 364 scenarios, later filtered down by the project managers into 63 tasks and 128 scenarios. The final application has 39 UIDs, resulting in 61 conceptual classes, 29 navigation classes, 47 InContext classes, 45 contexts, and 41 access structures.

The formal validation of this approach type is extremely difficult, but it was observed that, besides the easy adoption by the users in the validation process, it was possible to execute applications project in smaller time, and with fewer revisions than in previous projects of similar dimensions.

5. References

1. Carroll, J.M. *Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley & Sons, 1995.
2. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading, 1999.
3. De Troyer, O. Audience-driven Web Design, in Conceptual Modeling in the Next Millennium, Rossi, M. (ed.), CRC Press, USA, 2000
4. Elmasri, R., and Navathe, S.B. *Fundamentals of Database Systems*, Second Edition, Benjamin / Cummings, 1994.
5. Erskine, L.E., Carter-Tod, D.R.N., and Burton, J.K. *Dialogical techniques for the design of web sites*. Int. Journal Human-Computer Studies, 47 (1997), 169-195.
6. Garzotto, F., Paolini, P., and Schwabe, D. *HDM - A Model-Based Approach to Hypertext Application Design*. ACM Transactions on Information Systems 11, 1 (January 1993), 1-26.
7. Güell, N. *User Centered Navigation Design of Hypermedia Applications* (in Portuguese). Tech. Report MCC10/98, Departamento de Informática, PUC-Rio (1998). 18p. Available in ftp://ftp.inf.puc-rio.br/pub/docs/techreports/98_10_schwabe.pdf.gz
8. Imaz, M., and Benyon, D. How *Stories Capture Interactions*, in Proceedings of Human-Computer Interaction - INTERACT'99, IOS Press, 321-328.
9. Isakowitz, T., Stohr, E., and Balasubramanian, P. *RMM: A methodology for structuring hypermedia design*. Commun. ACM 38, 8 (August 1995), 34-44.
10. Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process*, Addison-Wesley, 1999.

11. Lange, D. *An Object-Oriented Design Method for Hypermedia Information Systems* (1994), 366-375.
12. Rossi, G., Schwabe, D., and Lyardet, F. *Improving Web information systems with navigational patterns, in Proceedings of 8th International World Wide Web Conference* (Toronto, Canada, May 1999), Elsevier Science, 589-600.
13. Rossi, G., Schwabe, D., and Lyardet, F. *Web Application Models Are More than Conceptual Models*, in Proceedings of ER'99 (Paris, France, November 1999), Springer, 239-252.
14. Schwabe, D., and Rossi, G. *An object-oriented approach to Web-based application design*. Theory and Practice of Object Systems (TAPOS) (October 1998), 207-225.
15. Sutcliffe, A.G., Maiden, N.A.M., Minocha, S., and Manuel, D. *Supporting Scenario-Based Requirements Engineering*. IEEE Transactions on Software Engineering 24, 12 (December 1998), 1072-1088.
16. Vilain, P., Schwabe, D., de Souza, C. S.: *Use Cases and Scenarios in the Conceptual Design of Web Applications*. Technical Report MCC 12/00, Departamento de Informática, PUC-Rio (2000).
17. Vilain, P., Schwabe, D., de Souza, C. S.: A Diagrammatic Tool for Representing User Interaction in UML. To appear in UML 2000 -Third International Conference on the Unified Modeling Language, (York, UK, October, 2000).