A Development Frame for Web Sites

E. Crivello and G. Reggio[2]

DISI, Università di Genova, Italy

Abstract. In this paper we present a "development frame" for web sites.
Technically, a development frame consists of
– a problem frame [7], i.e., a pattern that provides a precise conceptual model of what is the problem to be solved;
– and a precise method based on UML to be used in that specific case.
The development frame for web sites presented in this paper have been defined by specializing the one proposed by Astesiano and Reggio in [3, 2].

1 Introduction

In this paper we tackle the problem of the UML-based development of web sites, by presenting the corresponding "development frame".

Patterns are ready-to-use structures drawn from experience, and it is now quite widespread to use them to help systems development. Various kinds of patterns are available, problem frames propose an overall problem structure [7], architectural styles are useful to provide an overall structure for the system, while design patterns are more appropriate when structuring the design before coding, thus patterns may differ in granularity. A *development frame* is a pattern at a larger grain, roughly corresponding to the "structure" of the way followed to solve a particular kind of development problem.

Technically, a development frame consists of
– a problem frame [7], i.e., a pattern that provides a precise conceptual model of what is the problem to be solved;
– and a precise method based on UML to be used in that specific cases.

For each problem frame, a diagram is settled, showing the involved domains, the requirements, the design, and their interfaces. Problem frames are also presented with the idea that, once the appropriate problem frame is identified, then the associated development method should be given "for free". While the structuring concepts brought by problem frames seem highly valuable to help start the development effort, by showing clearly the items to consider and the general tasks to do, we still need to propose development methods associated with them. A development frame thus complements a problem frame with a method specifically designed for the cases captured by it. Such method gives detailed guidelines to develop all the required artifacts through a dedicated choice of appropriate UML diagrams together with predefined schemas or skeletons for their contents. Thus, a development frame provides a more direct path to the UML models, which saves time (no long questions about which diagrams to use and how) and improves the models quality (relevant issues are addressed, a uniform style is offered).

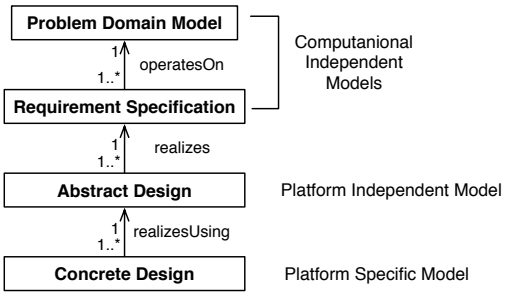
Various development frames covering different problems have been proposed, see e.g., [4] for classical development problems, as control systems or transformation applications, and [5] for enterprise applications, also if in such papers the term "development frame" is not used.

In this paper we present a development frame for web sites, where the method part have been defined by specializing the precise UML-based method of Astesiano and Reggio, introduced in [3, 2].

Many development methods for Web sites and Web based applications centered on the use of the UML can be found in the literature and in the current practice, see Sect. 6, covering different phases and activities in the development process and very different kinds of applications so why bother to propose another one. First of all, the problem frame part helps precisely present in which cases the considered method may be applied, clarifying precisely how is intended the term "Web Site"; thus it will be possible to avoid mismatches between the method and the problem, frequently due to the ambiguity of the terminology used in this field. For example, a system for handling the procedures for the Quality Standard in a big logistic company using the web for entering the data will be not considered a web site, and will be developed using more appropriate methods. Moreover, the proposed method cover all the phases of the development from the modelling of the domain, to the requirement specification till to the (abstract/platform independent) design, guiding the developer to consider all the real relevant issues, such us who is responsible for updating dynamic pages, and if there is any security policy for accessing some data, and not immediately to start to discuss about html pages and buttons.

Another advantage of using a development frame, as the one we propose in this paper for the web site, is that the developer does not need to investigate how to apply the chosen method, here the precise UML based of Astesiano-Reggio, to the particular problem under consideration, here the development of a web site. Instead, s(he) can reuse the tailorization associated with the development frame, making the development more speedy.

In this paper, we present the development frame for web sites by giving in Sect. 2 the web site problem frame and in Sect. 3, 4 and 5 the associated method, by showing how to produce the problem domain model, the requirement specification, and the abstract (or platform independent) design, from which various platform specific designs may be derived (not considered in this paper). Thus, the proposed method follows the MDA approach as summarized by the following schema.



For lack of room we do not detailed describe how the proposed specialized method for web sites is derived by the general purpose precise method of Astesiano-Reggio [3, 2], but directly we say which models have to be produced and for which aim.

Finally, in Sect. 6 we draw some conclusions and consider related work.

2 (Administrated Dynamic) Web Site: The Problem Frame

2.1 Web Sites

When we have to develop a web site, one of the most important things we have to consider is who will use the site and what are hers/his needs. Starting a simple analysis, we can divide users in four categories, each one having a role in the context of a site use:

Unknown visitors: the casual visitors of the site;
Registered visitors: visitors that can access reserved data;
Content managers: visitors that can access to the backend system; they can read, write and update contents;
Registered content managers: visitors that can read, write and update reserved contents;
Administrators: users that can define data access rules.

Basing on roles described upon and on the information flow, we can distinguish between:

Static Web Site It is a site in which all pages are shown one by one and contain all the information that we have decided to publish, once for all. A Static Web Site has a finite number of pages, always quantifiable. All its users are unknown visitors.
Dynamic Web Site A Dynamic Web Site is equipped with functionalities that provide the dynamic contents of pages. This kind of site can be browsed by (registered) visitors and be updated by a given set of (registered) content managers.

Moreover, a web site can be an

Administrated Web Site in which administrators manage visitors/content managers and their rights (like what is the scope of a registered content manager or which pages that can be viewed by a registered visitor).

In this paper we consider the administrated dynamic web sites, and proposed a development frame for them; however, it is trivial to derived from it frames for more specific kind of sites, e.g., static sites.

When speaking of web based systems, a term frequently used is "web applications". In our opinion, web applications are not web sites following our definition, because web sites are characterized by info to be made publicly available, whereas web applications are applications using the web technology for realizing the interface to their users. In most of the case, the so called "web applications" are enterprise applications where the presentation tier is realized via web, e.g., an e-commerce site.

Our approach, does not propose a development frame for web applications, since their common feature is the use of a particular realization technology, but they do no solve a common problem. Whenever the application to develop is a web-application, obviously we should not use the frame for web sites, but we should look instead for the frame appropriate for the application forgetting the web aspects (e.g., see [4] for enterprise applications, and [5] for classical cases). Then, we should propose a particular design for such frame using the web technology. We think, that in this way the use of

the web technology will not obscure the other relevant aspects of the development problem to tackle.

In this section we use as a running example COURSES, a site devoted to the information about the courses given at a university (programs, prerequisites, teachers, and the course related activities, i.e., classes and exams). The teachers provide the information about their courses, some administrative clerks take care instead of the information about the activities, and the dean may add and remove courses. Only teachers, students and clerks may see the information about the course activities.

2.2 Problem Frame

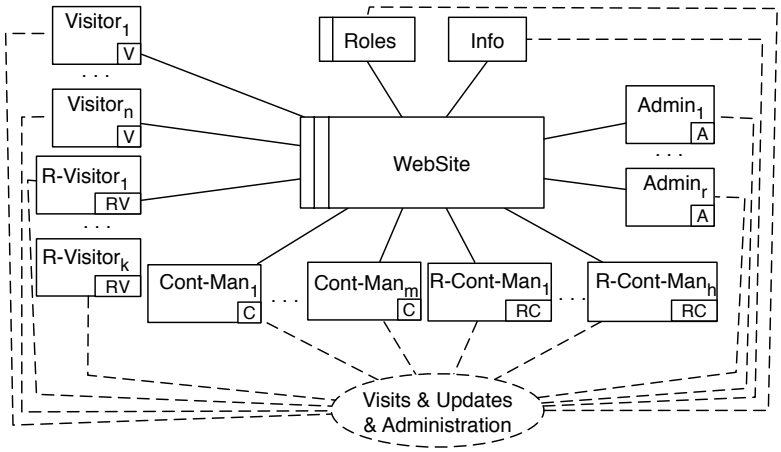


Fig. 1. Administrated Dynamic Web Site Problem Frame

In Fig. 1 we present the problem frame for the Administrated Dynamic Web Sites using the visual notation introduced by M. Jackson in [7]; that we briefly summarize below.

The *WebSite* is a machine domain (marked by two lines on the left), i.e., the software to be built. *Roles* (marked by one line on the left) is a designed domain, i.e., a data structure that may be freely designed and that will be used by the machine. The plain boxes are given domains, i.e., existing domains that are not responsibility of the developer; s(he) must not design/develop them, they already exist and must be carefully described to correctly develop the machine. Jackson considers three kinds of domains: lexical or inert domain (data), causal (able to react to external events), and biddable (usually persons able to send requests/commands to the machine). In this frame, *Info* and *Roles* are lexical, whereas all the other given domains are biddable. A solid lines connecting two domains means that they share some common phenomena, such as exchanged messages or data. The dashed oval represents the requirements on the machine to be built; a dashed line connecting it to a domain means that the domain is referenced in the requirements, whereas a dashed arrow means that the requirements are constraining that domain.

A problem frame can be seen as a conceptual model of the development problem under consideration, listing all its parts, distinguished between those already existing and those which have to be given by the developer, and their mutual relationships.

Let now us analyse the parts of the Administrated dynamic Web Site Problem frame.

Info is lexical domain, and corresponds to the information that the web site should made publicly available.

Visitor$_i$, $i = 1, \ldots, n$, (marked by V) and *R-Visitor*$_i$, $i = 1, \ldots, k$, (marked by RV) are the categories of the unknown and registered visitors of the web site, i.e., the navigators of the site. *Cont-Man*$_j$, $j = 1, \ldots, m$, (marked by C) and *R-Cont-Man*$_j$, $j = 1, \ldots, m$, (marked by RC) are the categories of unknown and registered content managers of the web site, i.e., those entities able to provide the information inside the site, The registered visitors/content managers are known by the site and must be identified before to access it, whereas nothing is assumed/required about the unknown ones. *Admin*$_l$, $l = 1, \ldots, h$, (marked by A) are the categories of the site administrators, i.e., who is taking care to supervise the registration procedures.

Visits&Updates&Administration represents the requirements on the machine to be built; in this case they are just a list of visits (navigations), of updates and of administrative procedures that must be possible on the web site. An update describes a possible way to change the information made available by the site; whereas an administrative procedure describes how to register either visitors or content managers and give them some security related role. The *Roles* domain, which should be designed by the developer, models which are the roles used in the site to grant the access to information and/or to perform updates on the site.

The Web Site problem frame presented above helps to makes clear that whenever we are going to develop a web site we should have:
– some information *Info* to make publicly available on the web that may be dynamically changed. Notice that because the information are a given domain, then the responsibility of the developer is to find out which they are and not to invent/design them;
– visitors to access such info and content managers to provide/update them;
– administrators to supervising the registration procedures assigning to visitors and content managers security roles. Notice, that instead, the roles have to be designed by the developer to guarantee the required access policy.

The requirements given in terms of required visits, updates and administrative procedures of the web site should help clarify for which scope the site will be built, how can evolve and how it should be administrated.

In Fig. 2 we show the instance of the Web Site problem frame corresponding to the COURSES site. Notice that here we have:
– one category of unknown visitors *ContentsInterested*, someone interested in the course contents, i.e., the students and the teachers (to see what is taught/required in the courses), and people outside the university to see what is taught in such university;
– one category of registered visitors *ActivitiesInterested*, someone interested in the course related activities: students, teachers and administrative clerks wanting to know if some students/teachers/rooms are free or occupied and in this case by what;
– three categories of registered content managers
* *Dean*, which may add and remove courses,
* *Teacher*, which may change the info about hers/his courses,

* *Clerk*, which take care of the information about classes and exams;
– one category of administrators *Admin*, which take care of all registration procedure.

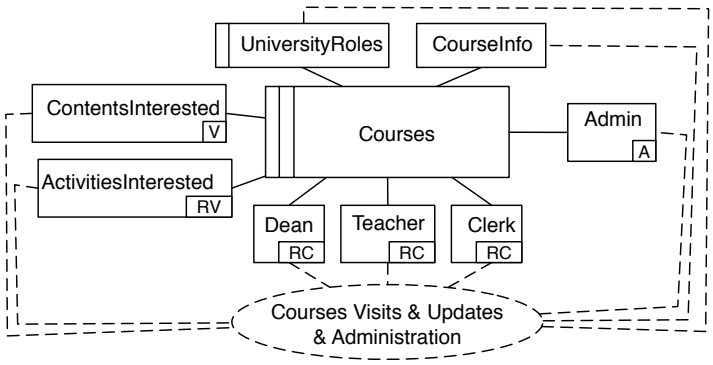


Fig. 2. COURSES Case Study

3 Information Model (Problem Domain Model)

The precise method of Astesiano-Reggio [3, 2] as a first step proposes to give a model of the problem domain, i.e., of those aspects of the real world that are relevant for the system to be developed, for providing a solution to the problem under consideration. In the case of the development of a web site the problem domain corresponds to the information to make available, i.e., to the *Info* part of the relative problem frame, introduced in Sect. 2.

Following [3, 2] the *Info* part of the Web Site problem frame will be specified by a UML model consisting of a class diagram where

– all classes are datatype and have no operations,
– specialization, aggregation/composition, named binary association (obviously with their multiplicities) may appear,
– and invariant constraints may be attached to the classes.

We name this model Information Model, since it abstractly models the information made available by the web site.

In Fig. 3 we present the model of the information considered by the COURSES site. In this diagram and in all the paper, for simplicity, we drop the association multiplicity whenever it is "1".

4 Requirement Specification

The problem frame of Fig. 1 says that the requirements for a web site consist of a set of visits, updates and administrative procedures that should be possible on the site itself. The use cases are quite satisfactory to express visits/updates/administrations of

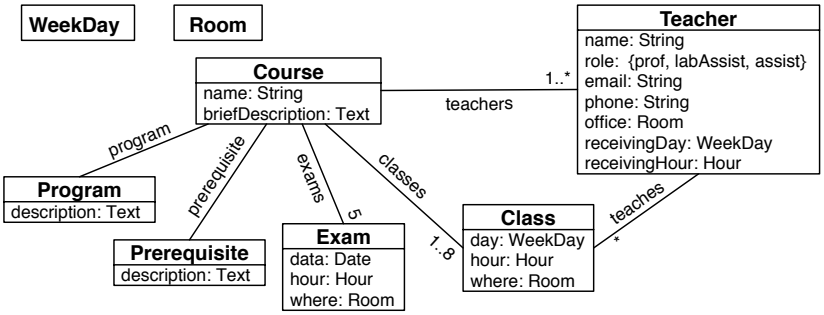


Fig. 3. COURSES: Information Model

the site, also if they need to be specialized to those particular activities. We call these specialized use cases respectively "Visit Cases", "Update Cases" and "Administration Cases".

A visit case will have exactly one actor, the unknown or registered visitor performing the visit; and a step in one of its scenarios either corresponds to the actor asking for some specific information, or to the web site showing some information.

An update case will have exactly one actor, the unknown or registered content manager performing the update; and a step in one of its scenarios either corresponds to the actor asking for some specific information, or to delete/add/modify an information, or to the web site showing some information or confirming the modification or refusing the modification and showing some error message.

An administration case may have as actors administrators or unknown visitors, and it is a standard use case.

The required visits will impose some navigation structure over the information considered by the web site (notice that the Information Model introduced before does not consider the possible way to navigate over the information but just which are such information and the logical/conceptual relationships among them). Furthermore, some visits may require additional information which can be derived by the original ones; think for example the case when some statistics over some data are required. Thus the Requirement Specification must include also a new model of the information making explicit the mandatory navigations and any information required in a visit, that we call Navigation Model. To precisely define the required updates we need to introduce in the Navigation Model also appropriate operations to express how the information may be modified. Finally, to express the required access policy we mark the links and the operations of the Information Model with the roles that are allowed to navigate/perform them. Those roles will be precisely modelled by a Roles Model, corresponding to the *Roles* part of the Web Frame, defined in the following.

Summarizing, the Requirement Specification of a (administrated, dynamic) web site consists of a Use Case Diagram, a Navigation Model, a Roles Model, and some Visit/ Updates/Administration Case Descriptions (one for each case appearing in the Use Case Diagram).

Use Case Diagram It is a UML Use Case Diagram, where the actors are stereotyped by ≪v≫, ≪rv≫, ≪c≫, ≪rc≫, ≪a≫ corresponding to the categories of unknown/ registered visitors/content managers and administrators of the site, and each use case is stereotyped either by ≪visit≫ (with a unique actor of kind ≪v≫ or ≪rv≫) or ≪update≫ (with a unique actor of kind ≪c≫ or ≪rc≫) or ≪a≫ (whose actors are either ≪v≫ or ≪a≫).

The method requires that each actor, is detailed described making clear hers/his motivation or duty, e.g., by attaching a note to each actor in the diagram. To fill these information will help to check whether the aims of the web site are well motivated and that who must take care of filling the contained information and supervising the accesses have been clearly identified.

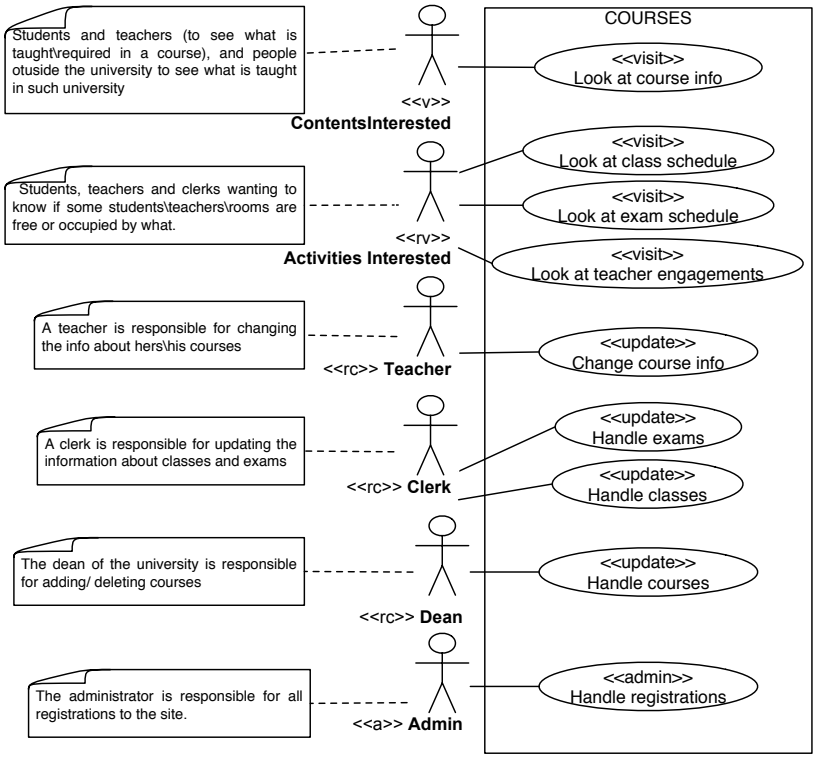


Fig. 4. COURSES: Use Case Diagram

In Fig. 4 we show the use case diagram part of the Requirement Specification of COURSES. The actors of the use case diagram allow to precisely define which are the categories of visitors, content managers and administrators.

Roles Model The Roles Model is a class diagram, where classes define the possible roles, and specialization models the hierarchy among the roles.

In Fig. 5 we present the Roles Model for the COURSES site.

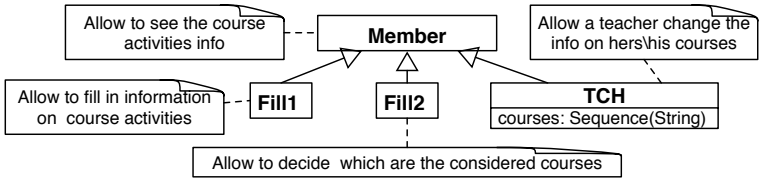


Fig. 5. COURSES: Roles Model

Navigation Model The Navigation Model is a class diagram presenting all the information required by the visit cases together with the navigation structure over them, and all the modification operations required by the updates.

It is defined by extending the Information Model given before by adding all the derived information and the navigable links required by the visit cases. A *link* from one info to another is an association navigable in one sense stereotyped by ≪link≫, and visually presented as a simple arrow; whenever the linked info is determined using some values, the link will be an association class whose attributes correspond to such values; for readability the association class attributes are depicted between graph parentheses near the association name. The Navigation Model must have a class named Home, which will be the starting point of the navigations. Obviously, it may include invariant constraints to make precise which are the information reached by the navigation links.

The various information classes may have operations, modelling the possible modifications performed by the required updates. Obviously, such operations will be defined either by pre-post constraints or by associated methods.

The links and the operations may have an additional tagged value containing the roles which are allowed to navigate/execute them (visually presented enclosed within double square parentheses).

In Fig. 6 we present the Navigation Model of the COURSES site.

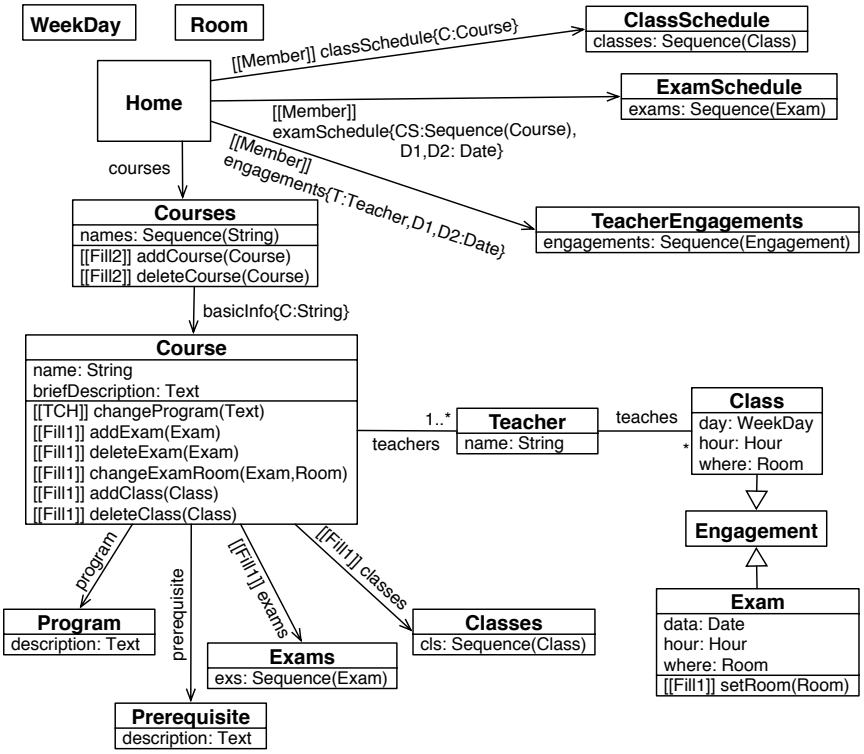
Use Case Descriptions The precise method of Astesiano-Reggio [3, 2] requires to describe the use cases by means of state machines having a particular form; but because the visit/update/administration cases are rather simple, we prefer here to present them in the classical way by means of scenarios written using natural language, following the template proposed in [12].

Visit Case A visit case presents some visits (navigations) that the web site should allow.

A visit case must have exactly one actor stereotyped by ≪v≫ (visitor) or ≪rv≫ (registered visitor).

A step of a scenario of a visit case is
– either the actor that while seeing some info asks to see other specific information,
– or the web site showing the required information (possibly error messages are considered special information).

A visit case must have a precondition that consists of assuming that the actor is looking at some specific information, indicated in the template by **Entry point**. If the actor is a registered visitor, then the case precondition must also specify which are the (security) roles authorizing the visit, they will be indicated in the template by **Roles**.



context Home inv:
  courses.names = Course.allInstances.name and
  classSchedule{C} = courses.basicInfo{C}.classes and
  examSchedule{CS,D1,D2} = Course.allInstances->select(CS->includes(name)).exams and
  engagements{T,D1,D2} =
    (Teacher.allInstances->select(name = T).teaches union
    Teacher.allInstances->select(name = T).teachers.exams->select(date< D2 and date > D1))
context Courses inv: basicInfo{C}.name = C
context changeProgram(P) post: program.description = P
context addExam(E) post: exams->includes(E)
context deleteExam(E) post: exams->excludes(E)
context changeExamRoom(E,R) post:
  exams->excludes(E) and exams->includes(E.setRoom(R))
context addClass(C) post: classes->includes(C)
context deleteClass(C) post: classes->excludes(C)
context addCourse(C) post: Course.allInstances->includes(C)
context deleteCourse(C) post: Course.allInstances->excludes(C)
context Exams inv: exs->size= 5
context Classes inv: cls->size<= 8 and cls->size>= 0

Fig. 6. COURSES: Navigation Model

The visit case description must be in accord with the Navigation Model and with the Roles Model. Precisely, the shown information should be defined in the Navigation Model, the information asked while looking some other one must be reachable by a (navigation) link, the referred roles should be defined in the Roles Model and should be in accord with the role markings of the Navigation Model.

Below we give the description of one of the visit cases of the COURSES web site; the remaining ones can be found in the complete version [6].

Visit Case Look at course info
Intention in context: A person wants to know the program and/or the prerequisite of a course, to see if s(he) if it is interesting, or if s(he) can take it, or if it may be reused in another degree, or what knows a person that took such course.
Visitor: Contents Interested
Entry point: Home
Main success scenario:
1. The Contents Interested asks all courses
2. The Web Site shows the list of all course names.
3. The Contents Interested asks for a specific course.
4. The Web Site shows the course basic info.
5. The Contents Interested asks for the course program.
6. The Web Site shows the course program.
Extensions:
5a. The Contents Interested asks for the course prerequisite.
5a.1. The Web Site shows the course prerequisite.

Update Case An update case presents some ways to modify the information made available by the web site that should be possible perform.

An update case must have exactly one actor stereotyped by ≪c≫ (content manager) or ≪rc≫ (registered content manager).

A step of a scenario of an update case is
– either the actor that while seeing some info requires to see other specific information,
– or the web site showing the required information,
– or the actor that while seeing some info requires to modify them,
– or the web site confirming the required modification,
– or the web site informing that the modification cannot be done and explaining why.

Similarly to the visit cases, the template for an update case includes an **Entry point** (i.e., a precondition requiring that the actor is looking at some specific information). Moreover, if the actor is a registered content manager, then the template includes also a **Roles** part (i.e., a precondition specifying which are the roles allowing to perform the update).

The use case description must be in accord with the Navigation Model and with the Roles Model. Precisely, the shown information should be defined in the Navigation Model, the information asked while looking some other one must be reachable by a (navigation) link, the referred roles should be defined in the Roles Model and should be in accord with the role markings of the Navigation Model, and any required modification of an information should correspond to an operation of its class.

Below we present an update case description of the COURSES site, the other ones are in [6].

Update Case Handle exams
Intention in context: A clerk takes care of putting in the site the information about the exams of a course.
Content Manager: Clerk
Roles: Fill1
Entry point: Course
Main success scenario:
1. The Clerk asks to see all the exams of a course.
2. The Web Site shows all the exams.
3. The Clerk selects one of that exams.
4. The Web Site shows the info for such exam (day, hour, room).
5. The Clerk asks to delete it.
6. The Web Site asks for a confirmation.
7. The Clerk confirms.
8. The Web Site shows the list of all exams.
Extensions:
6a. The Clerk does not confirm.
The case continues at step 8.
3a. The clerk asks to add an exam giving the relative info (day, hour and room).
3a.1. The Web Site confirms.
The case continues at step 8.
5a. The Clerk inserts the new room.
5a.1. The Web Site confirms, and shows the new information for such exam.
The case continues at step 8.

Administration Case An administration case describes how to perform the registration to the site of either visitors or content managers.

The actors of an administration case must be stereotyped by ≪v≫ (visitors) or ≪a≫ (administrators) and can be any number.

The scenarios of an administration case are as in a standard use cases.

The use case description must be in accord with the Navigation Model and with the Roles Model. Precisely, the roles assigned to the registered ones should be defined in the Roles Model, and the rights assigned to a role should be in accord with the role marking of the Navigation Model.

Administration Case Handle registrations
Intention in context: The Admin autonomously wants to register people to the web site.
Actor: Admin
Entry point: Home
Main success scenario:
1. The Admin asks to start the registration procedure.
2. The Web Site asks to give the data of the person to register.
3. The Admin gives the data.
4. The Web Site asks which are the (security) roles for such person.
5. The Admin gives them.
6. The Web Site shows the data and the roles for the person and asks to confirm of the registration.
7. The Admin confirms.

5 Abstract or Platform Independent Design

We present here the specialization to the particular case of the web sites of the general-purpose precise method of Astesiano-Reggio [3, 2] for presenting the abstract or plat-

form independent design. The abstract design is presented by a UML model consisting of a Static View, i.e., a class diagram, giving the site structure, and of a set of Interface View.

Static View The Static View is a class diagram where classes and associations are stereotyped in such a way to denote specific web ingredients; among the used stereotypes we have

– ≪page≫ stereotype of class (static web page), the class attributes represent the elements appearing on the page.
– ≪link≫ stereotype of association or of association class (navigation link between two pages), similar to the links of the Navigation Model of Sect. 4, must be navigable in one sense; in case of an association class its attributes are written between graph parentheses near the association name.

A link to a page may be marked by a role enclosed between doubles square brackets ([[ ]]), to model the fact that the access to such page (and to anything reachable from it) is reserved to registered users having such role.

– ≪store≫ stereotype of class (persistent data), to model the persistent data used to generate the dynamic pages.
– ≪dynamic page≫ stereotype of class (dynamic web page, i.e., whose contents depend on the persistent data). Usually, only a part of a dynamic page depends on the persistent data, thus, we split the attribute compartment in two parts: the first contain the fixed attributes, and the second those depending on the persistent data.
– ≪query≫ stereotype of association (querying the persistent data), to model the fact that a dynamic page uses some persistent data to generate its content; a ≪query≫ is depicted with a line between the two classes with a small black ball attached to the ≪store≫ class.
– ≪upd≫ stereotype of association (updating the persistent data), it links a page (static or dynamic) class and a store class, and has a tagged value containing a list of method definitions for the store class, which will be called by the page class. An update is represented by a line with a small black box on the side attached to the page class. To improve the readability of the diagram, those methods may be reported in a note in the diagram or may be given apart.

The class diagram must include at least a ≪page≫ Home, and may contain also standard classes and associations, which are used to model the information contained by the pages.

Each parameterized link leaving a page class, say P, must be defined by an invariant constraint attached to P saying how the (fixed) attributes of the target class are determined by the link parameters. The dynamic attributes of a dynamic page class must be defined by an invariant constraint saying how their values is determined by the stores reachable by the query associations.

In Fig. 7 we present the Static View of the COURSES case study. The constraints are given below.

```
context RA: ReservedArea inv:
  RA.classSchedule{C}.course = C and
  RA.examSchedule{CS,D1,D2}.courses = CS and
  RA.examSchedule{CS,D1,D2}.from = D1 and
  RA.examSchedule{CS,D1,D2}.to = D2 and
```

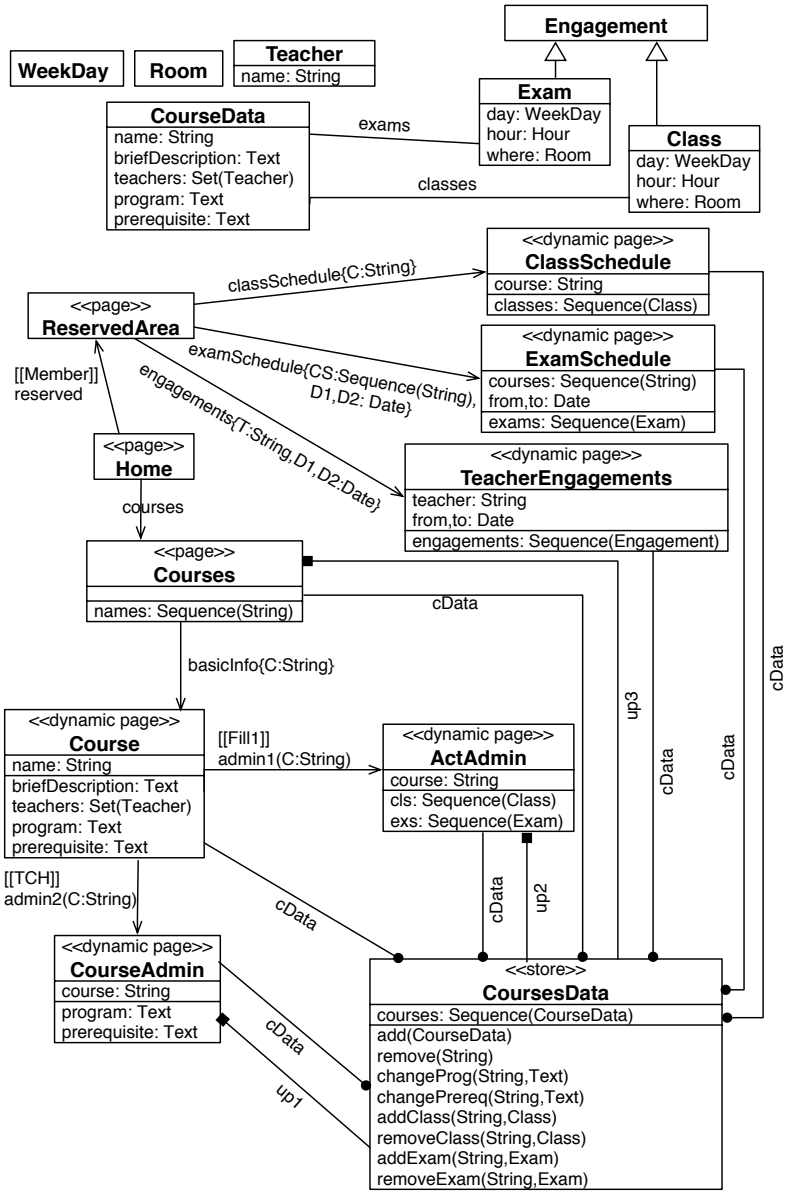


Fig. 7. COURSES: Static View

```
   RA.engagements{T,D1,D2}.teacher = T and
   RA.engagements{T,D1,D2}.from = D1 and
   RA.engagements{T,D1,D2}.to = D2
context CS: ClassSchedule inv:
   CS.classes = cData.courses->select(C | C.name = CS.course).classes
context ES: ExamSchedule inv:
   ES.exams = cData.courses->select(C | ES.courses->includes(C.name)).exams
      ->select(E.date< ES.to and E.date > ES.from)
context TE: TeacherEngagements inv:
      TE.engagements = (cData.courses->select(C | C.teachers.name->includes(TE.teacher)).classes)
   union
      (cData.courses->select(C | C.teachers.name->includes(TE.teacher)).exams
         ->select( E  | TE.to and E.date > TE.from))
context CS: Courses inv:
   CS.names = CS.cData.name and CS.basicInfo{C}.name = C
context CC:Course inv:
   CC.briefDescription = CC.cData->select(CX | CX.name = C.name).briefDescription and
   CC.teachers = CC.cData->select(CX | CX.name = C.name).teachers and
   CC.program = CC.cData->select(CX | CX.name = C.name).program and
   CC.prerequisite = CC.cData->select(CX | CX.name = C.name).prerequisite and
   CC.admin1{C}.course = C and CC.admin2{C}.course = C
context CA:CourseAdmin inv:
   CA.program = CA.cData.courses->select(CX | CX.name = CA.course).program and
   CA.prerequisite = CA.cData.courses->select(CX | CX.name = CA.course).prerequisite
context AA:ActAdmin inv:
   AA.cls = CA.courses->select(CX | CX.name = AA.course).classes and
   AA.exs = CA.courses->select(CX | CX.name = AA.course).exams
context CoursesData::addCourse(CD) post:
   courses->includes(CD)
context CoursesData::removeCourse(C) post:
   courses.name->excludes(C)
context CoursesData::changeProg(C,P) post:
   self.courses->select(CX | CX.name = C).program = P
context CoursesData::changePrereq(C,P) post:
   courses->select(CX | CX.name = C).prerequisite = P
context CoursesData::addExam(C,E) post:
   courses->select(CX | CX.name = C).exams->includes(E)
context CoursesData::removeExam(C,E)  post:
   courses->select(CX | CX.name = C).exams->excludes(E)
context CoursesData::addClass(C,CL) post:
   courses->select(CX | CX.name = C).classes->includes(CL)
context CoursesData::removeClass(C,CL) post:
   courses->select(CX | CX.name = C).classes->excludes(CL)
context CourseData inv:
   exams->size= 5 and classes->size<= 8 and classes->size>= 0
upd1
   changeProg, changePrereq
upd2
```

addClass,removeClass, addExam, removeExam

upd3

addCourse, removeCourse

The Static View of Fig. 7 shows the decisions made by the designer for the COURSES site; precisely, the structuring in (static and dynamic) pages of the site, which persistent data will be used (store class CoursesData), where the modifications to such data may be required (the upd's), and how the dynamic pages depends on the persistent data (queries).

Interface View The precise method of Astesiano-Reggio [3, 2] requires for each boundary class, such as the page classes used in this case, to provide an Interface View, which is a description of the GUI associated with such class. In this case, the Interface View will consist of the graphical layout of the page classes, showing how the data containd in the attributes must be arranged, which graphical devices are used to call the links associated with the page, and how the associated updates may be called.

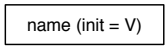
An Interface View associated with a (dynamic) page class PG consists of a schematic representation of the aspect of such page, as it will be rendered by a browser. It is just a drawing showing how to layout within a box (representing the corresponding web pages) the various elements (pure graphical items, e.g., logos; the information contained in the page class atttributes together with hints on how to represent them; the links leaving the class together the chosen visual mechanism to make possible to click them; buttons and forms to call the updates associated with the class).

Below, we list all the elements that can be put on an drawing realizing an Interface View.

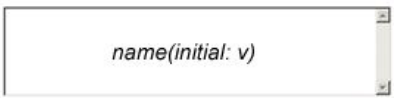
Link represented by an underlined string or some other visual device (e.g., a button), followed by the name of the corresponding ≪link≫ association, written in italic style.
Whenever, the page reacheable throught the link is reserved to user with a specific role, then such role will be written enclosed between double square bracktes ([[,]]) near the link name. This means, that when the link is selected, then the website must ask to the user to autheticate herself/himself; but to keep the design model quite simple, and because the interface for the authentication may be quite standard, we do not detail it.

information these information are contained in the attributes of the page class, and will be represented by the name of such attributes written in the italic style. If the type of an attribute is structured, it is possible to say how the value contained in the attribute is decomposed in parts in a note attached to the Interface View, and to depict instead its parts on the drawing (again using the italic style).

textbox (used to insert string values) Its name will written inside the box using the italic style; and if it has an initial value, say V, then we will write (initial = V) near the textbox name.



textarea (used to insert text values) Its name will written inside the box using the italic style; and if it has an initial value, say V, then we will write (initial = V) near the textarea name.



button (used to call update methods without parameters), we will write near the button using the italic style the called method (which must be part of an update leaving the page class)

form contains a combination of textboxes, textareas and buttons; near each button we will write using the italic style either a call of an update method, or of a parametric link; the arguments of the method or of the link will be expressions built by the components of the form and by the class attributes

The Interface View's associted with the page class Home of COURSES is given in Fig. 8; the other can be found in Appendix B.

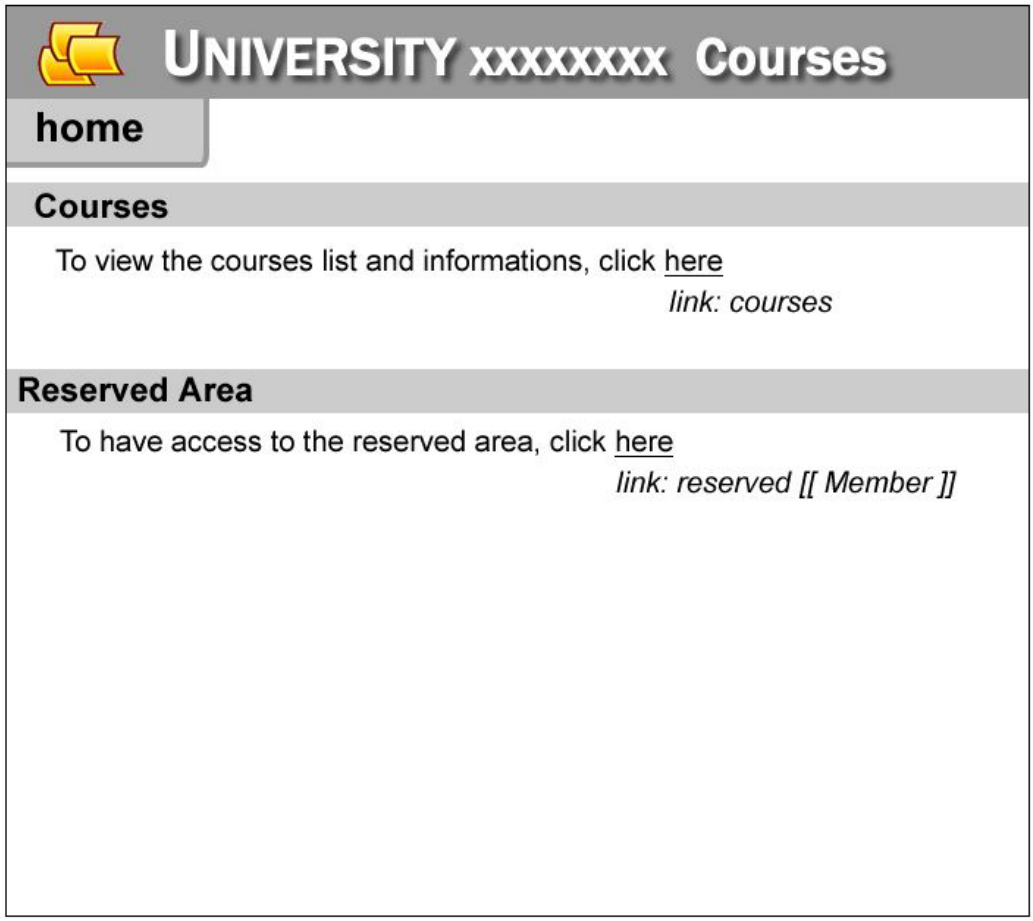


Fig. 8. Home: Interface View

6 Related Work and Conclusions

We have presented a "development frame" for web sites; that is a precise conceptual description of what is the problem of developing a web site, by means of a Jackson Problem Frame [7], associated with a UML-based development method specialized for that particular kind of problem.

The associated method has been obtained by specializing to the web site case the general purpose precise method of Astesiano-Reggio [3, 2], follows the MDA approach, and tightly guides the developer during the development, stating precisely which UML diagrams have to be produced and which form they should have. Moreover, this method helps the developer to concentrate on the relevant issues of a web site, such as which are the information to show, which are the intended users, who is responsible for the information, which are the access rights and so on, avoiding that they are blurred by the peculiar aspects the web technology (e.g., pages, forms, buttons, menus, ...).

Furthermore, the web site problem frame should help avoid to make confusion between developing a web site and very different kinds of applications, which just use the web technology for the user interface, such as enterprise applications.

In this proposal we have used the notation proposed by M. Jackson [7] for representing the problem frames. It is clear that we could instead have defined a UML profile, and we are going to investigate this possibility. In this paper we have used UML 1.3, but we are going to migrate to UML 2.O to take advantage of the better functionalities to modular decompose complex diagrams. Moreover, we are going also to consider the last step of the concrete design oriented to a particular kind of web technology (e.g., php, JSP, ASP), investigating if and how the abstract design model has to be modified, or if it is sufficient to mark it with some specific information.

Among the many proposals concerning web development and their modeling, we choose a few of them to make some comparisons.

UWE (UML-based Web Engineering) [11, 9, 10] is the most similar work to ideas presented in this paper. UWE, defines a UML profile for modelling web applications, and comprises an initial phases of requirements analysis, in which a use case model is produced. This can be compared to our Requirement Specification Sect. 4. Then, in the design process, a web application is described by three models: Conceptual Model, Navigation Model and Presentation Model. Their Conceptual Model (something similar to our Information Model Sect. 3) describes classes of object participating in the system and their relationships. In this model some of the conceptual classes have a particular attribute (isNavigationRelevant) that indicates which classes are relevant for the Navigation Model. The Navigation Model models the navigation structure of the web application. We do something like this in the Navigation Model in Sect. 4. In this model for each class of the Conceptual Model that is navigation relevant a "navigation class" is added, maintaining the association from the Conceptual Model; then other associations can be added to represent navigation path between navigation classes. Moreover "access primitives" (like index, menus, guided tour and queries) are added to model the possibilities for the user to navigate in the application. As last, in the Presentation Model the presentation structure of the web application is modeled. Also an ARGO plugin was implemented (ARGO-UWE) to support this kind of models.

The main difference with our works is that we first propose a development frame, and not just only a method, clarifying thus in which cases the method may be used; and we put more emphasize on guiding the developer to take into account all the fun-

damental aspects of developing a web site (information, visitors, information providers, access rights).

James Conallen [8] proposes a UML extention to model web applications, defining various stereotypes of classes (such as Server Page, Client Page, Form, Frameset and Target) and associations (links, submits, builds, redirect). Our stereotypes used in the design pahse are a bit more abstract, so we do not talk talk about Client and Server page, but only of pages and dynamic pages, basing the difference on the dependance between these classes and persistent data.

Then we cannot not cite WebML (Web Modeling Language) [1], that is a visual language used for modelling web data-intensive application. WebML follows the style of other modeling languages, like UML and ER; but, using our terminology, they consider a different problem frame.

References

1. In S.Ceri, P.Fraternali, A.Bongio, M.Brambilla, S.Comai, and M.Matera, editors, *Designing Data-Intensive Web Applications*, The Morgan-Kaufmann Series in Data Management Systems. Morgan-Kaufmann, 2002.
2. E. Astesiano and G. Reggio. Towards a Well-Founded UML-based Development Method. In A. Cerone and P. Lindsay, editors, *Proc. of SEFM '03*, pages 102–117. IEEE Computer Society, Los Alamitos, CA, 2003.
3. E. Astesiano and G. Reggio. Tight Structuring for Precise UML-based Requirement Specifications. In M. Wirsing, A. Knapp, and S. Balsamo, editors, *Radical Innovations of Software and Systems Engineering in the Future, Proc. 9th Monterey Software Engineering Workshop, Venice, Italy, Sep. 2002.*, number 2941 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2004.
4. C. Choppy and G. Reggio. A UML-Based Approach for Problem Frame Oriented Software Development . *Journal of Information and Software Technology*, 2005. To appear.
5. C. Choppy and G. Reggio. Requirements Capture and Specification for Enterprise Applications: a UML Based Attempt. Technical Report DISI-TR-05-10, DISI – Università di Genova, Italy, 2005. Available at `ftp://ftp.disi.unige.it/person/ReggioG/ChoppyReggio05c.pdf`.
6. E. Crivello and G. Reggio. A Development Frame for Web Sites – Complete Version. Technical Report DISI-TR-05-16, DISI – Università di Genova, Italy, 2005. Available at `ftp://ftp.disi.unige.it/person/ ReggioG/CrivelloReggio05a.pdf`.
7. M. Jackson. *Problem Frames: Analyzing and Structuring Software Development Problems.* Addison-Wesley, 2001.
8. J.Conallen. *Building Web Applications with UML-2th edition.* Addison-Wesley, 2002.
9. A. Knapp, Koch N, and G. Zhang. Modeling the Structure of Web Applications with ArgoUWE. In *Proc. Fourth Int. Conference on Web Engineering (ICWE04)*, number 3140 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2004.
10. A. Knapp, Koch N, G. Zhang, and H.-M. Hassler. Modeling Business Processes in Web Applications with ArgoUWE. In *Proc. UML'2004*, number 3273 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2004.
11. N.Koch, H.Baumeister, R.Hennicker, and L.Mandel. Extending UML to model navigation and presentation in web applications. In R.France and B.Rumpe, editors, *Proceeding of the Modelling Web Applications in the UML Workshop-UML'2000, York,England, oct 2000.*, 2000.
12. S. Sendall and A.Strohmeier. Requirements Analysis with Use Cases. `http://lglwww.epfl.ch/research/use_cases/RE-A2-theory.pdf`, 2001.

A COURSES: Use Case Descriptions

Visit Case Look at class schedule
Intention in context: A person wants to know the class schedule of a course, to know whether some students/teachers are free at some time.
Visitor: Activities Interested
Roles: Member
Entry point: Home
Main success scenario:
1. The Activities Interested asks for the class schedule of a course.
2. The Web Site shows the class schedule of that course.

Visit Case Look at exam schedule
Intention in context: A person is interested to see the exam schedule, to know when some exams will take place.
Visitor: Activities Interested
Entry point: Home
Roles: Member
Main success scenario:
1. The Activities Interested asks for the exam schedule indicating the courses of interest and a time period.
2. The Web Site shows the exam schedule relative at the selected courses and time period.

Visit Case Look at teacher engagements
Intention in context: A person is interested to see the engagements of a teacher.
Visitor: Activities Interested
Entry point: Home
Roles: Member
Main success scenario:
1. The Activities Interested asks for the engagements of a teacher selecting a time period.
2. The Web Site shows all classes and exams scheduled in the selected period of courses taught by the given teacher.

Update Case Change course info
Intention in context: A teacher wants to change the info about one of hers/his courses.
Content Manager: Teacher
Roles: TCH
Entry point: Course
Precondition: The Course is one of those taught by Teacher.
Main success scenario:
1. The Teacher asks to see the program of the course.
2. The Web Site shows the program.
3. The Clerk inserts the new program.
4. The Web Site confirms, and shows the new program.
Extensions:
1a. The Teacher asks to see the prerequisite of the course.
1a.1. The Web Site shows the prerequisite.
1a.2. The Clerk inserts the new prerequisite.
1a.3. The Web Site confirms, and shows the new prerequisite.

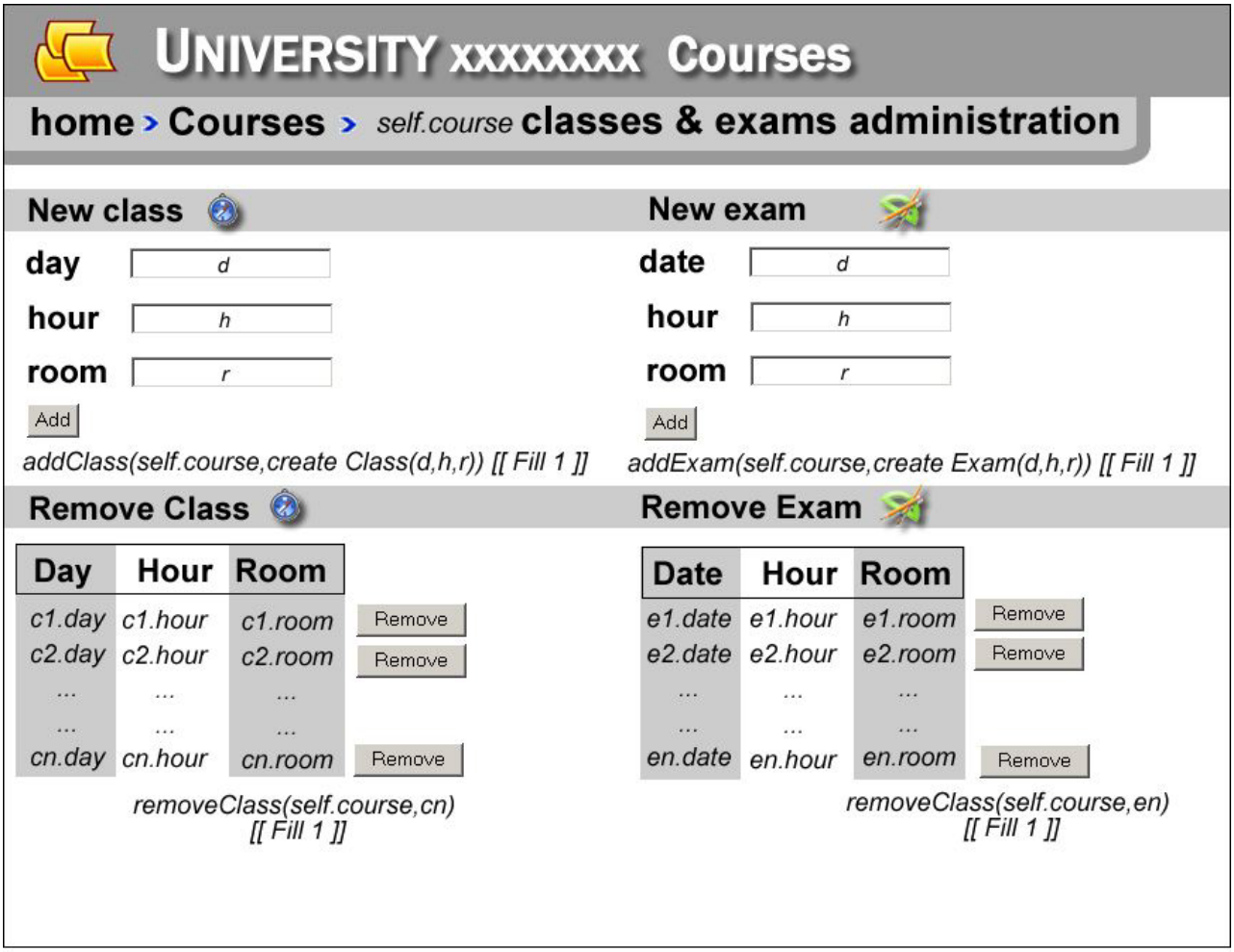
Update Case Handle classes
Intention in context: A clerk takes care of putting in the site the information about the classes of a course.
Content Manager: Clerk
Roles: Fill1
Entry point: Course
Main success scenario:
1. The Clerk asks to see all the classes of a course.
2. The Web Site shows all the classes.
3. The Clerk selects one of that classes.
4. The Web Site shows the info for such class (weekday, hour, room).
5. The Clerk asks to delete it.
6. The Web Site asks for a confirmation.
7. The Clerk confirms.
8. The Web Site shows the list of all classes.

Extensions:
6a. The Clerk does not confirm.
The case continues at step 8.
3a. The clerk asks to add a class giving the relative info (weekday, hour and room).
3a.1. The Web Site confirms.
The case continues at step 8.

Update Case Handle courses
Intention in context: The Dean takes care of adding and deleting courses depending on the university decisions.
Content Manager: Dean
Roles: Fill2
Entry point: Home
Main success scenario:
1. The Dean asks to see the list of all courses.
2. The Web Site shows the names of all courses.
3. The Clerk selects one of that courses.
4. The Web Site shows the basic info for such course.
5. The Clerk asks to delete it.
6. The Web Site asks for a confirmation.
7. The Clerk confirms.
8. The Web Site shows the names of all courses.

Extensions:
6a. The Clerk does not confirm.
The case continues at step 8.
3a. The clerk asks to add a course giving the relative basic info.
3a.1. The Web Site confirms.
The case continues at step 8.

B COURSES: Interface View



self.cls = c1, c2, ..., cn sorted by day attribute
self.exs = e1, e2, ..., en sorted by date attribute

Fig. 9. ActAdmin: Interface View

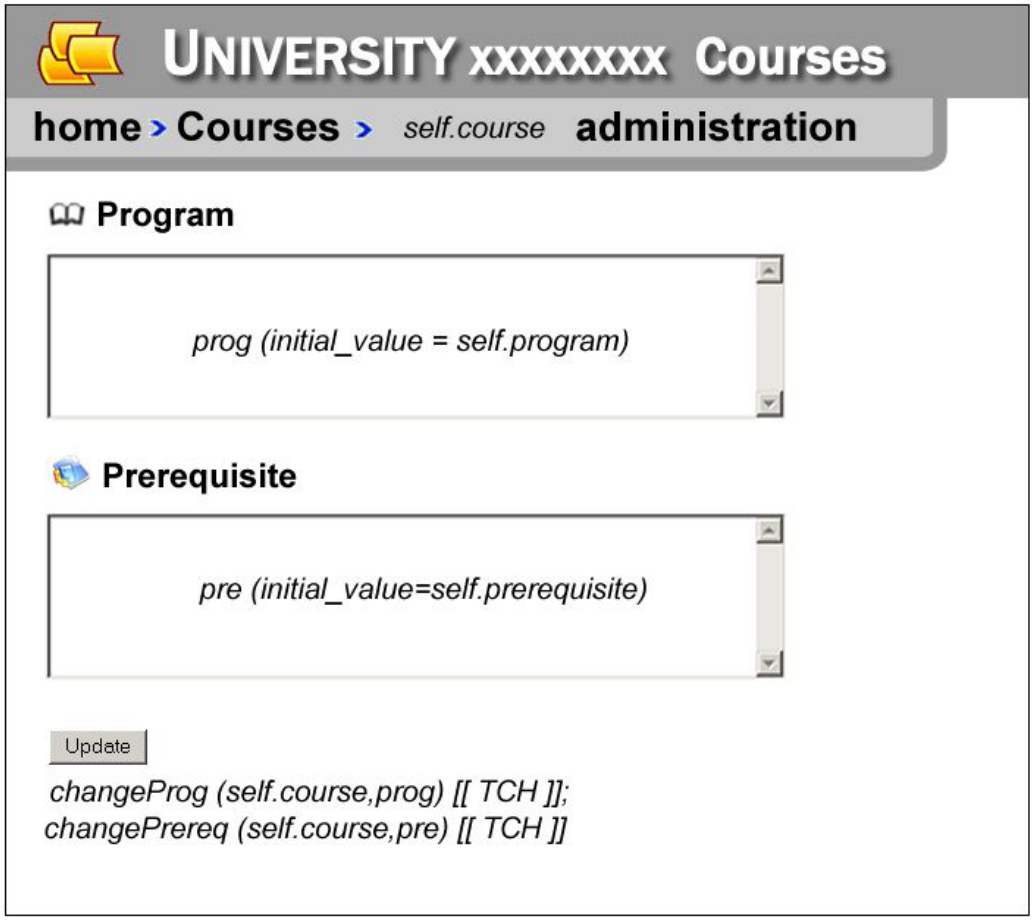


Fig. 10. CourseAdmin: Interface View

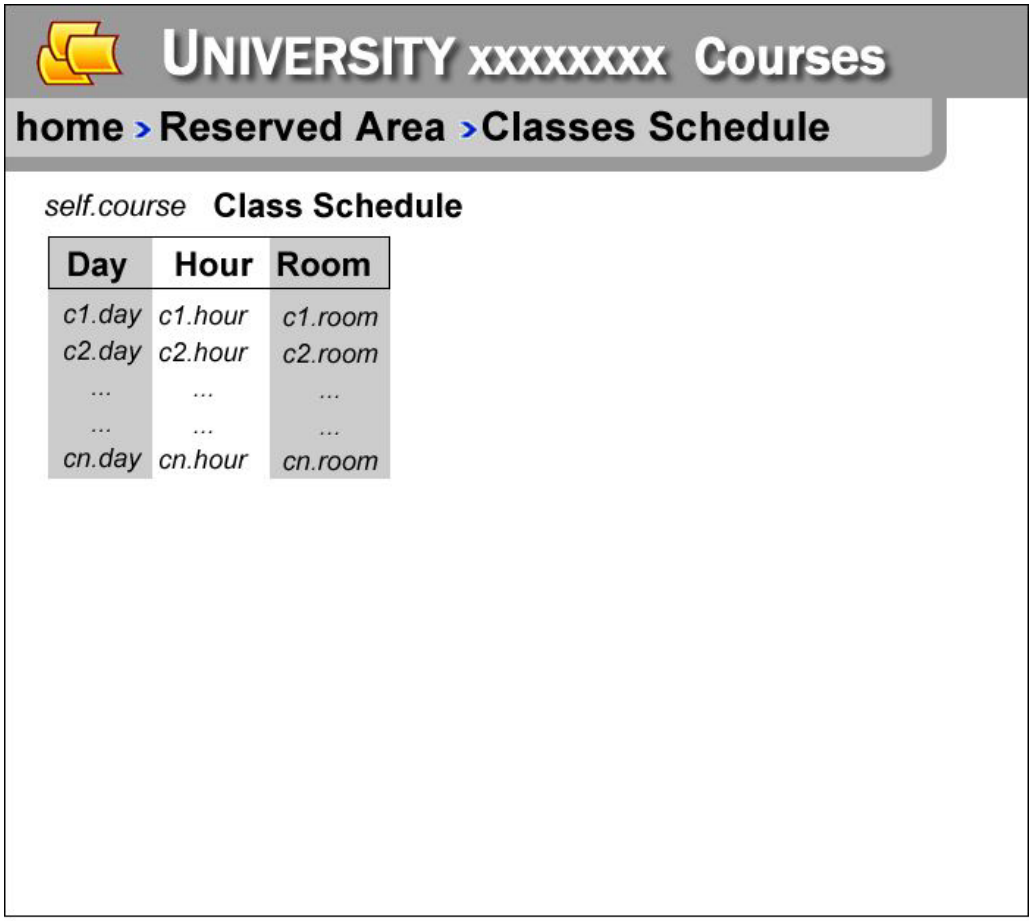


Fig. 11. ClassSchedule: Interface View

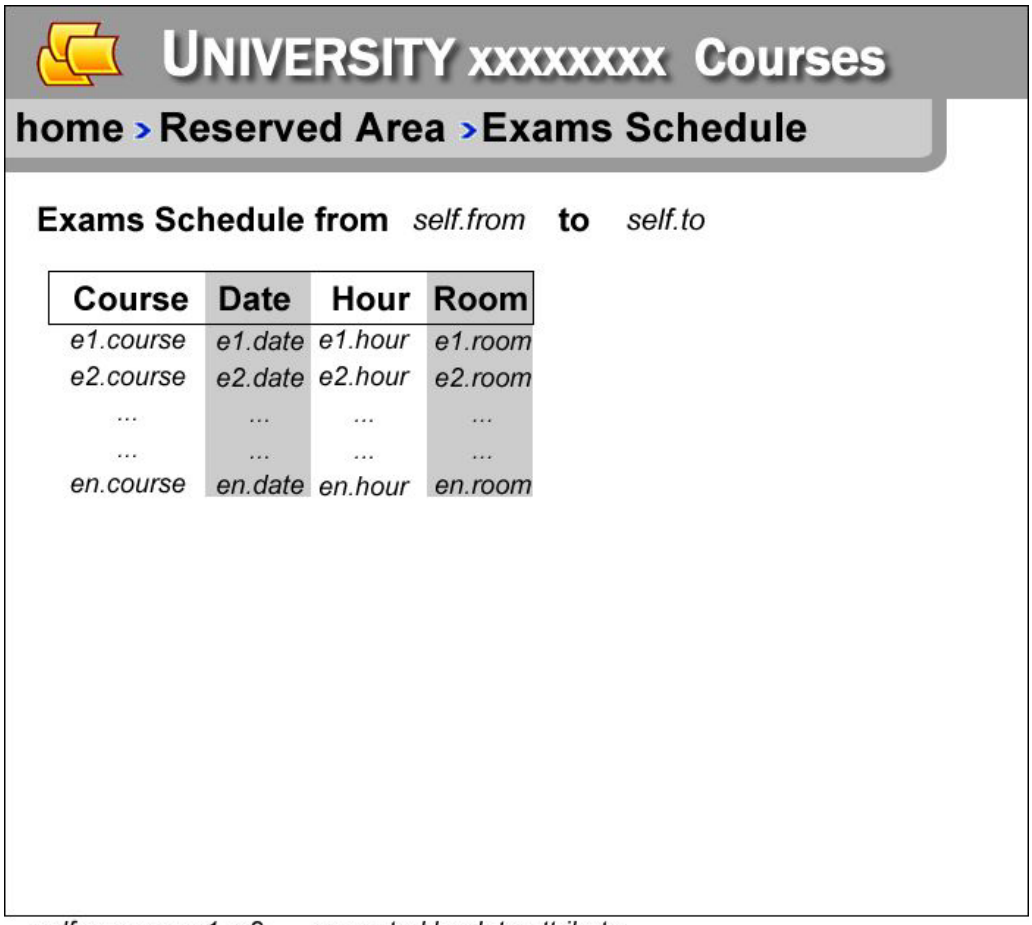
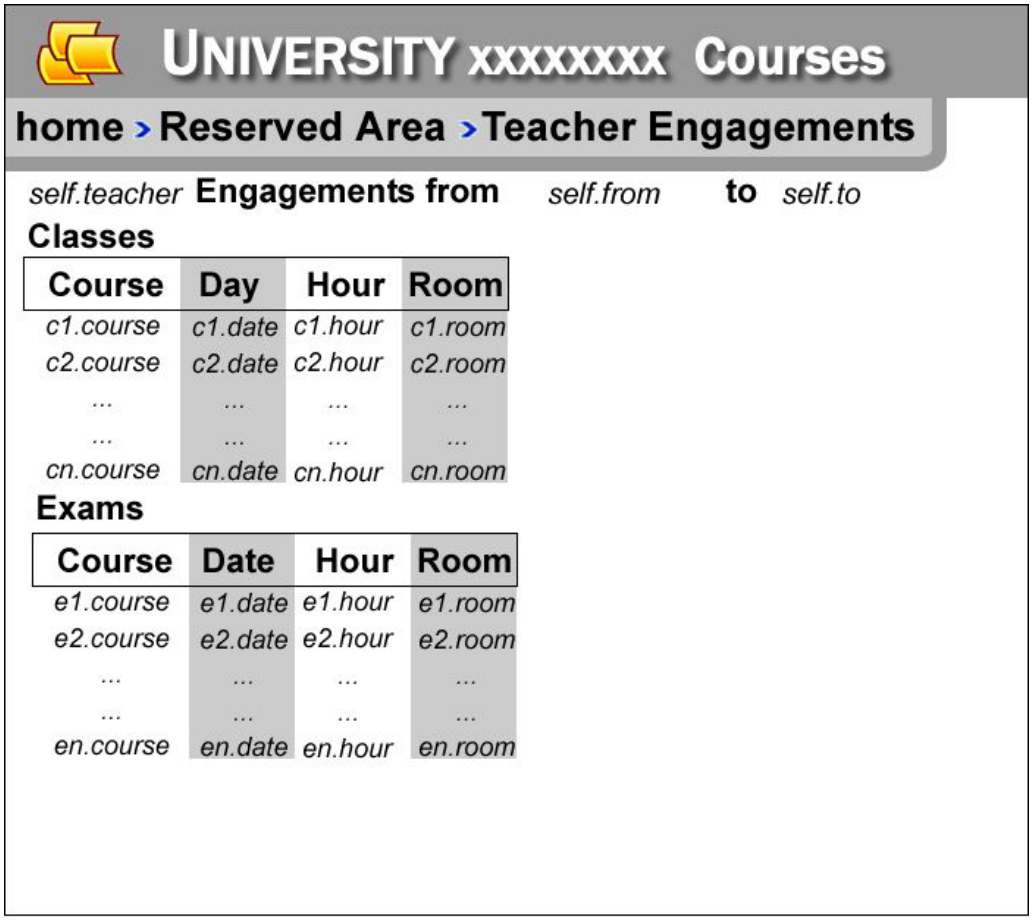


Fig. 12. ExamSchedule: Interface View



self.engagements->select(eng | eng.isTypeOf(Class) = c1, c2, ..., cn sorted by day

self.engagements->select(eng | eng.isTypeOf(Exam) = e1, e2, ..., en sorted by date

Fig. 13. TeacherEngagements: Interface View

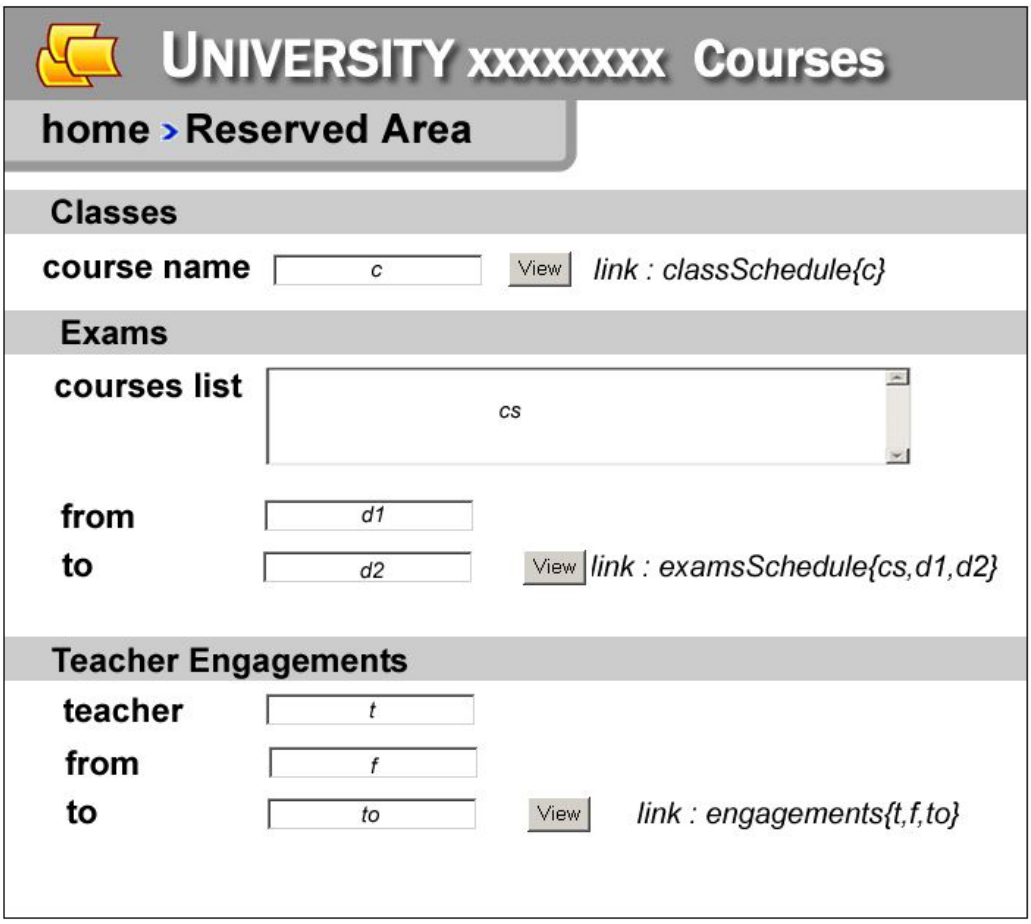
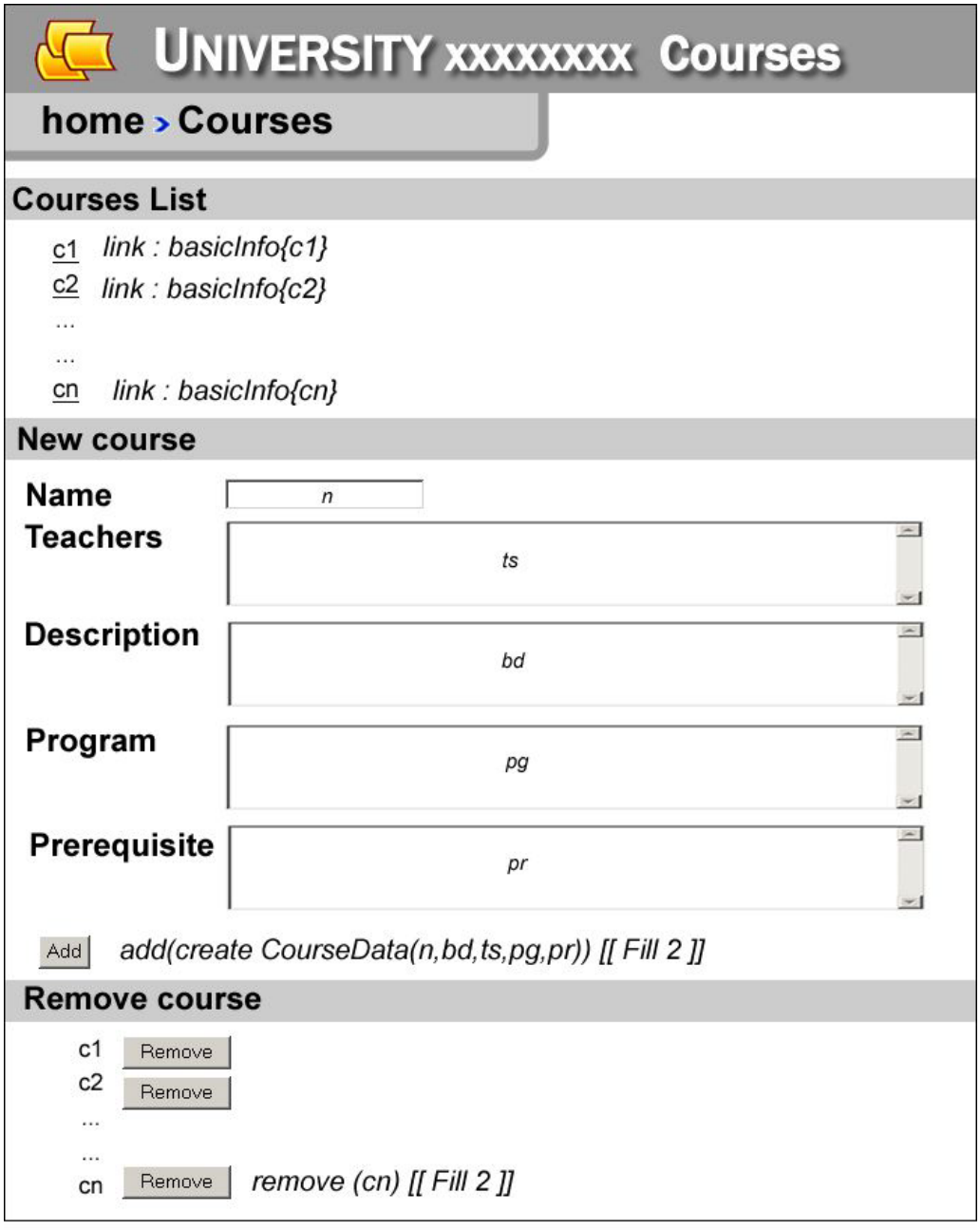


Fig. 14. ReservedArea: Interface View

Fig. 15. Course: Interface View



self.names = cn1,cn2, ..., cnn

Fig. 16. Courses: Interface View