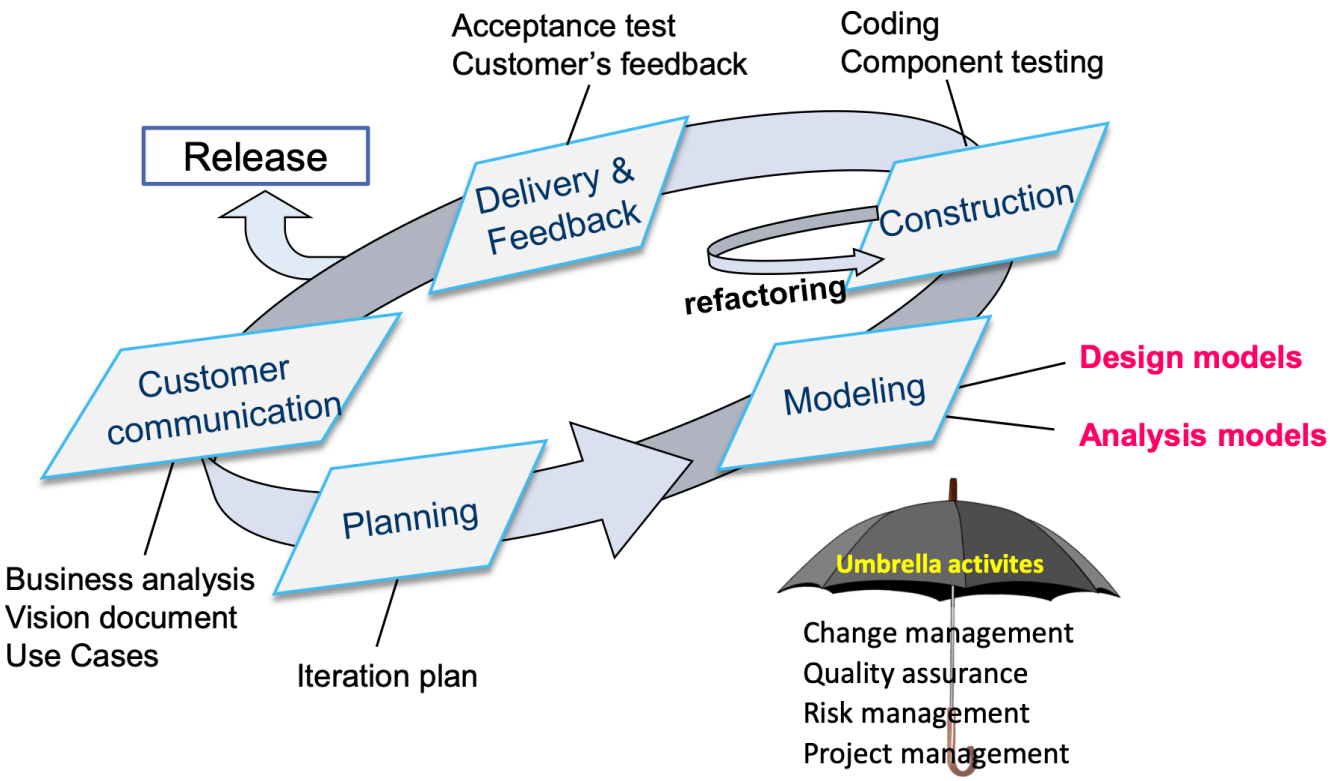
5 FIB - Aplicacions i Serveis Web

[transpes] Unit 4. Design of Web Applications (1/2)

Web Application Engineering

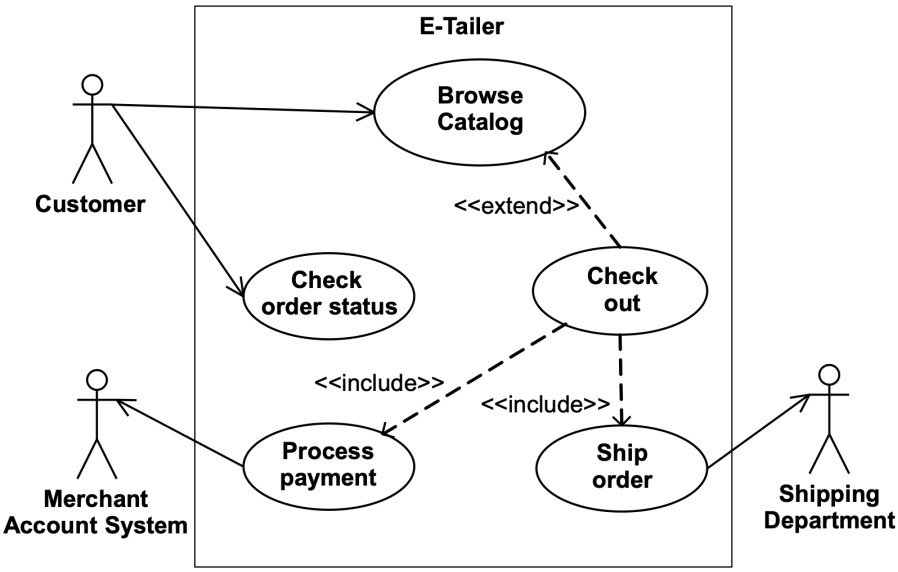


Web Application Analysis

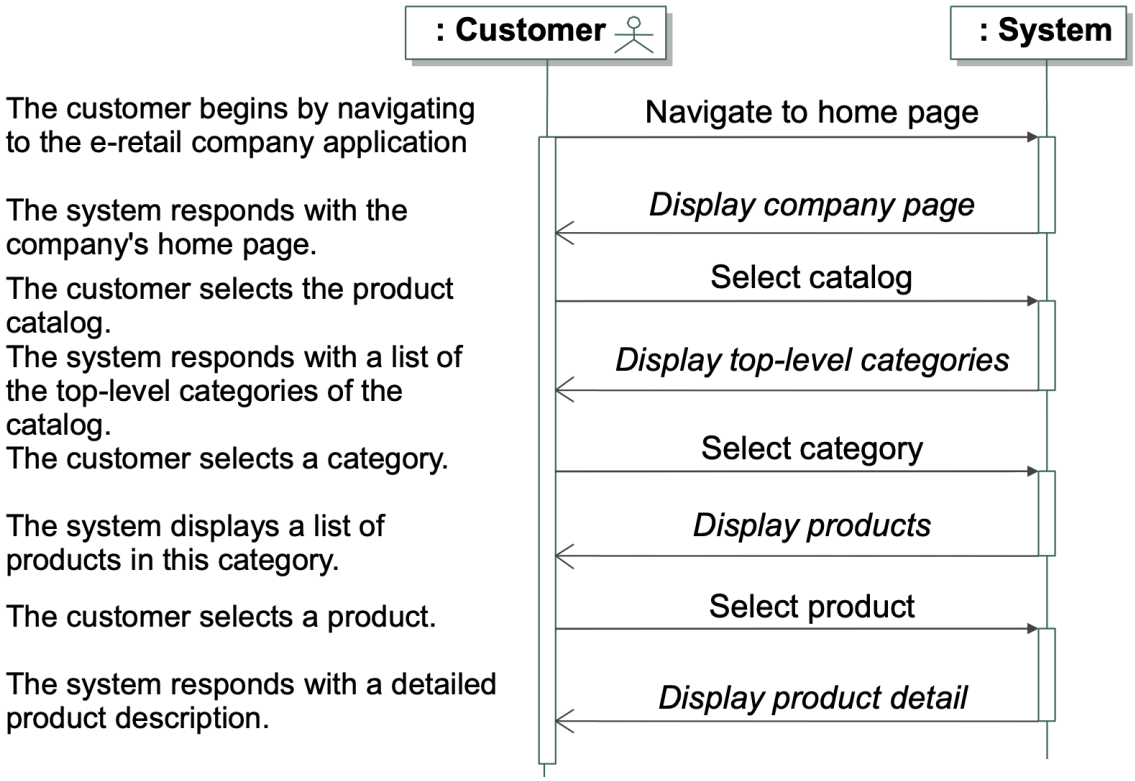
Web Application Analysis (similar to Software Application Analysis) tasks include:

- `[Class Diagram]` Represent WebApp content
- `[Class Diagram]` Identify content relationships
- `[Use Cases]` Refine and extend usage scenarios
- `[Use Cases]` Review usage scenarios
- `[Sequence Diagrams for Use Cases]` Identify system functions
- Define constraints and non-functional requirements

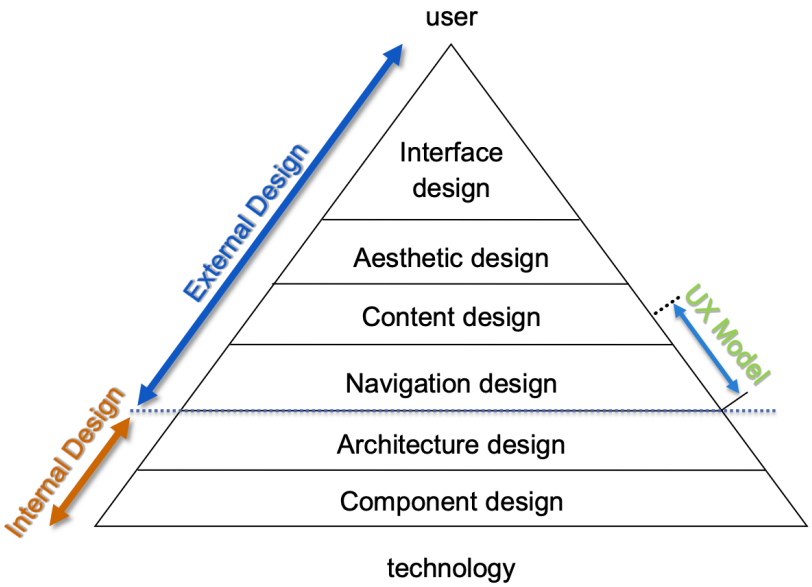
Use Case Diagram Example



System Sequence Diagram Example



Web Application Design



- Interface design:
  - Describes the structure and organization of the WebApp pages
  - Layout, menus, tabs, links, content, context information, search, etc.
- Aesthetic design (Graphic design):
  - Look and feel of the WebApp, colors, text size, font and placement, the use of graphics, etc…
- Content design:
  - Content structure and organization in pages ▶ Navigation design:
  - Definition of the navigational flows among pages that implement the different use cases.
- Architecture design:
  - Definition of the overall structure for the WebApp, components and interactions between them.
- Component design:
  - Develops the detailed processing logic required to implement functional components that support a complete set of WebApp functions.

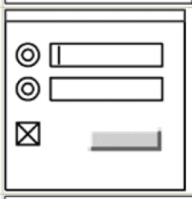
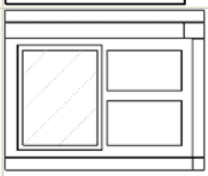
External Design

The UX Model

- [Conallen]
  - Corresponds to the Content design (partially) and Navigation design layers of the “pyramid”
  - L’UX Model describes how the (dynamic) content will be structured and organized in different screens and how the user will navigate among those screens to execute the WebApp use cases
  - Artifacts of the UX Model:
    - **Screens:**
      - something that is presented to the user, which contains the user interface infrastructure, such as menus and controls, as well as business-relevant content
    - **Storyboard sequences:**
      - describes a typical use of the system through the combination of a set of screens
    - **Navigational paths and maps:**
      - a road map of the application’s screens
- Screen Description

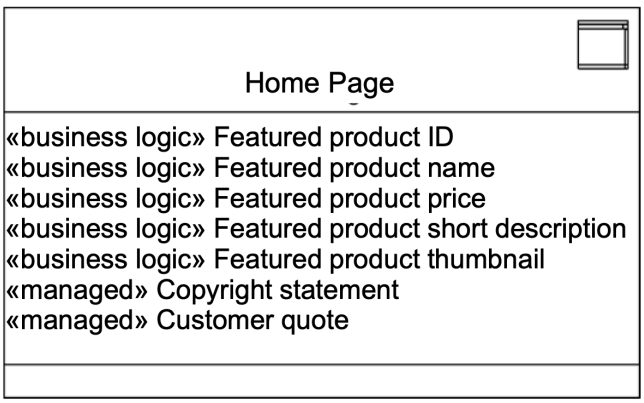
- A screen's properties and its behaviour with the user define the screen. These include:
  - Name and description
  - Structure: screen prototype
  - Content:
    - Static content (constant for users): field names, titles, text,...
    - Dynamic content
      - Business logic content
      - Managed content: Banner ads, help and informational messages, press releases, company and application FAQs, white paper
  - Input fields and controls that accept user input
  - Description of user interactions with the screen

- UX Model Stereotypes

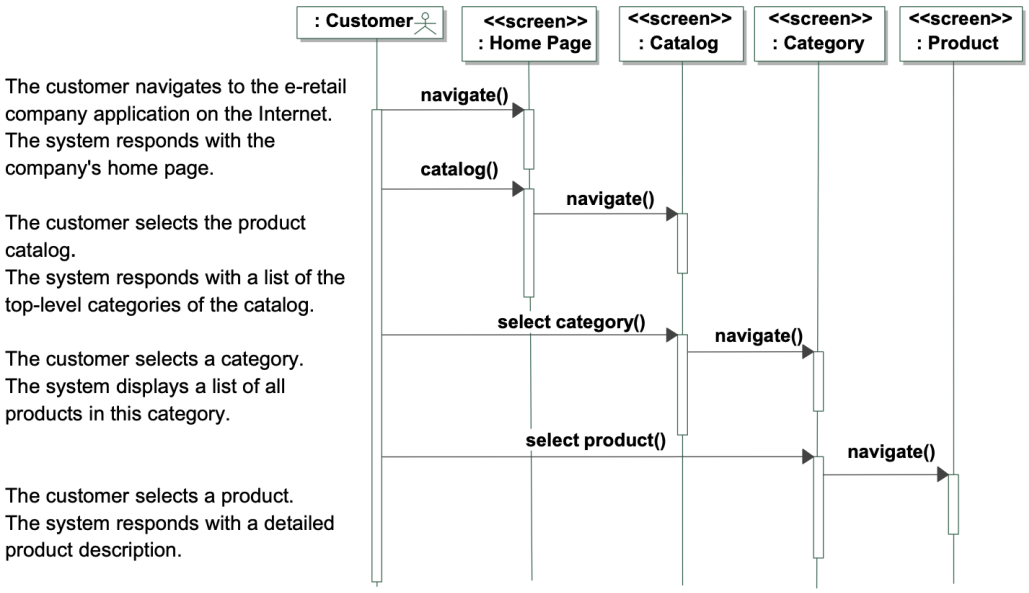
Stereotype	Icon	Decoration Icon
«screen»		Home Page Product special Product price Visitor count
«input form»		Payment Card holder : String Card number : String Expiration : String
«screen compartment»		Footer Copyright statement

UX Modeling by Example

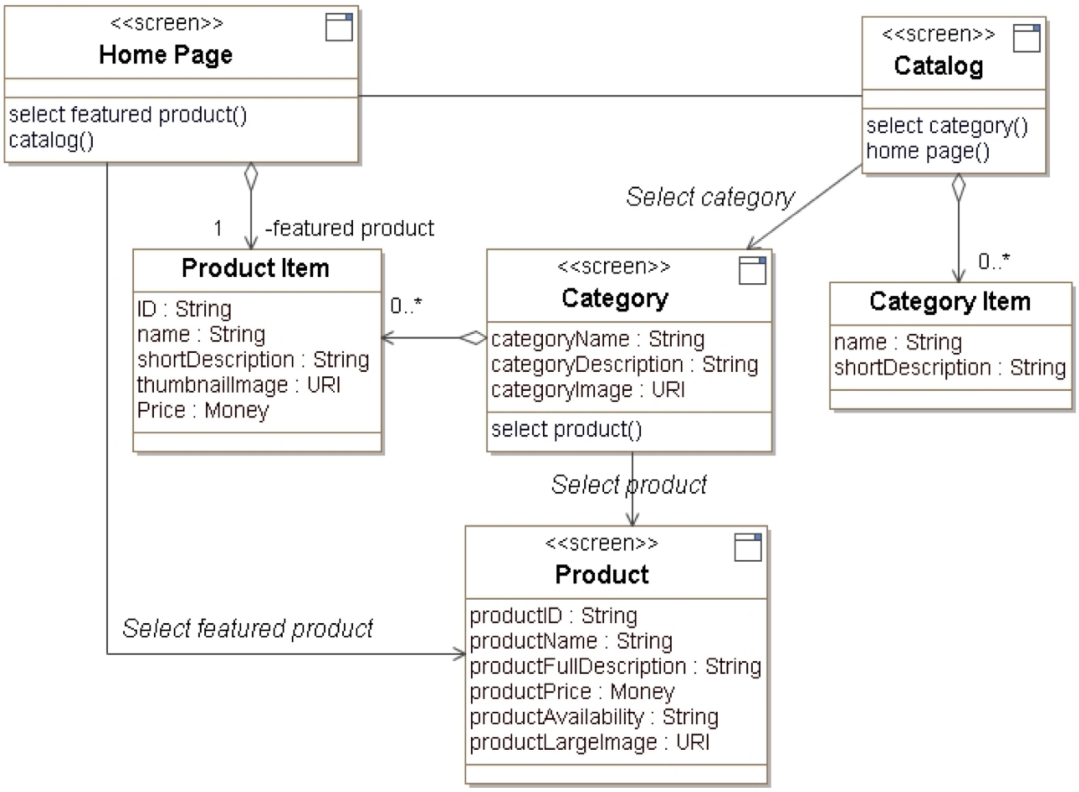
- Home Page Screen



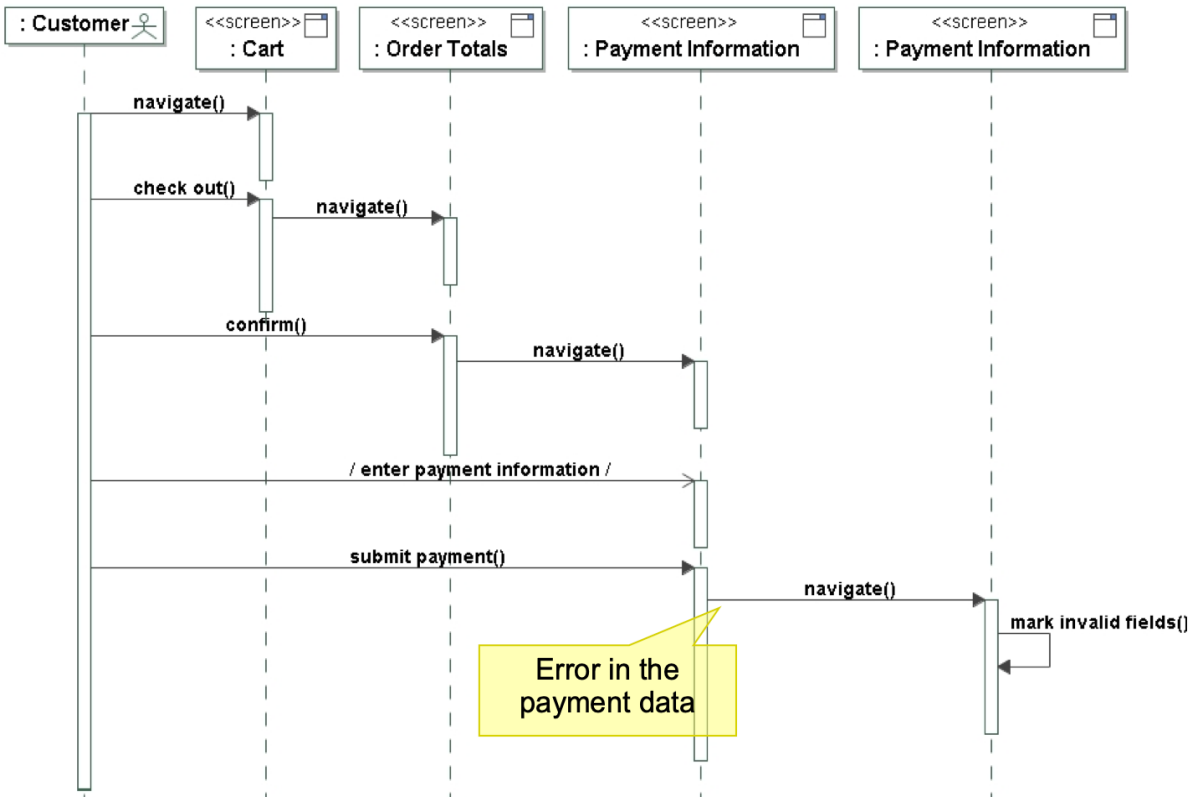
- Browse Catalog Storyboard



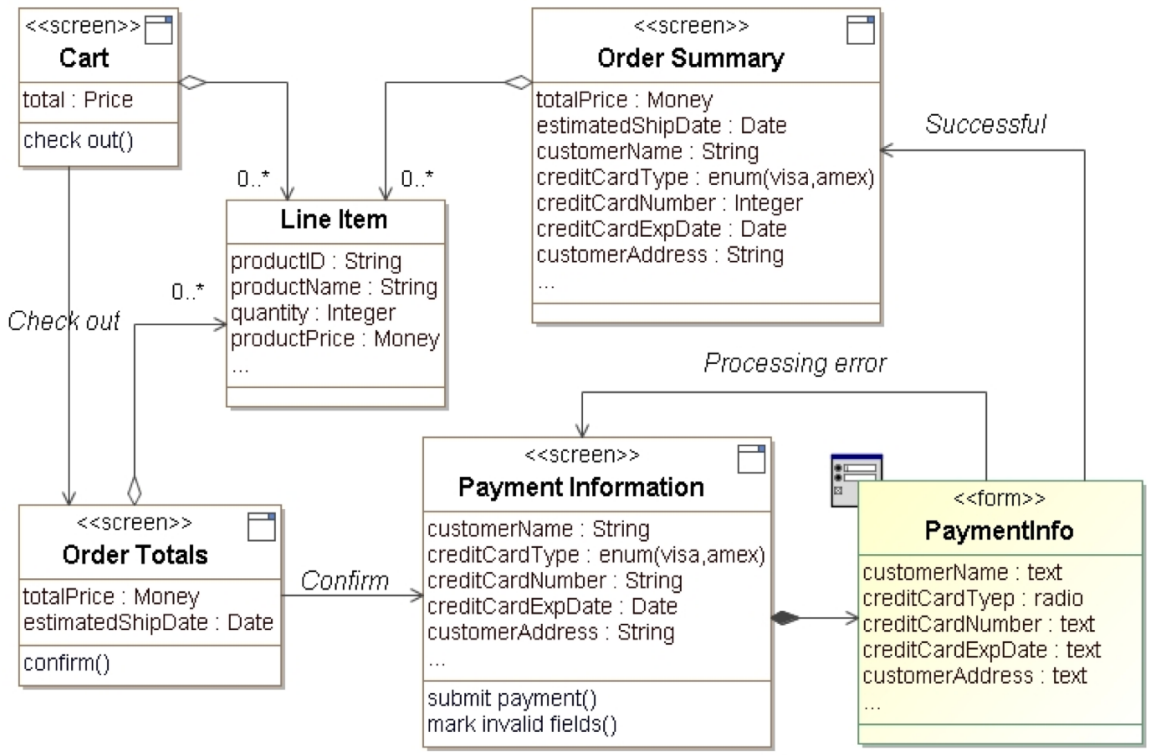
- Screens and Navigational paths



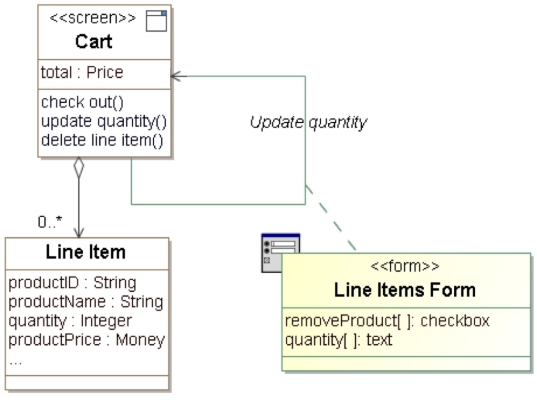
- Checkout Storyboard



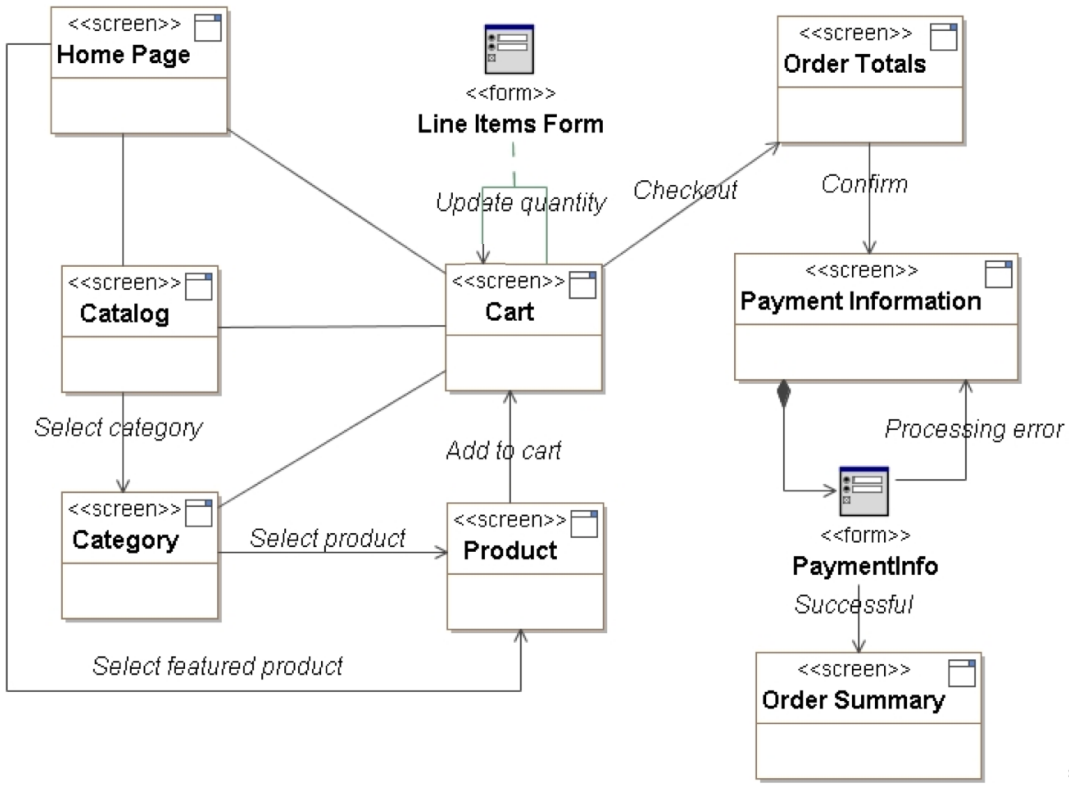
- Screens and Navigational paths



- Cart Update Screens and Navigational paths



- Navigational Map




```
Note: EXERCICI - Exercici 4.1.sol.pdf
```

Internal Design

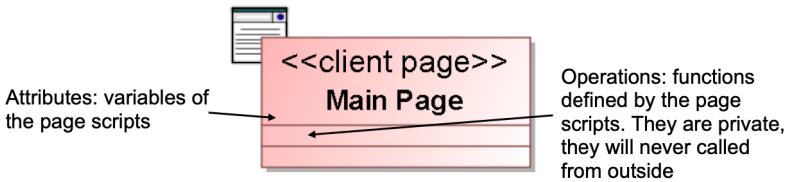
The Web Application Extension for UML (WAE2)

- Web pages should be modeled as first-class elements in the internal design model and represented alongside the classes and components that make up the rest of the Web presentation layer.
- However, the default building blocks of UML are not sufficient to express the necessary subtleties of Web pages as objects:
  - Web pages may have scripts to be executed on the server, which interact with server-side resources before being sent to the browser as a completed Web page.
  - Web pages may have scripts that execute on the browser
  - When processed by the server, the Web page does one thing; when processed by the browser, it does a completely different thing.
- The Web Application Extension (WAE2) to UML enables us to represent Web pages and other architecturally significant elements in the internal design model of the Web presentation layer

- UML Mechanisms To Extend UML
  - In general, any extension to UML is expressed in terms of stereotypes, tagged values, and/or constraints.
  - Combined, these mechanisms extend the notation of UML, enabling creating new types of building blocks to be used in the model.
  - Stereotype: is an extension to the vocabulary of the language that allows to attach a new semantic meaning to a UML model element (a class, an association, etc).
  - Tagged value: is the definition of a new property that can be associated with a model element.
    - UML Classes, for instance, have names, visibility, persistence, and other properties associated with them.
  - Constraint: specifies the conditions under which the model can be considered well formed.

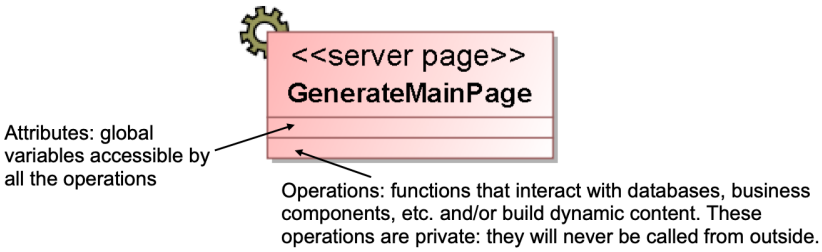
- WAE2 Stereotypes
  - Main Class Stereotypes:
    - Server Page
    - Client Page
    - Form
  - Main Association Stereotypes:
    - Link
    - Build
    - Submit
    - Redirect

- WAE2 Client Page



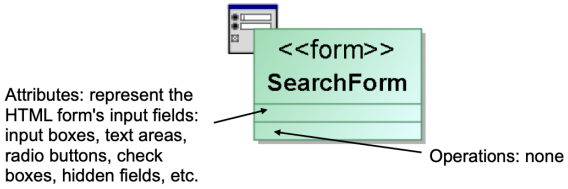
  - A client page instance is an HTML-formatted Web page with a mix of data, presentation, and even logic.
  - Constraints: none
  - Tagged values:
    - TitleTag, the title of the page as displayed by the browser.
    - BaseTag, the base URL for dereferencing relative URLs.
    - BodyTag, the set of attributes for the `<body>` tag, which sets background and default text attributes.

- WAE2 Server Page



  - A server page represents a dynamic Web page that contains content assembled on the server each time it is requested. Later it can be implemented as a Servlet, JSP, ASP, or PHP page
  - Constraints: Server pages can have only “normal” relationships with objects on the server
  - Tagged values: none

- WAE2 Form



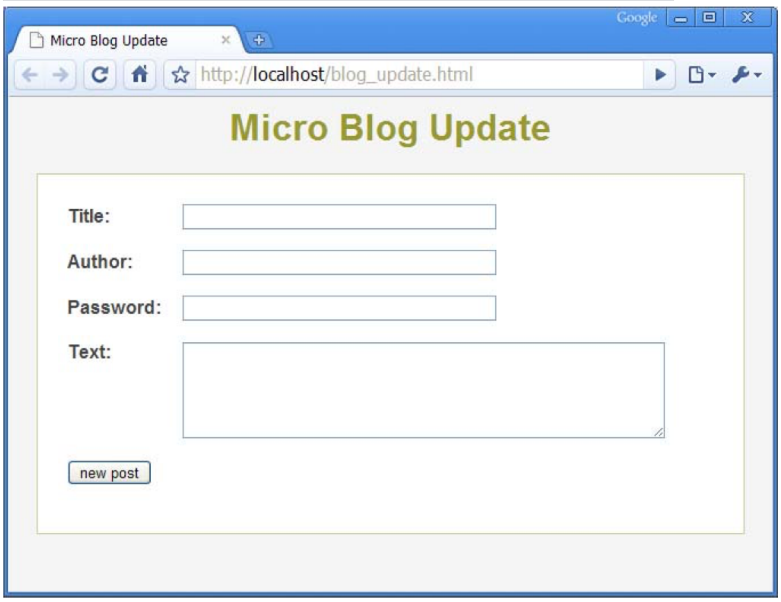
  - A form instance represents a HTML form in a client page.
  - Constraints: none.
  - Tagged values:
    - Either GET or POST: the method used to submit data to the action URL.

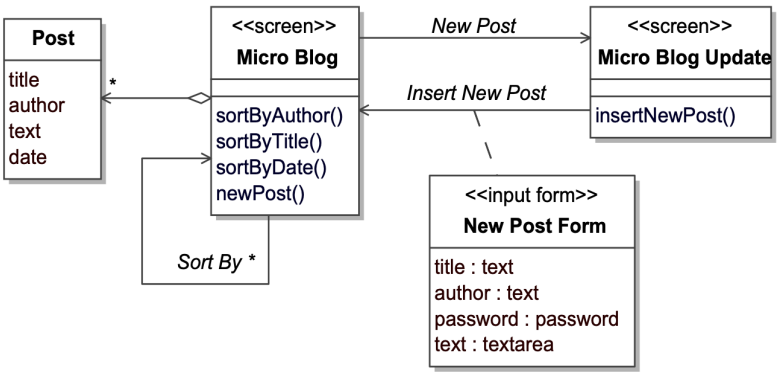
- WAE2 Association Stereotypes

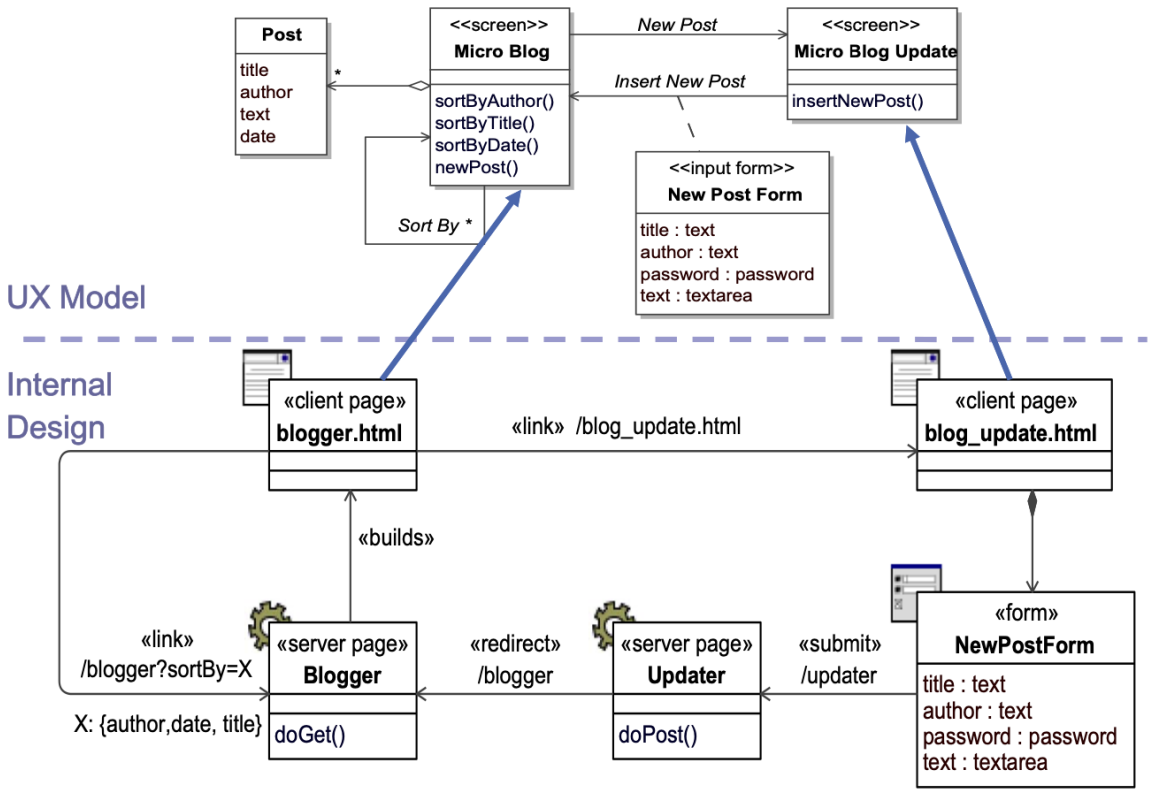
Association Stereotype	Source	Target	Description
<<link>>	<<Client Page>>	<<Client Page>> <<Server Page>	Abstraction of <a href= ...> Tagged value: parameters
<<build>>	<<Server Page>>	<<Client Page>>	Identifies the HTML output of a server page's execution.
<<submit>>	<<Form>>	<<Server Page>>	Form data submission
<<redirect>>	<<Client Page>> <<Server Page>>	<<Client Page>> <<Server Page>	Makes the browser request the target resource
<<forward>>	<<Server Page>>	<<Client Page>> <<Server Page>	Delegation of processing a client's request for a resource to another server-side page
<<include>>	<<Server Page>>	<<Client Page>> <<Server Page>	the included page gets processed, if dynamic, and its contents are incorporated in the container page.
<<object>>	<<Client Page>>	ActiveX, Applet Class	Abstraction of <object> o <applet>
<<script>>	<<Client Page>>	<<Script Library>	Script Library import

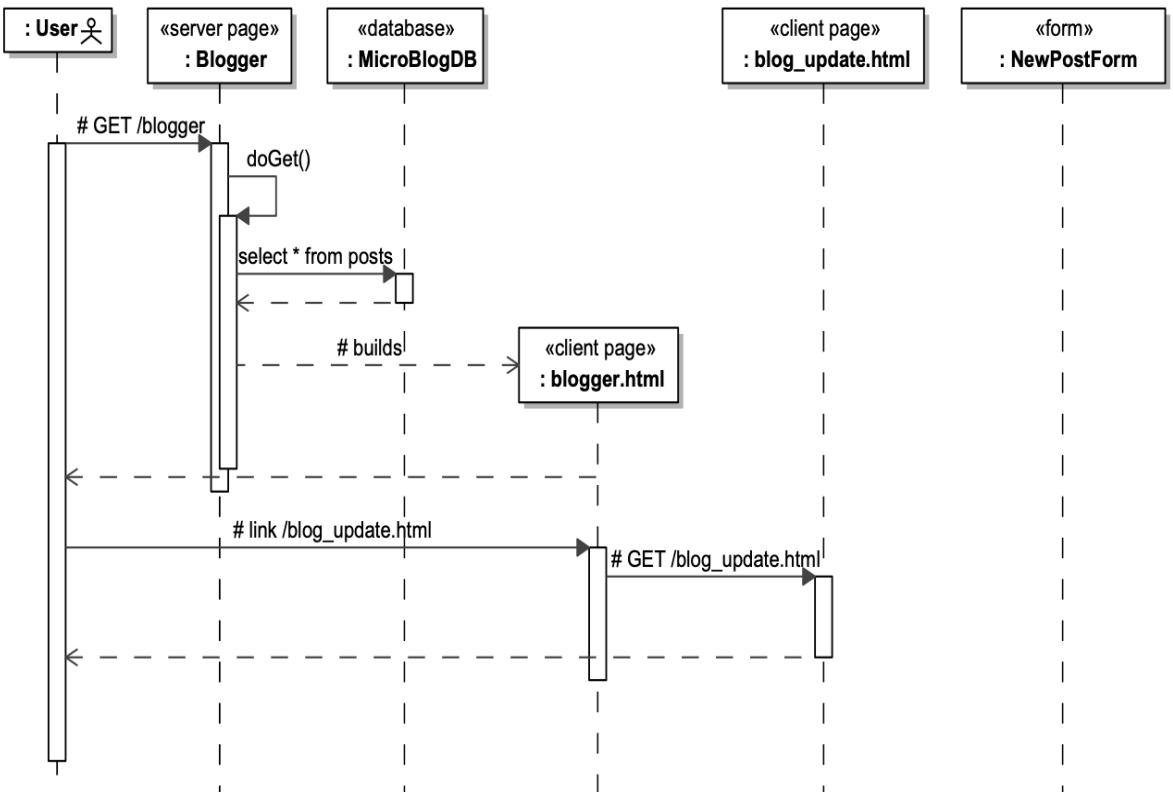
- Case Study: Micro Blog Example

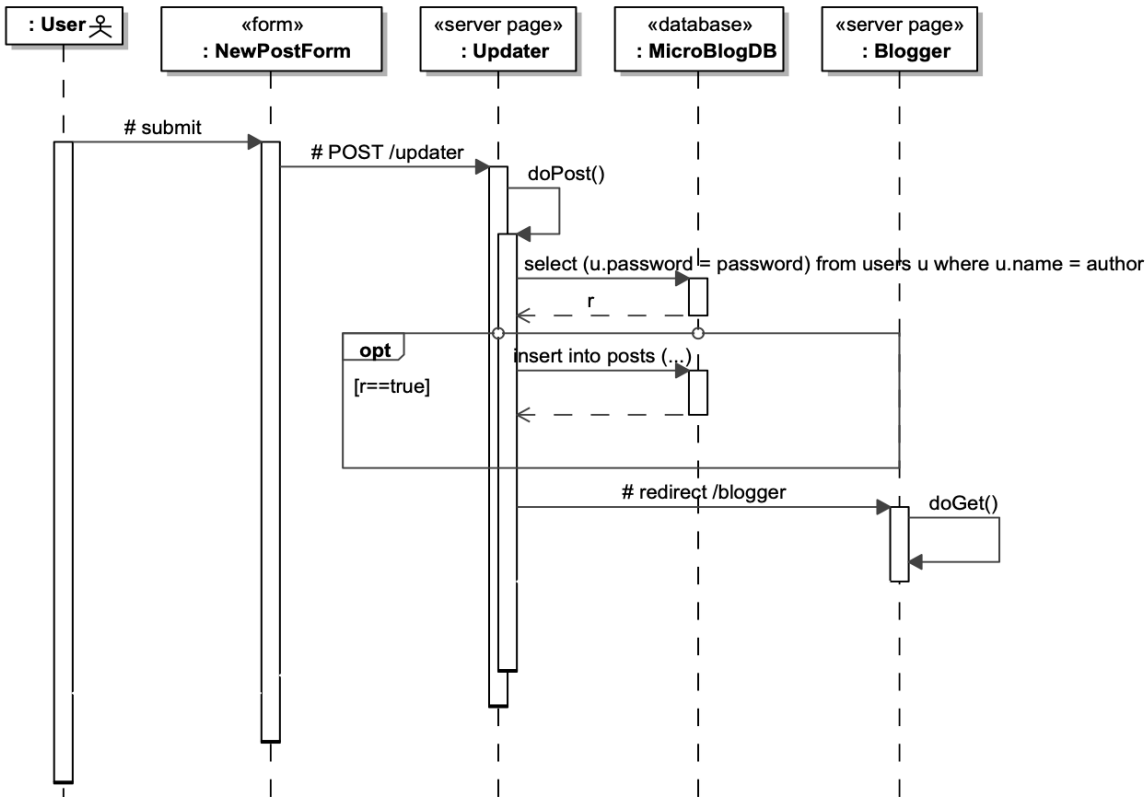

Micro Blog (beta)
http://localhost/blogger
Micro Blog (beta)
Sort posts by author, title or date. To add a new post, click here
Friday, March 27, 2009
Message Title #3
Nulla congue tellus sit amet felis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae. (posted by marianico @ 12:07:16 AM)
Friday, March 27, 2009
Message Title #2
Aliquam pulvinar nulla ac lectus. (posted by marianico @ 12:07:16 AM)
Thursday, March 26, 2007
Message Title #1


Micro Blog Update
http://localhost/blog_update.html
Micro Blog Update
Title:
Author:
Password:
Text:
new post


Post
title
author
text
date
*
<<screen>>
Micro Blog
sortByAuthor()
sortByTitle()
sortByDate()
newPost()
Sort By *
New Post
Insert New Post
<<screen>>
Micro Blog Update
insertNewPost()
<<input form>>
New Post Form
title : text
author : text
password : password
text : textarea

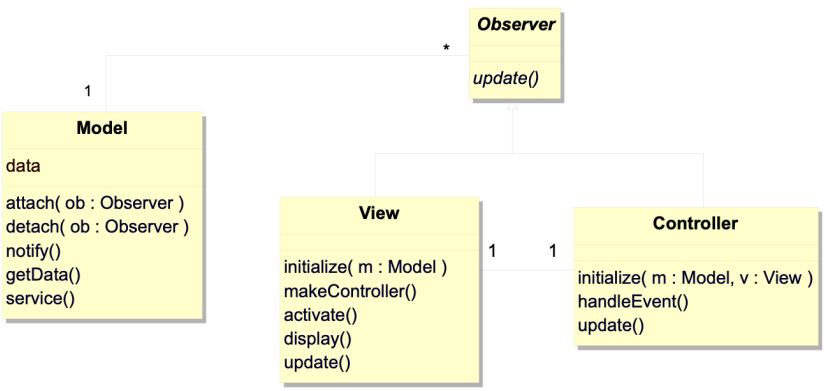

Post
title
author
text
date
*
<<screen>>
Micro Blog
sortByAuthor()
sortByTitle()
sortByDate()
newPost()
Sort By *
New Post
Insert New Post
<<screen>>
Micro Blog Update
insertNewPost()
<<input form>>
New Post Form
title : text
author : text
password : password
text : textarea
UX Model
Internal Design
«client page»
blogger.html
«link» /blog_update.html
«client page»
blog_update.html
«builds»
«link»
/blogger?sortBy=X
X: {author,date, title}
«server page»
Blogger
doGet()
«redirect»
/blogger
«server page»
Updater
doPost()
«submit»
/updater
«form»
NewPostForm
title : text
author : text
password : password
text : textarea


: User
«server page»
: Blogger
«database»
: MicroBlogDB
«client page»
: blog_update.html
«form»
: NewPostForm
# GET /blogger
doGet()
select * from posts
builds
«client page»
: blogger.html
link /blog_update.html
GET /blog_update.html



The Model-View-Controller (MVC) Architectural Pattern

- Divides an interactive application into three components/levels:
  - **Model**:
    - Contains the functional core of the application
    - Encapsulates the appropriate data, and exports procedures that perform application-specific processing
  - **View**:
    - Displays information to the user
    - Obtains the data from the model
    - Different views present the model's information in different ways
  - **Controller**:
    - Accepts user inputas events
    - Translates events to service requests for the model or display requests for the view
- The "Classical" MVC Pattern



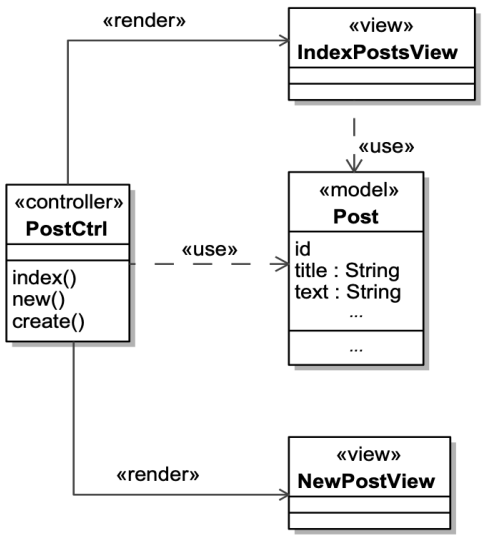
- MVC for Web Apps: The RoR way
  - Models
    - 1 model class for each type of entity manipulated by the app
    - Active Record pattern

- Controllers
  - Each controller class corresponds to 1 model class
  - Each controller action is handled by a particular method within that controller.
    - RESTful routes
- Views
  - Many view classes for each model class
  - “Rendered” by controllers

- RESTful routes
  - A route maps an incoming HTTP request to the appropriate controller and method.
  - REST (REpresentational State Transfer) is an architectural style for designing networked applications
  - RESTful routes:
    - Each entity manipulated by a Web app (model) is a resource
    - Any HTTP request should contain all the information necessary to identify both a particular resource and the action to be performed on it.
  - Example:

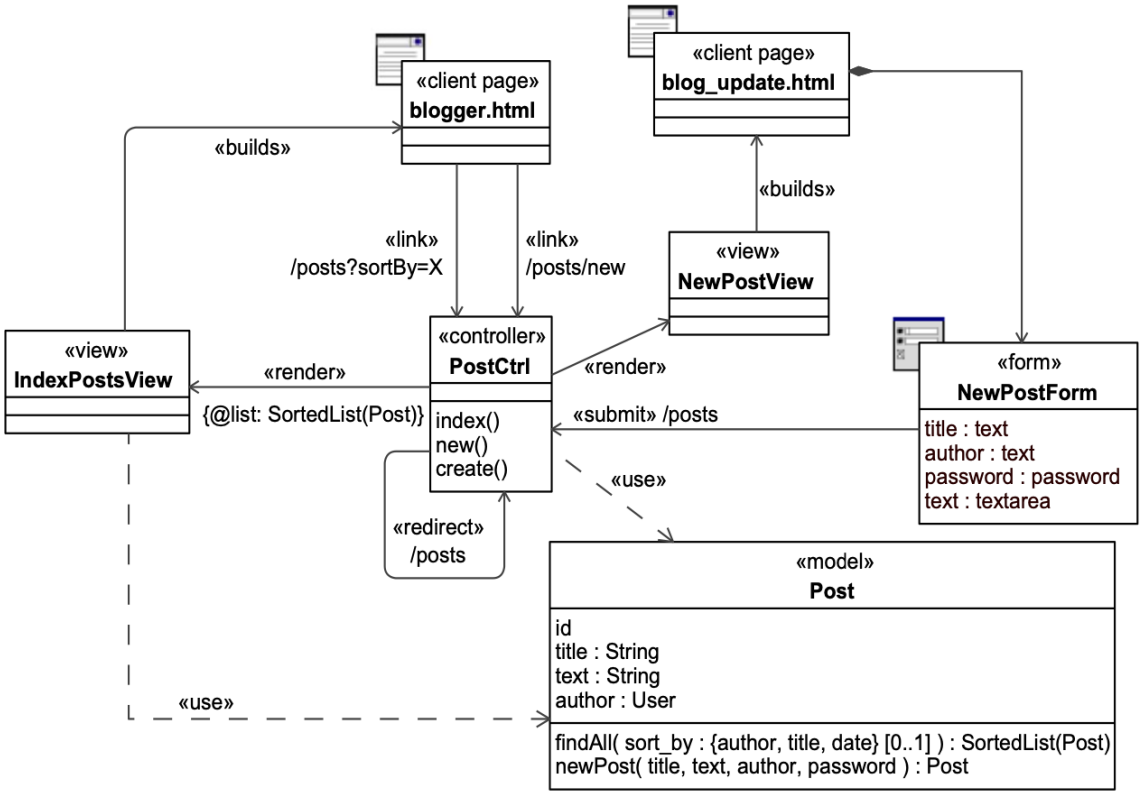
Operation on resource	Method & URI		Controller method	Rendered view
List (index) of posts	GET	/posts	index	IndexPostsView
Show just one post instance	GET	/posts/:id	show	ShowPostView
Display a fill-in form for a new post	GET	/posts/new	new	NewPostView
Create a new post from filled-in form	POST	/posts	create	-
Display from to edit existing post	GET	/posts/:id/edit	edit	EditPostView
Update post from fill-in form	PUT	/posts/:id	update	-
Destroy existing post	DELETE	/posts/:id	destroy	-

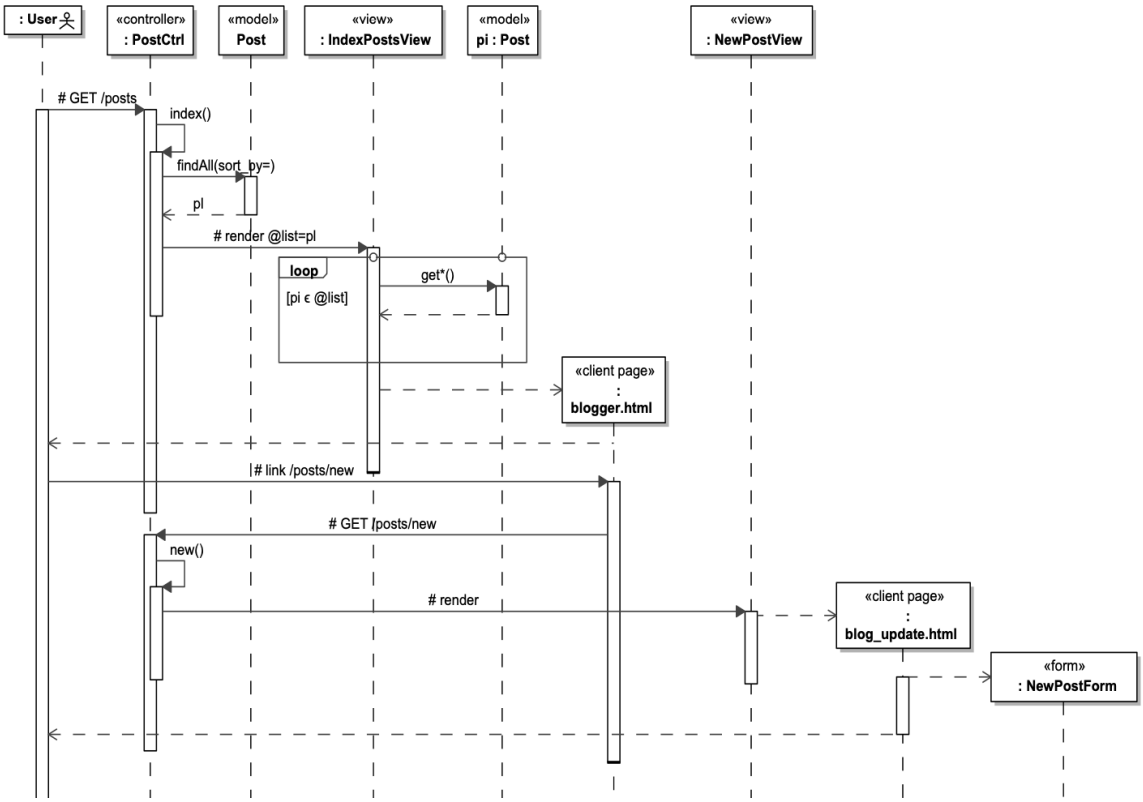
- WAE2 + MVC: New Stereotypes

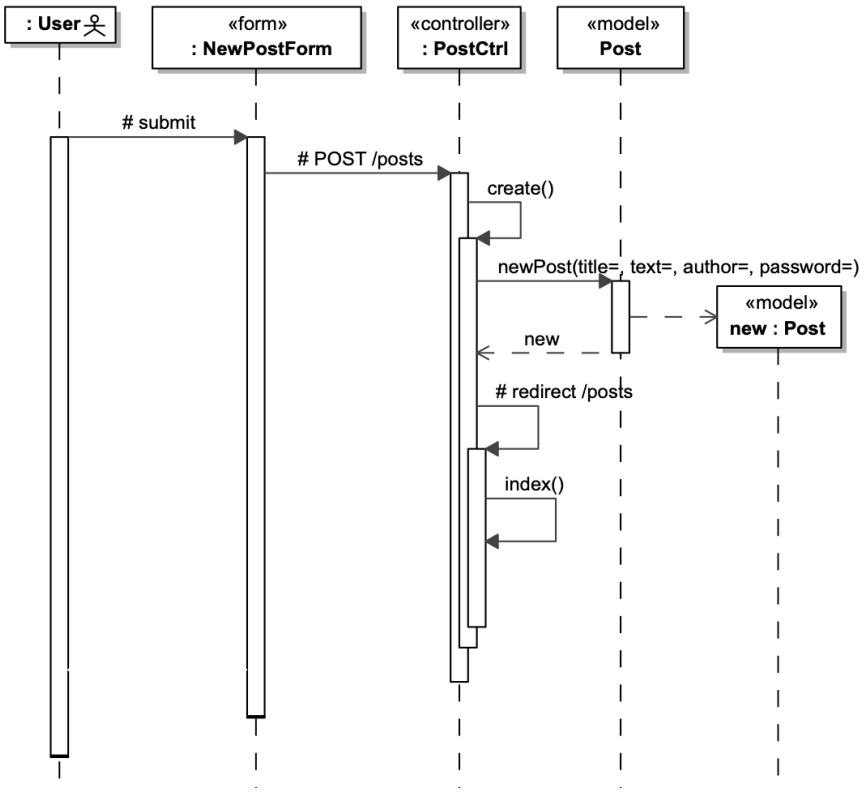


<<controller>> and <<view>> replace <<Server Page>>

- Micro Blog Example Revisited


«client page»
blogger.html
«client page»
blog_update.html
«builds»
«builds»
«link»
/posts?sortBy=X
«link»
/posts/new
«view»
NewPostView
«controller»
PostCtrl
«view»
IndexPostsView
«render»
{@list: SortedList(Post)}
«render»
index()
new()
create()
«submit» /posts
«form»
NewPostForm
title : text
author : text
password : password
text : textarea
«use»
«redirect»
/posts
«model»
Post
id
title : String
text : String
author : User
findAll(sort_by : {author, title, date} [0..1]) : SortedList(Post)
newPost(title, text, author, password) : Post
«use»


: User
«controller»
: PostCtrl
«model»
Post
«view»
: IndexPostsView
«model»
pi : Post
«view»
: NewPostView
# GET /posts
index()
findAll(sort_by=)
pl
render @list=pl
loop
[pi ∈ @list]
get*()
«client page»
:
blogger.html
# link /posts/new
# GET /posts/new
new()
# render
«client page»
:
blog_update.html
«form»
: NewPostForm


: User
«form»
: NewPostForm
«controller»
: PostCtrl
«model»
Post
# submit
# POST /posts
create()
newPost(title=, text=, author=, password=)
«model»
new : Post
new
# redirect /posts
index()