

Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak
WebApp Engineering

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING

Agenda

- Arsitektur Web App
- Requirement dan Use Case
- User Experience
- Analisis
- Perancangan

Arsitektur Web.App



Defining The Architecture

- Presentation tier pattern:
 - Thin Web Client
 - Thick Web Client
 - Web Delivery

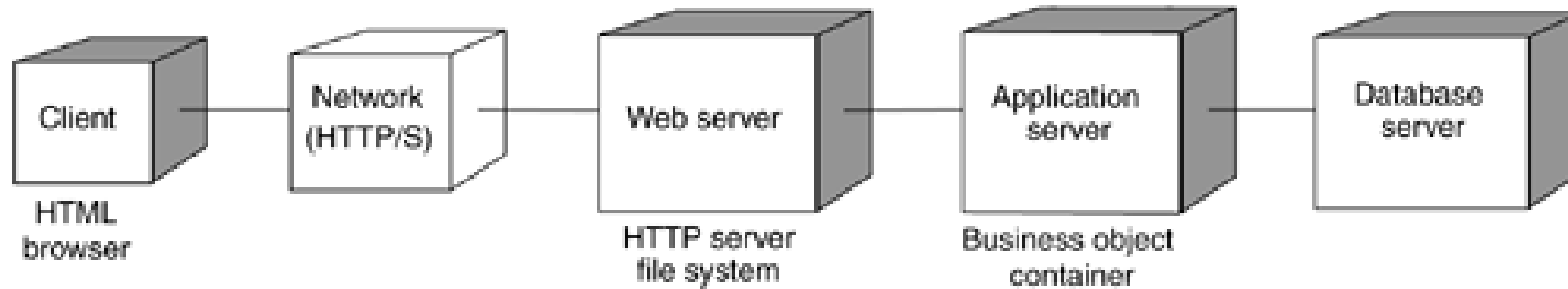


Patterns for The Presentation Tier

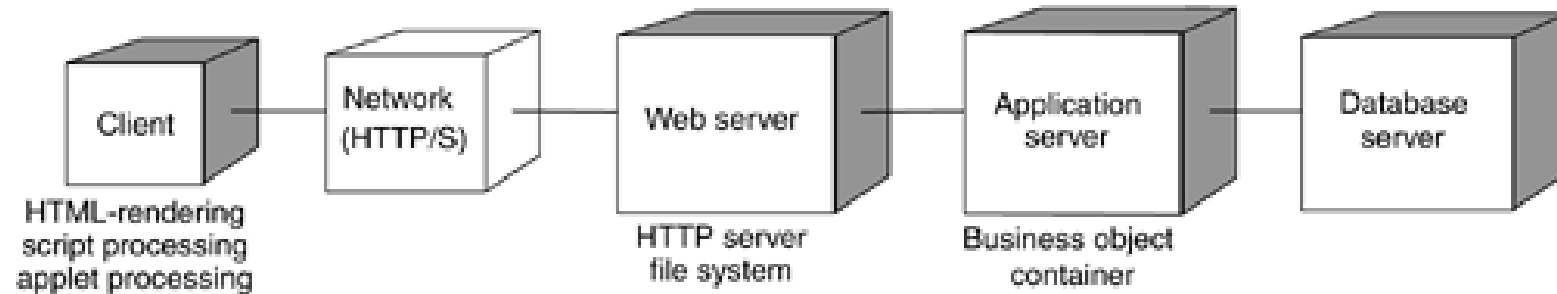
- **The thin Web client** is used mostly for Internet-based applications, in which there is little control of the client's configuration.
 - The client requires only a standard forms-capable Web browser.
 - All the business logic is executed on the server.
- **The thick Web client** pattern is used when an architecturally significant amount of business logic is executed on the client machine.
 - Typically, the client uses dynamic HTML, Java applets, or ActiveX controls to execute business logic.
- **The Web delivery** pattern is used when the Web browser acts principally as a delivery and container device for a distributed object system.
 - In addition to HTTP for client and server communication, other protocols, such as IIOP and DCOM, may be used to support a distributed object system.



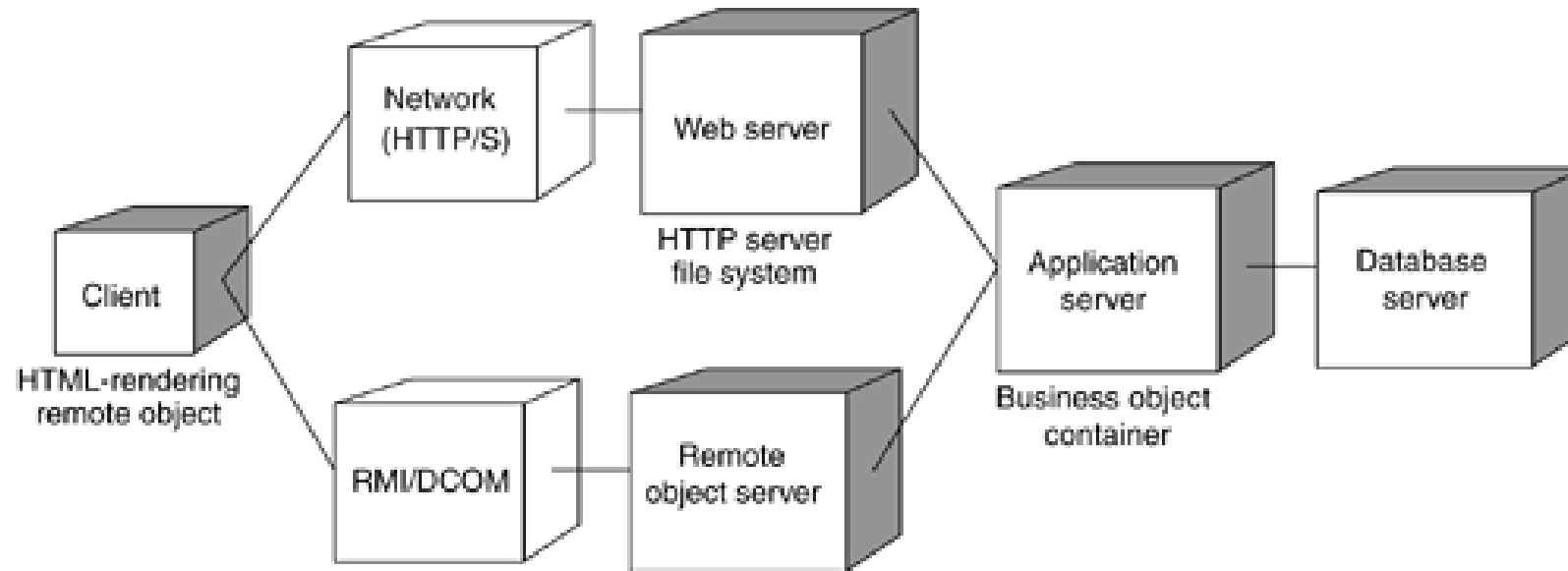
Deployment of simple thin Web client application



Deployment of simple thick Web client application



Deployment of simple Web delivery application



Requirement dan Use Case



Requirement - Example

3. Performance Requirements

This section describes the system's performance requirements.

These requirements usually relate to the execution speed and capacity of each component of the system.

3.1 Web server performance

The Web server performance section describes the expected performance of the Web server and network of the system.

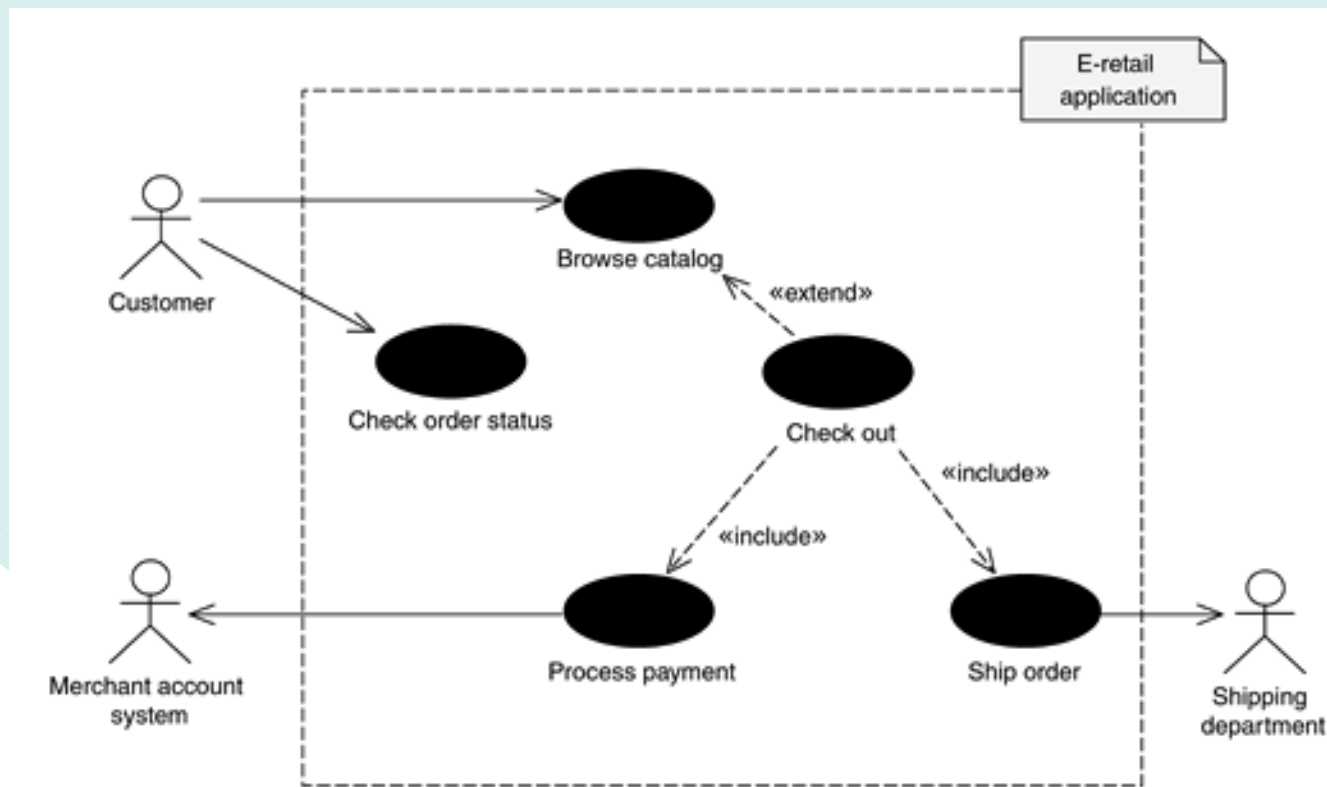
3.1.1 Each Web server in the system shall be able to handle at least 150 simultaneous user sessions.

3.1.2 The system shall require no more than 3 seconds to retrieve and to respond to a client's request for a static Web page.

3.1.3 The system shall require no more than 8 seconds to respond to a dynamic page.

→ The requirements fragment contains some pretty [specific knowledge about the architecture](#)

Use Case Diagram - Example



User Experiences (UX)



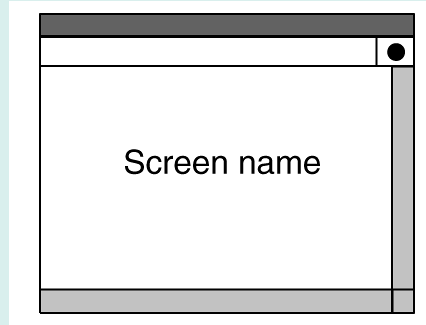
The User Experience (UX)

- Outlines the intended look-and-feel for the application
- Its purpose is to guide the team developing the user interface
- The user experience is driven and shaped by two philosophies: **the art** and **the architecture**
- The Artifacts:
 - Screens and content descriptions
 - Storyboard scenarios
 - Navigational paths through the screens

UX Modeling with UML

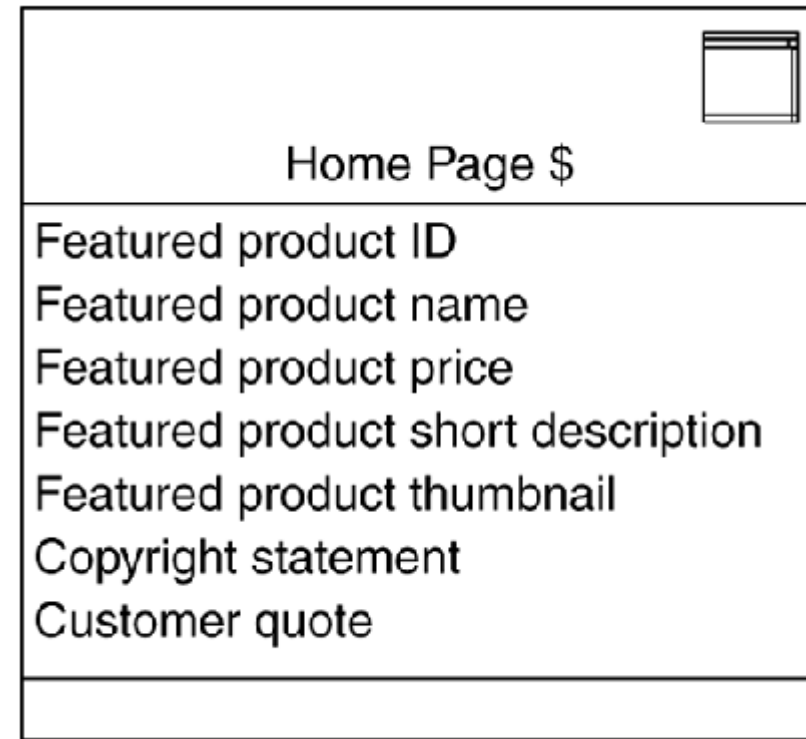
- Screen
- Input Form
- Screen Compartment
- Storyboard
- Navigational Path/Map

Screen



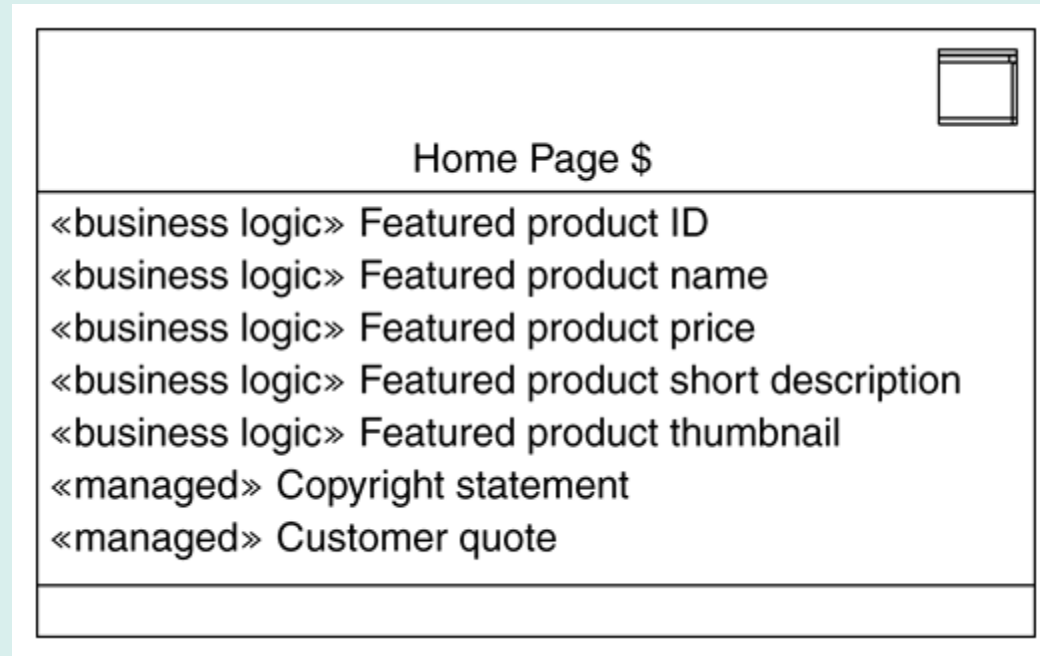
«Screen» class stereotype icon

→ modeling the dynamic content
of a single screen



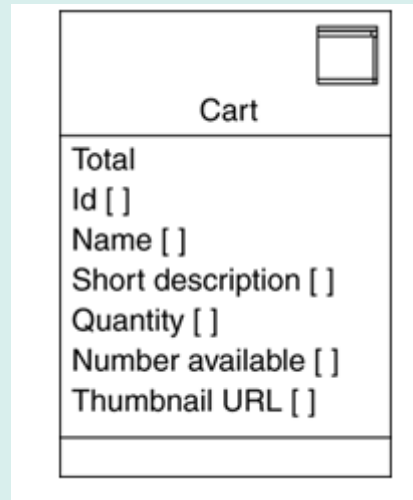
Simple dynamic screen content

Screen (2)

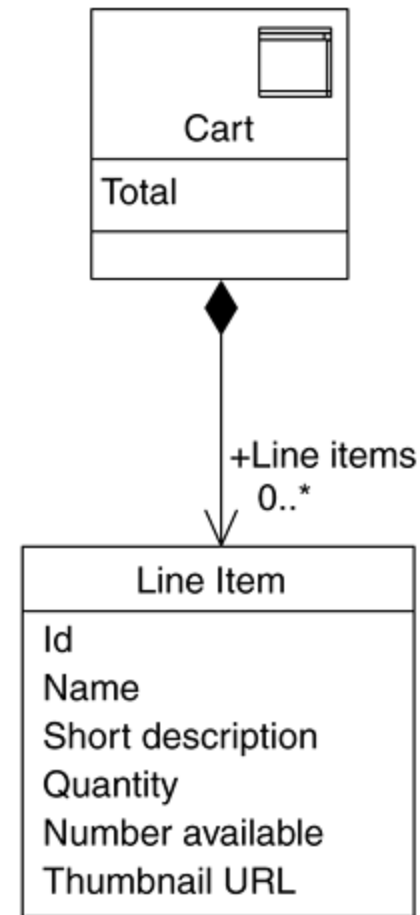


Screen with stereotyped attributes

Screen (3)



Variable-instance dynamic content model with array notation



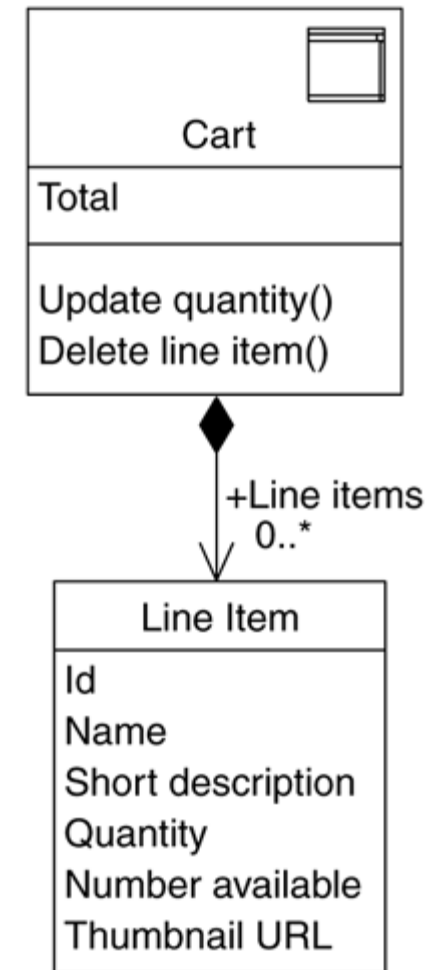
Variable-instance dynamic content model with contained class

Screen Behavior

- The behavior that the screen user can invoke
- One common "static" operation of a screen is `navigate_to()`,
 - which is called to create and to display a screen.

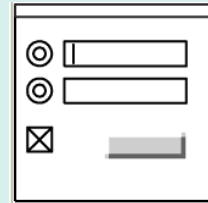
Screen Behavior (2)

Screen behaviors are captured as operations on the screen

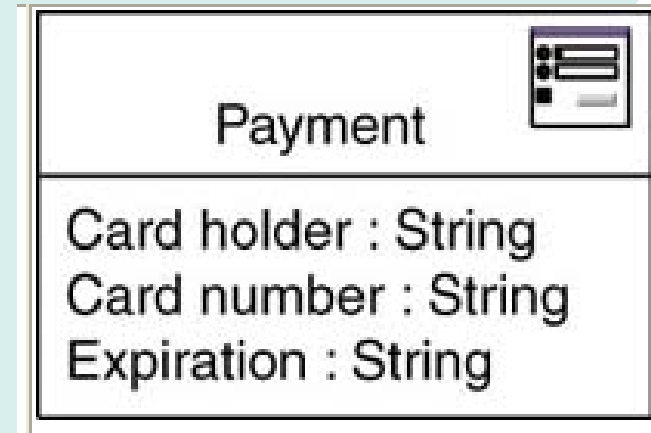


Shopping cart screen with defined operations

Input Form



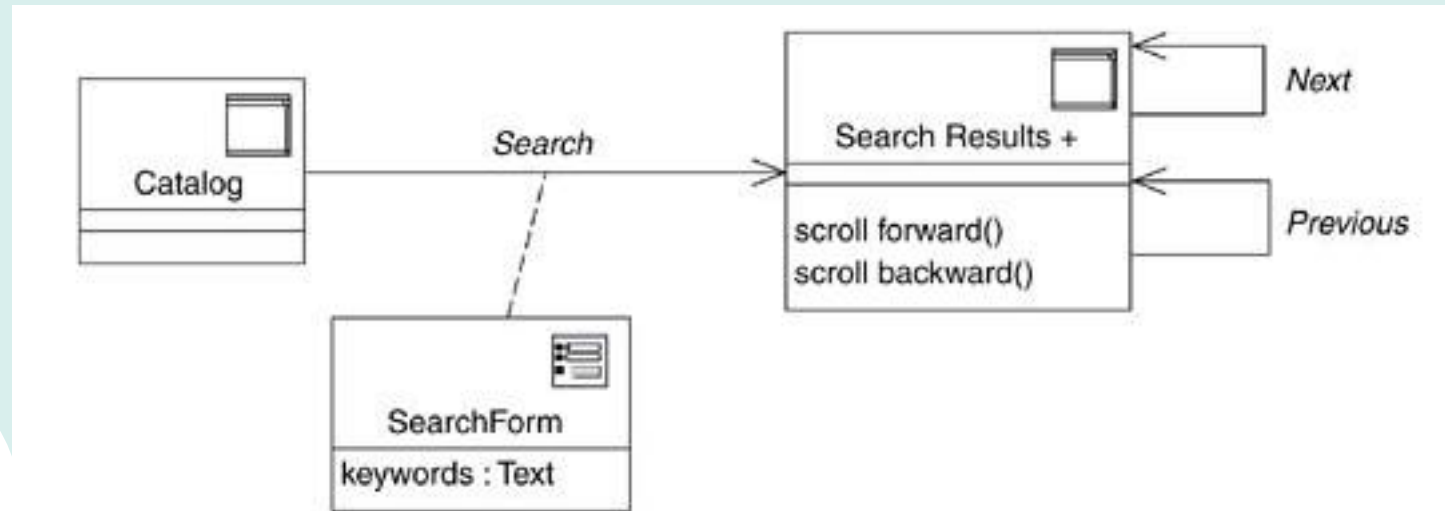
«input form» class stereotype icon



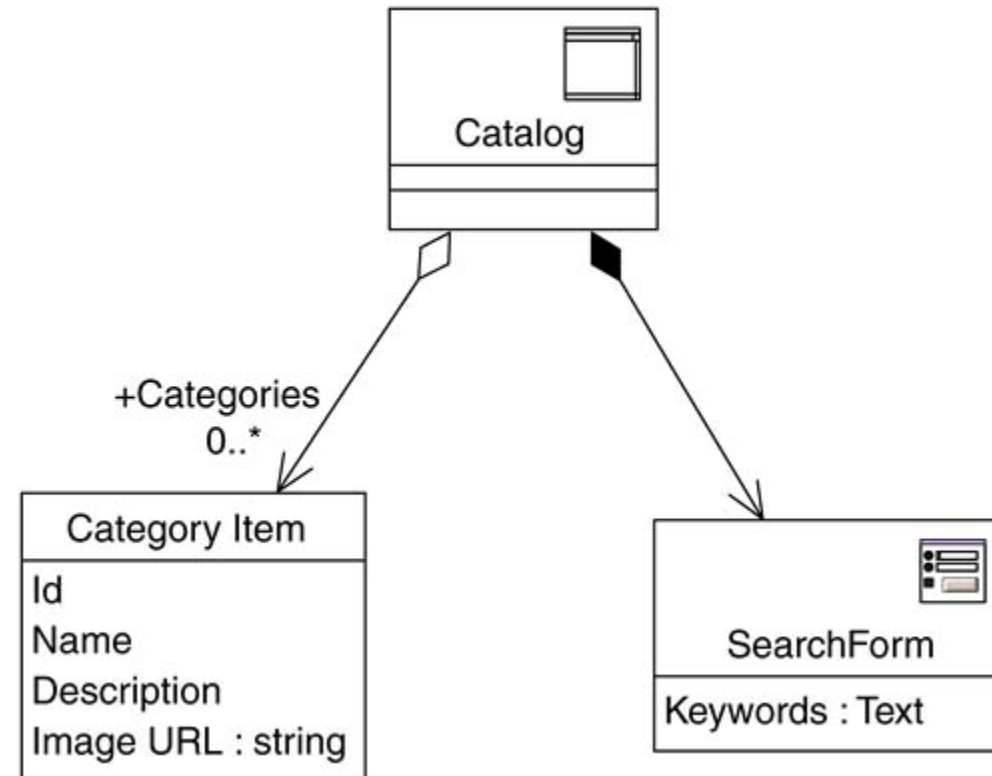
Simple Input Form

→ modeling a user input

Input form modeled with an association class



Input form modeled as a contained class

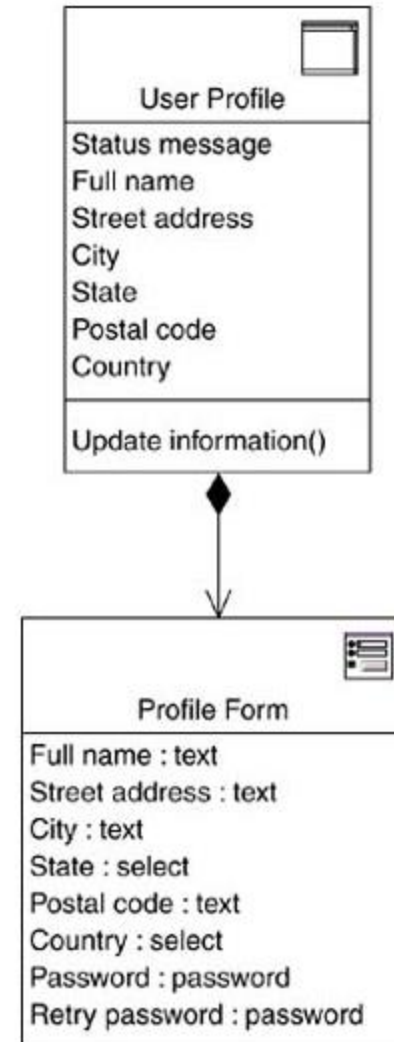


Input form with prefilled values

- It is tempting to combine input fields and dynamic content, especially when the dynamic content is used to prefill the input form.

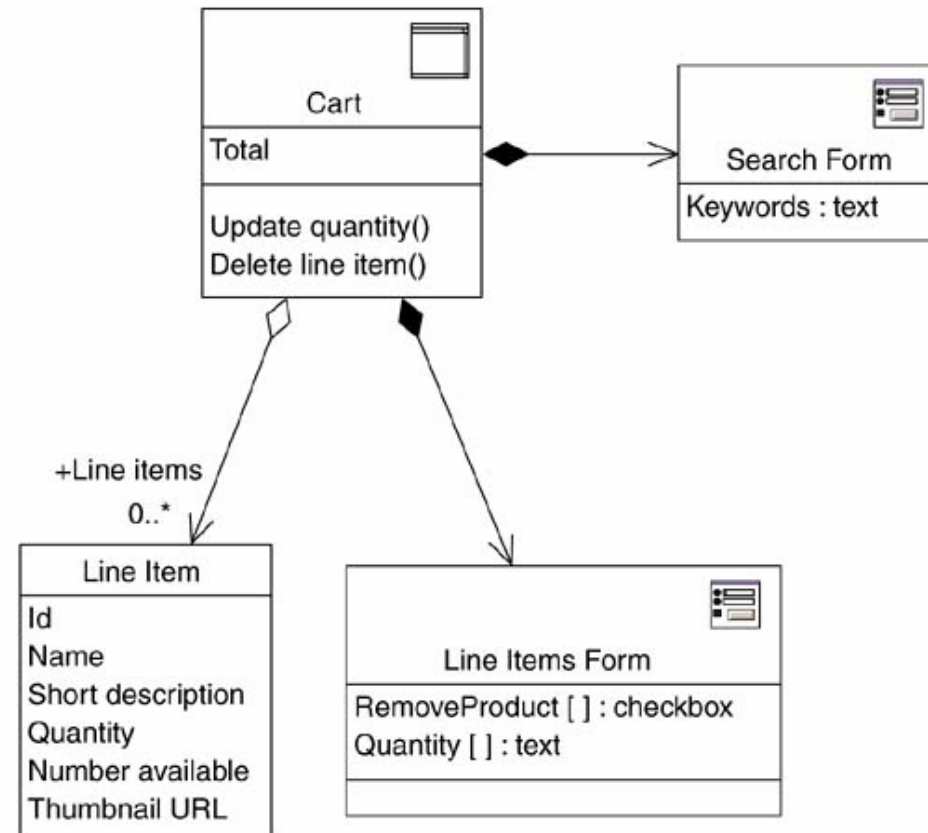
Example:

- A user profile screen allows the user to update personal profile information.
- As a convenience to the user, most of the screen is prefilled with some, but not all, the current information, if available.
- For security reasons, it is decided not to display password information.

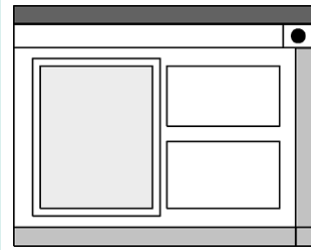


Multiple forms in the screen

- Each form is unique and designed with a completely different intent
- They are modeled with separate contained «input form» classes
- Each input form defines the fields that are part of it and has a name that indicates its purpose

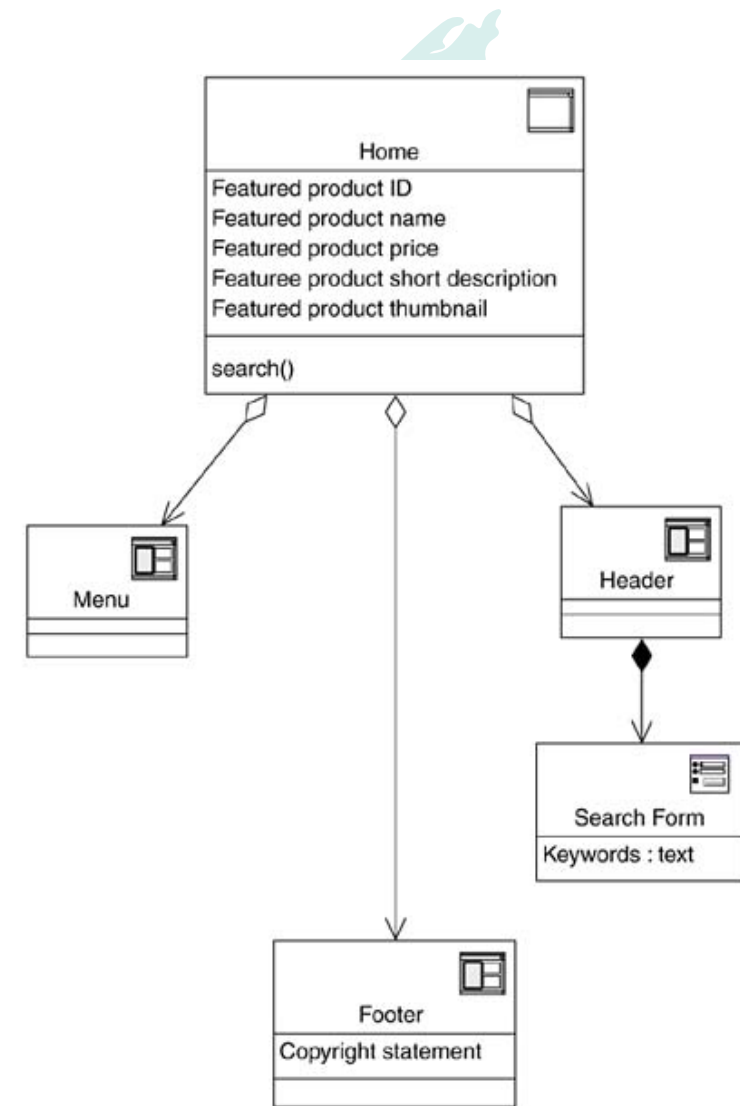


Screen Compartment



«screen compartment» class
stereotype icon

- modeling subscreens that are intended to be combined with other compartments to form a single screen

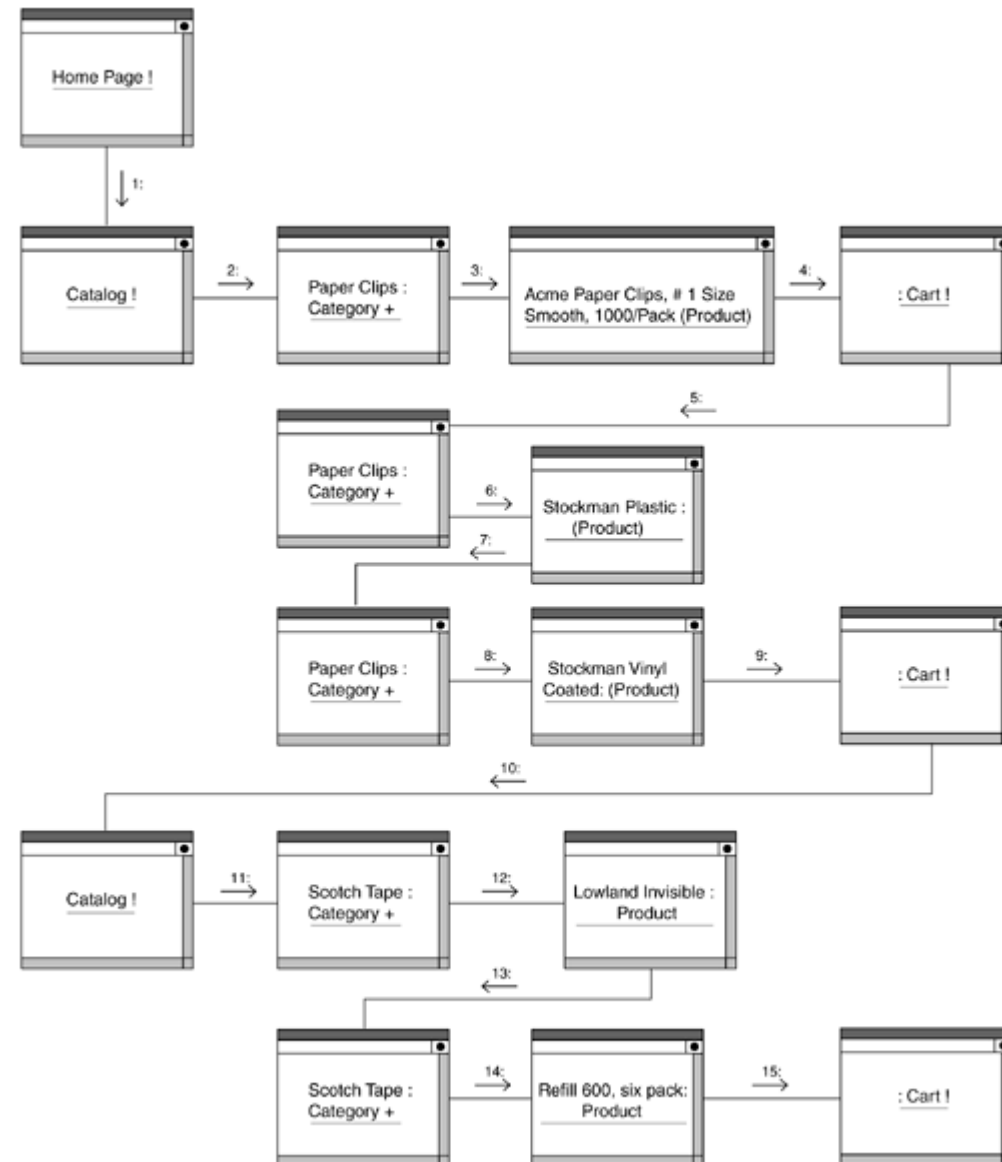


Defining screens with compartments

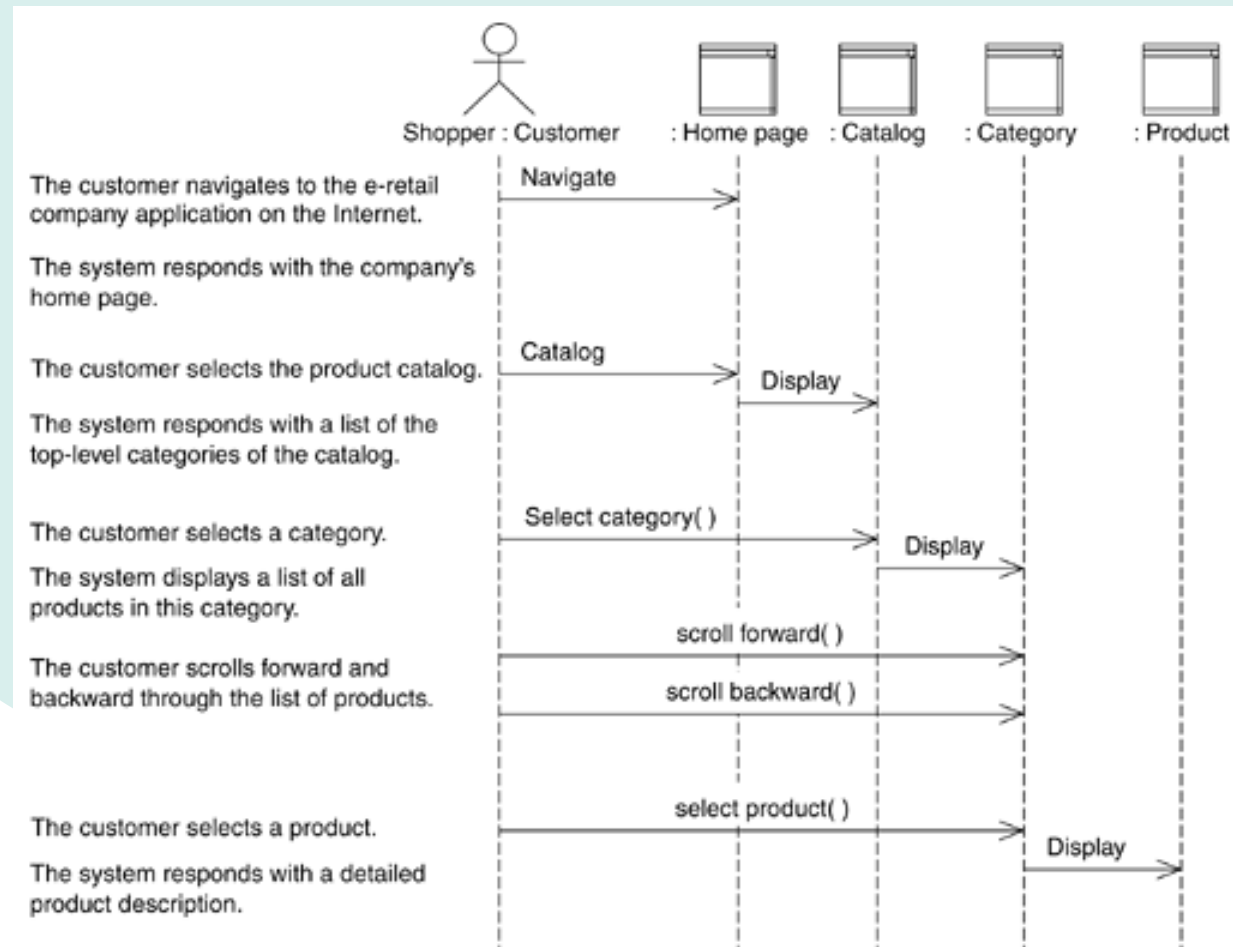
Storyboard

- A storyboard is a way to tell a story through the use of discrete static pictures
- Storyboards is a mechanism for understanding and structuring scenarios
 - The concept, used for comic books since the beginning of the twentieth century, allowed the animation team to visualize and to understand what they were about to create.

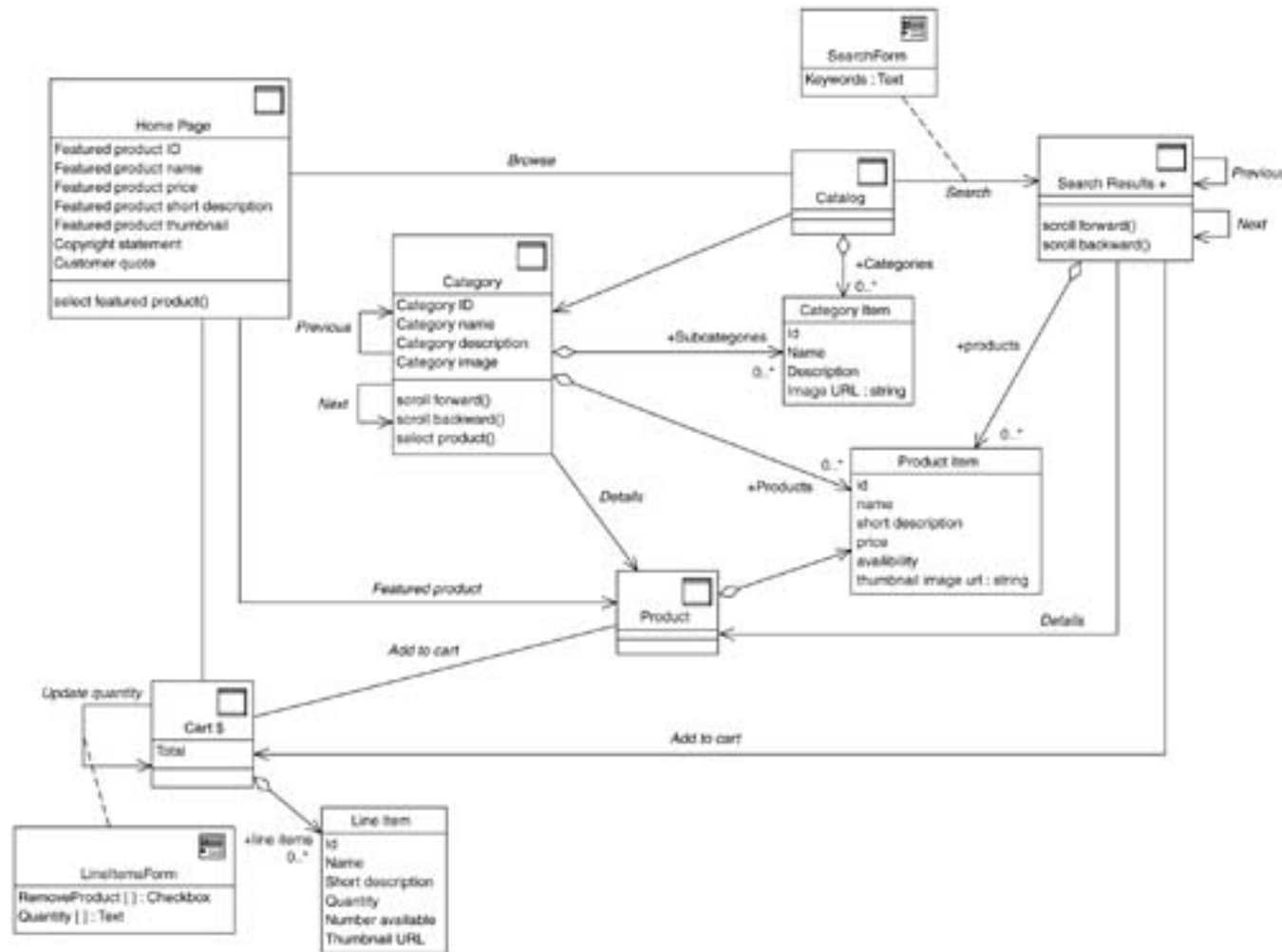
*Storyboard scenario
expressed as a UML
collaboration diagram*



Storyboard sequence for the Browse Catalog use case

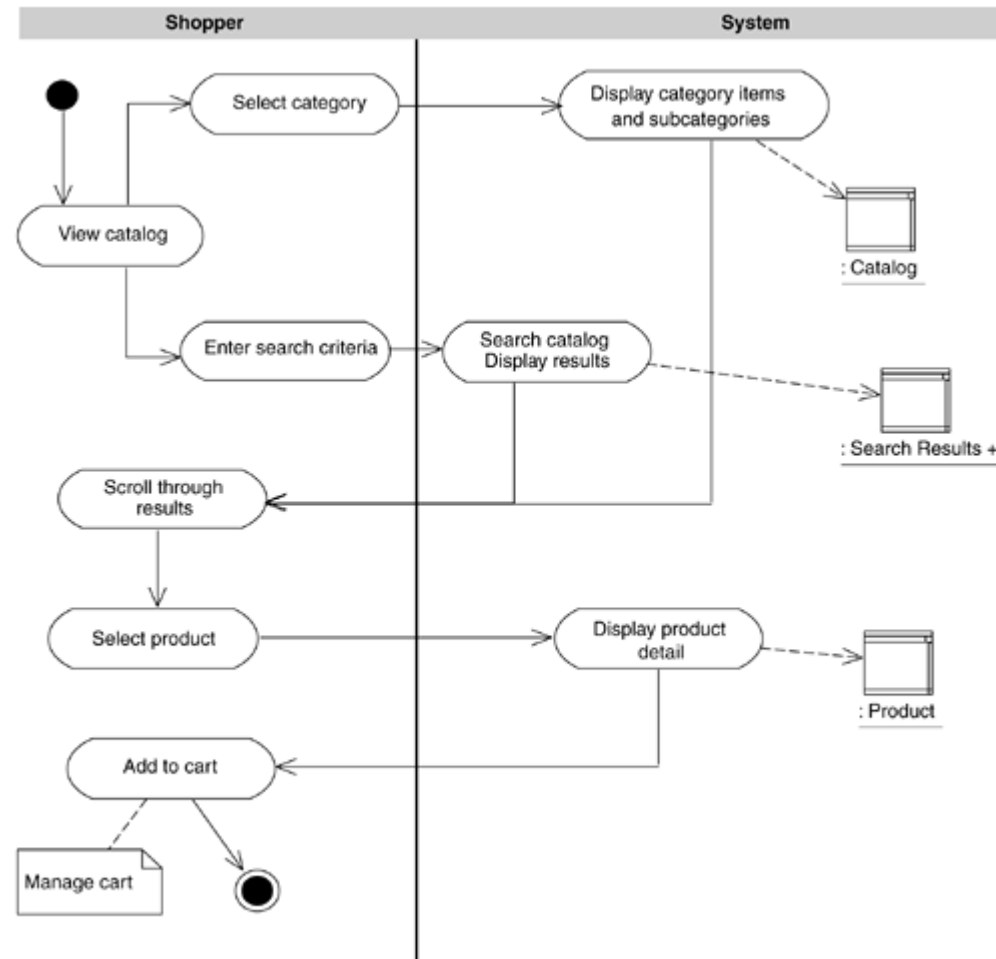


Participants diagram for Browse Catalog storyboard



Activity diagram describing the Browse Catalog use case

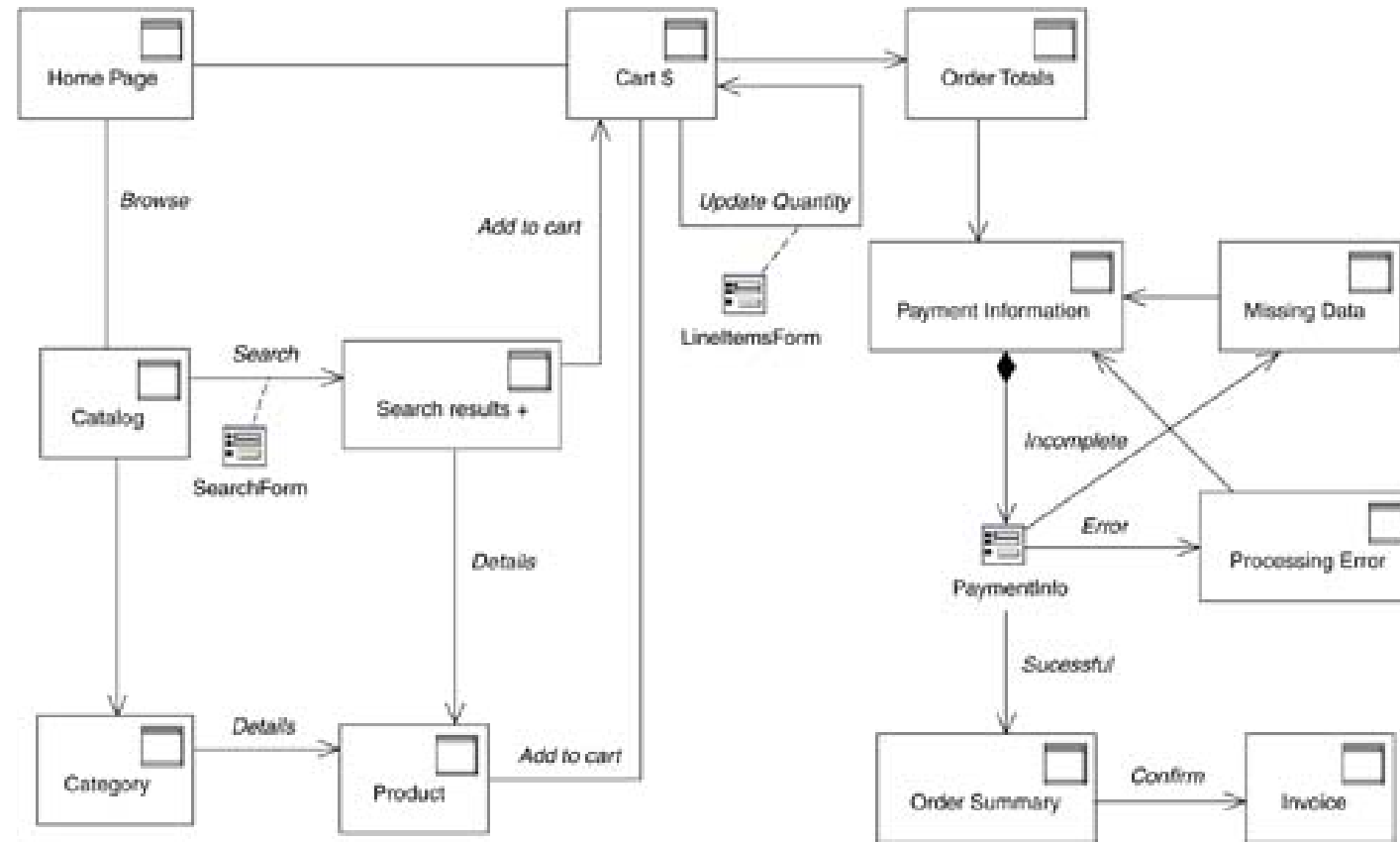
- An additional way to express navigational flow is with an activity diagram
- Link screen instances to the activities when appropriate to trace activities to UX screens



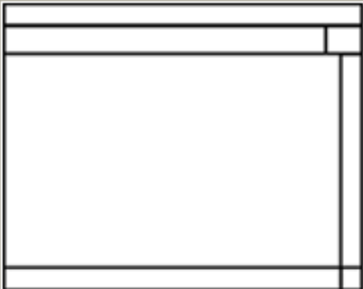

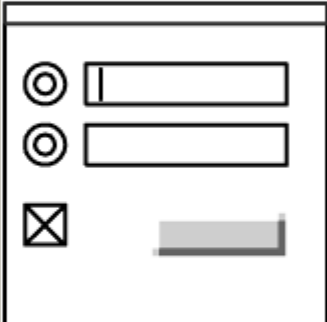

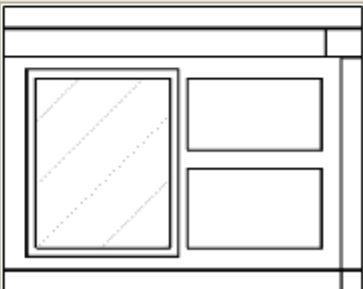

Navigational Map

- A top-level navigational map for a Web application
- Gives the cleanest top-level view of the "site."
 - If the application contains fewer than 50 screens, this can be accomplished, with the right tools, in a single diagram
 - For larger applications, it's best to divide the map into several diagrams, each focusing on related use cases
- Do not show screen details; render the classes with icons

A simple navigational map for an e-retail application



UX Model Stereotypes Summary

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

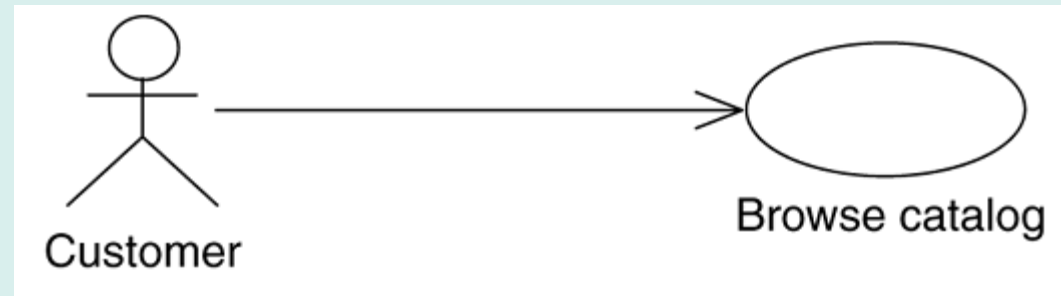
Analisis



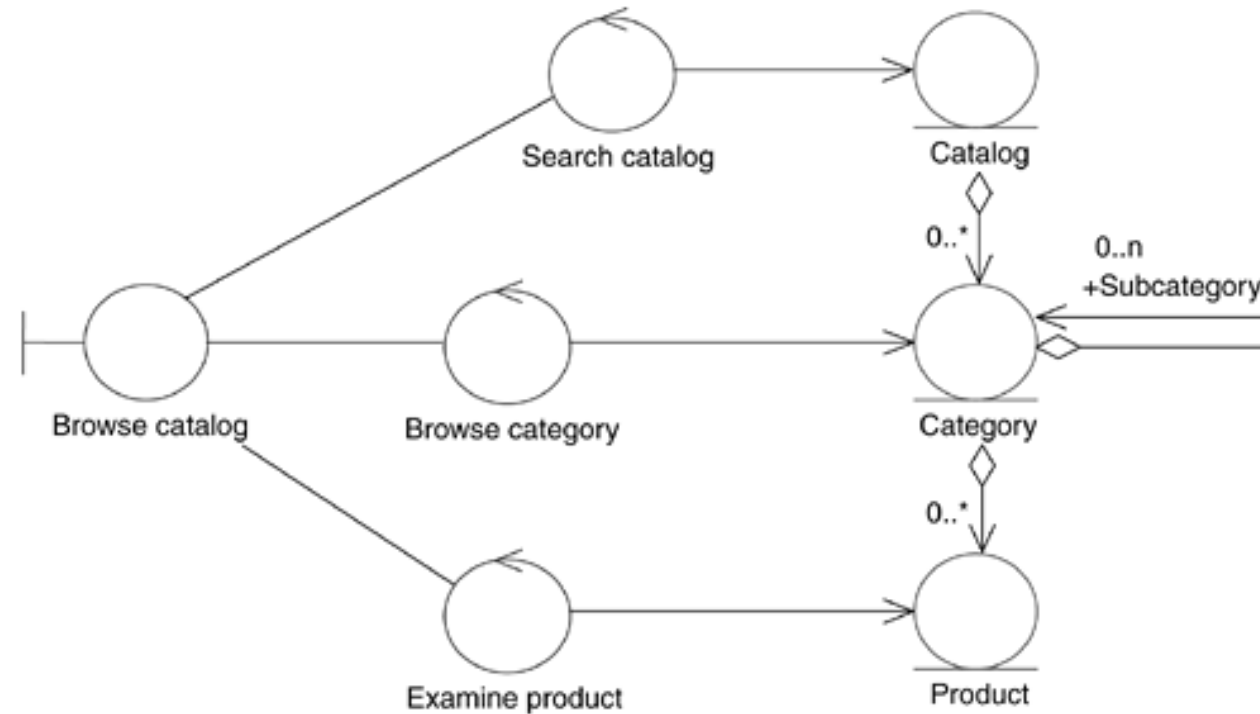
Analysis

- Use case analysis comprises those activities that take the use cases and functional requirements to produce an analysis model of the system
- The analysis model is made up of classes and collaborations of classes that exhibit the dynamic behaviors detailed in the use cases and the requirements

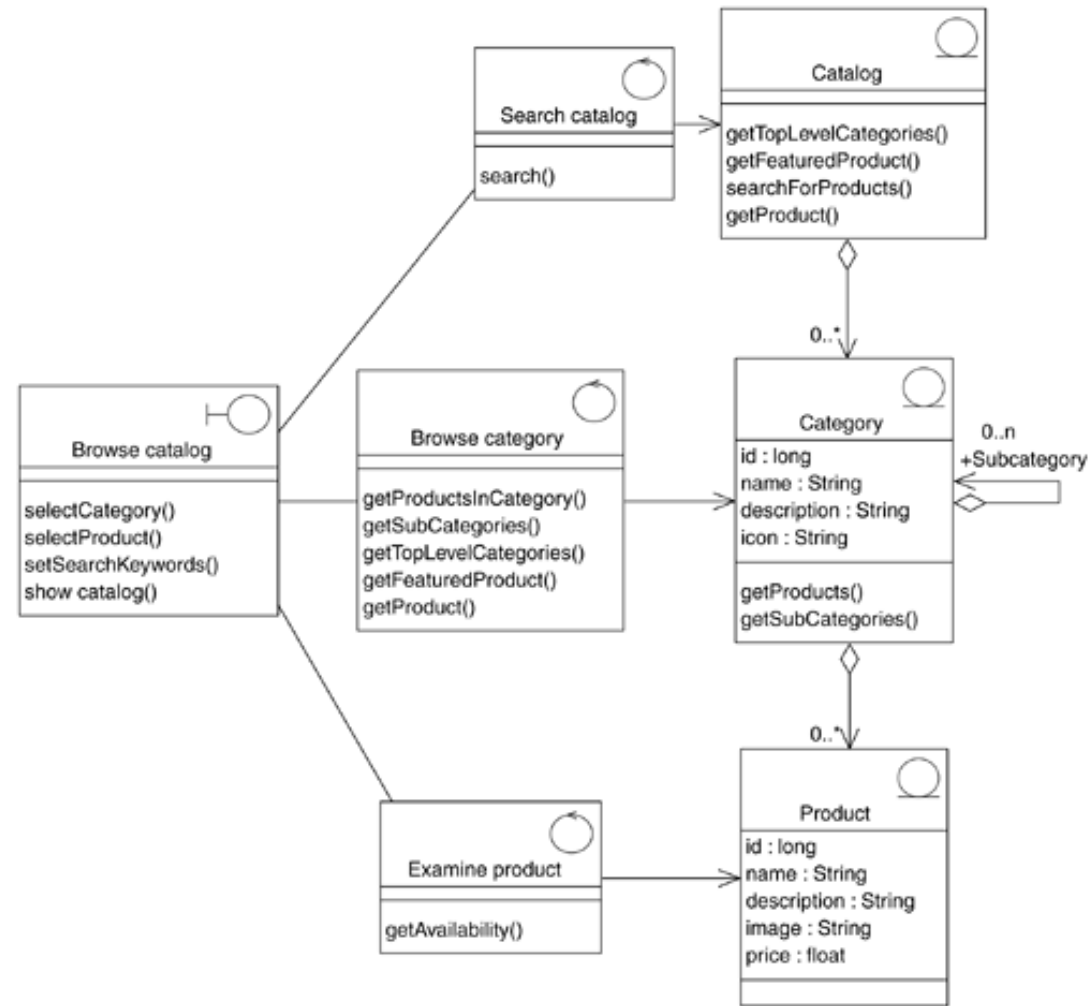
Example – Browse the catalog



First-pass analysis model for the Browse Catalog use case



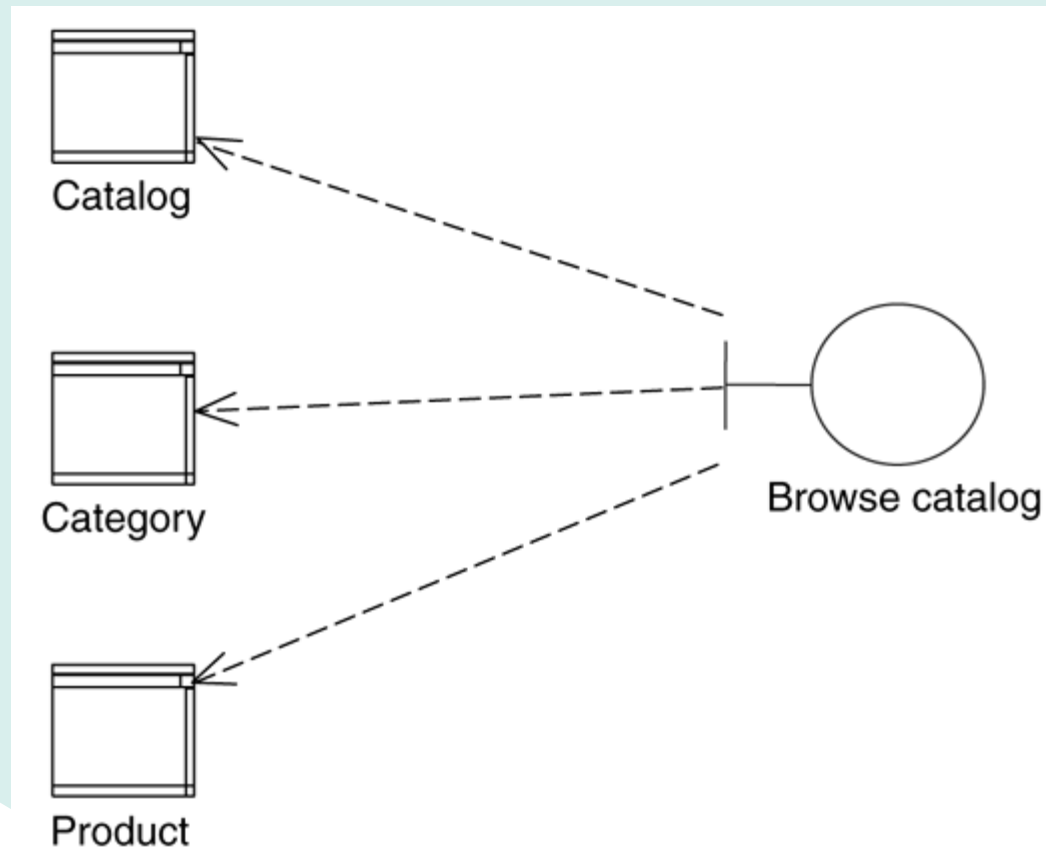
Elaborated analysis model elements



UX Model Mapping

- A useful mapping to create is that between the UX and analysis models
 - The mapping is simple, expressed as dependencies from analysis boundary classes and UX screens
- Creating this mapping early helps to ensure that both the UX and the analysis teams are working together on the solution to the same problem

UX Model Mapping - Example



Perancangan



Design

- Design starts with the analysis model, the user experience model, and the software architecture document as the major inputs

Web Application Extension (WAE) to UML

- Enables us to represent Web pages and other architecturally significant elements in the model alongside the "normal" classes of the model
- An extension to UML is expressed in terms of
 - stereotypes,
 - tagged values, and
 - constraints

These mechanisms enable us to extend the notation of UML, enabling us to create new types of building blocks that we can use in the model

WAE to UML (2)

- **Stereotype**

An extension to the vocabulary of the language; allows us to attach a new semantic meaning to a model element

- usually represented as a string between a pair of guillemets: « », or rendered as a new icon

- **Tagged value**

An extension to a property of a model element; the definition of a new property that can be associated with a model element

- rendered as a string enclosed by brackets

- **Constraint**

An extension to the semantics of the language, specifies the conditions under which the model can be considered well formed

- a rule that defines how the model can be put together
- rendered as strings between a pair of braces: { }

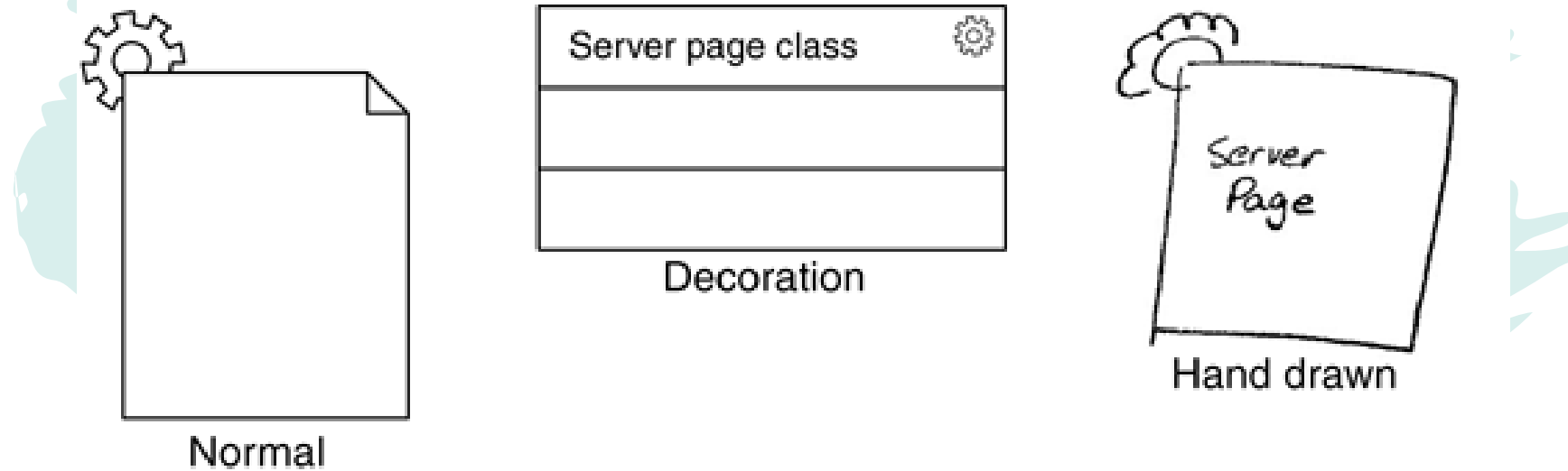


WAE - Logical View

- Consists mostly of classes, their relationships, and their collaborations
- Some stereotyped classes define multiple icons
- Defines three core class stereotypes and various association stereotypes:
 - Server page
 - Client page
 - HTML form

Server Page

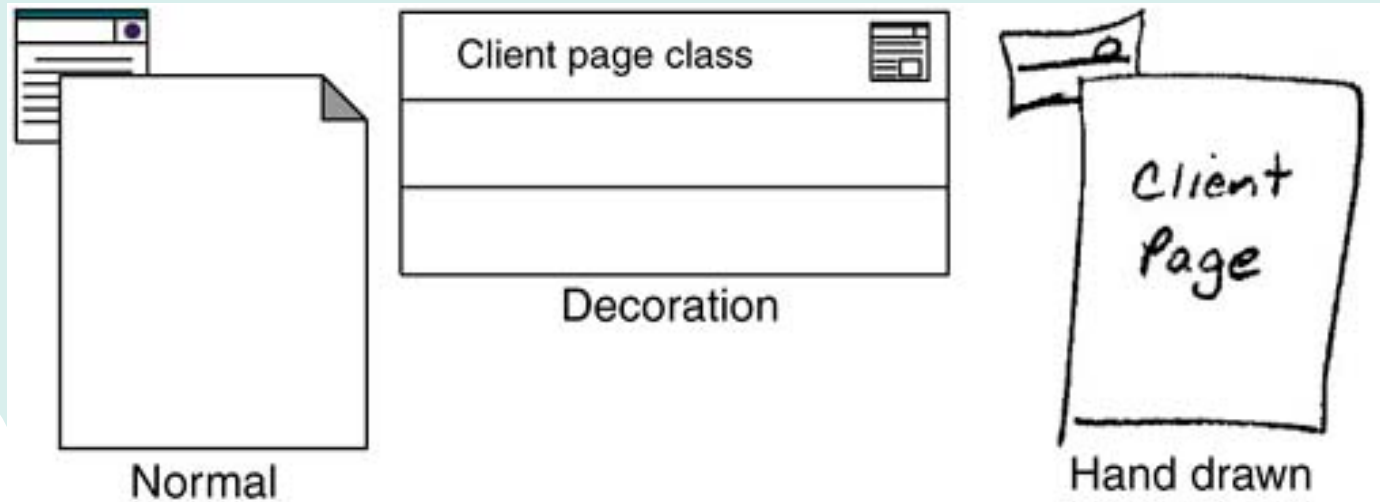
- Represents a dynamic Web page that contains content assembled on the server each time it is requested
- Typically, contains scripts that are executed by the server that interacts with server-side resources
- The object's operations represent the functions in the script
- The object's attributes represent the variables that are visible in the page's scope, accessible by all functions in the page
- Constraints:
 - Server pages can have only normal relationships with objects on the server



Client Page

- An HTML-formatted Web page with a mix of data, presentation, and even logic; rendered by client browsers and may contain scripts that are interpreted by the browser
- Client page functions map to functions in tags in the page
- Client page attributes map to variables declared in the page's script tags that are accessible by any function in the page, or page scoped
- Client pages can have associations with other client or server pages
- 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

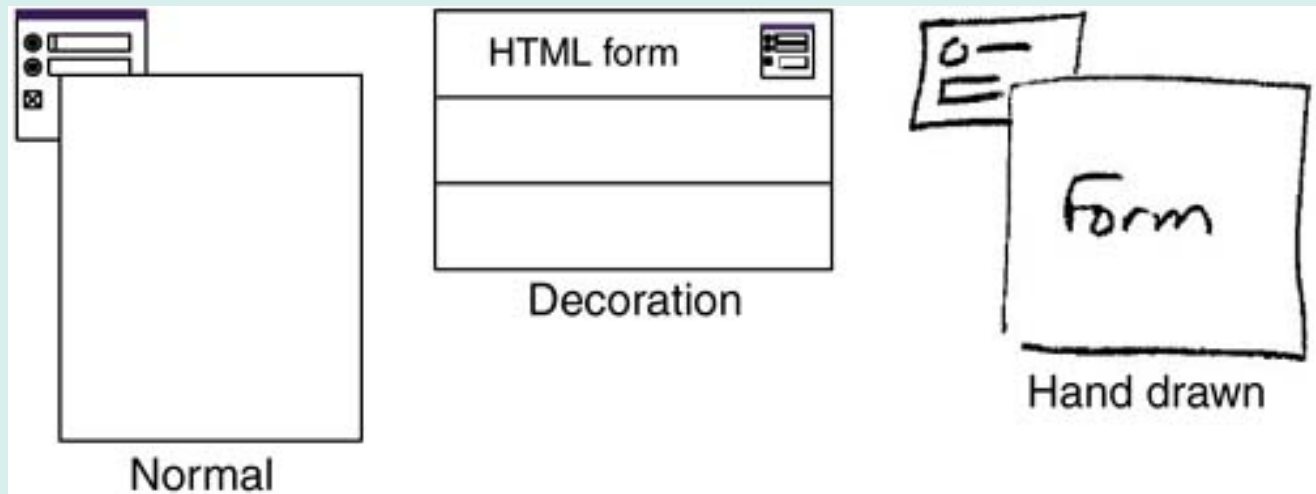
Client Page (2)



HTML Form

- A collection of input fields that are part of a client page
 - Maps directly to the HTML <form> tag.
 - Its attributes represent the HTML form's input fields: input boxes, text areas, radio buttons, check boxes, and hidden fields.
 - Has no operations
 - Any operations that interact with the form would be the property of the page that contains the form
- Tagged values:
 - GET or POST: the method used to submit data to the action URL

HTML Form (2)



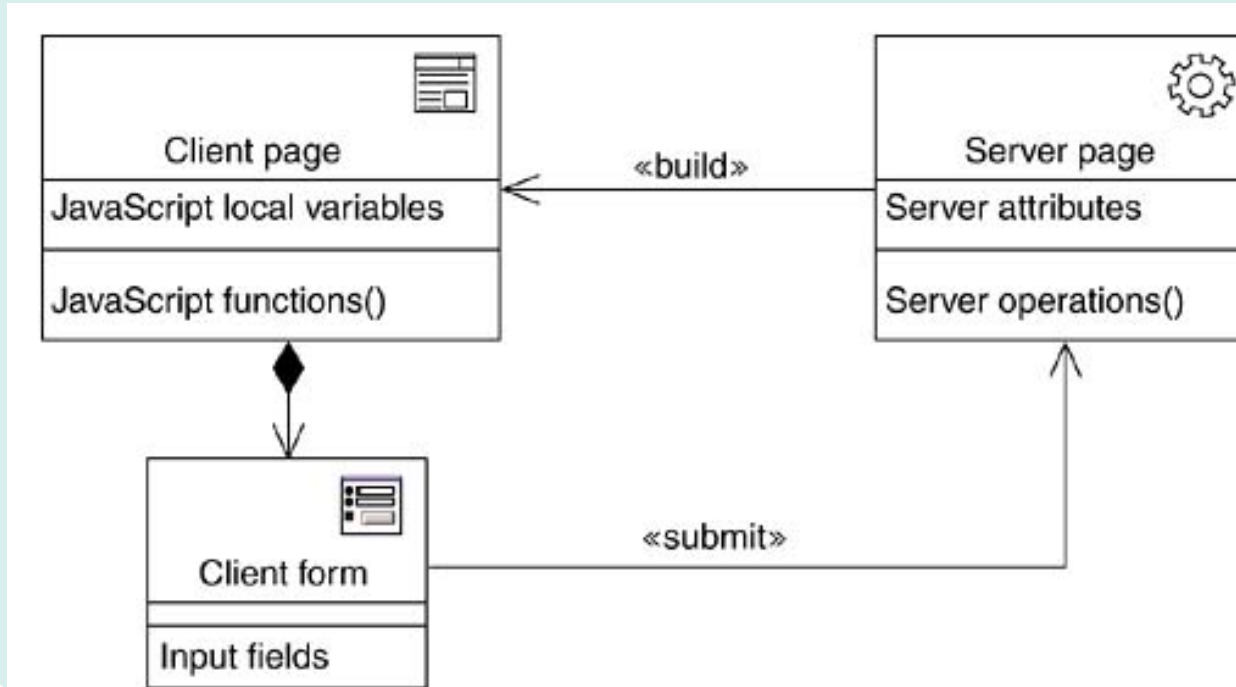
Association Stereotypes

- `<<link>>`
is an abstraction of the HTML anchor element, when the href attribute is defined
- `<<build>>`
a directional relationship between a server page and a client page
- `<<submit>>`
a directional relationship between an HTML form and a server page
- `<<redirect>>`
indicates a command to the client to request another resource

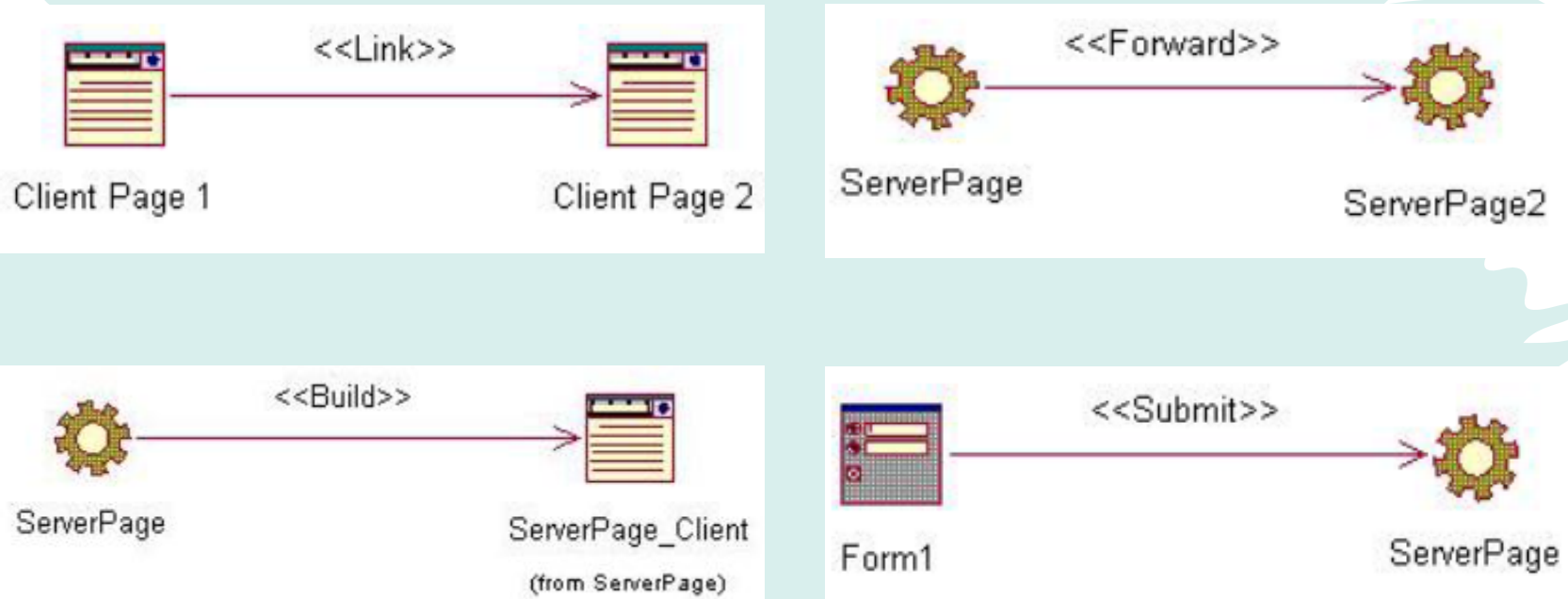
Association Stereotypes (2)

- <<forward>>
represents the delegation of processing a client's request for a resource to another server-side page
- <<object>>
a containment relationship drawn from a client page to another logical class, typically one that represents an applet, ActiveX control, or other embeddable component
- <<include>>
indicates that the included page gets processed, if dynamic, and that its contents or by-products are used by the parent

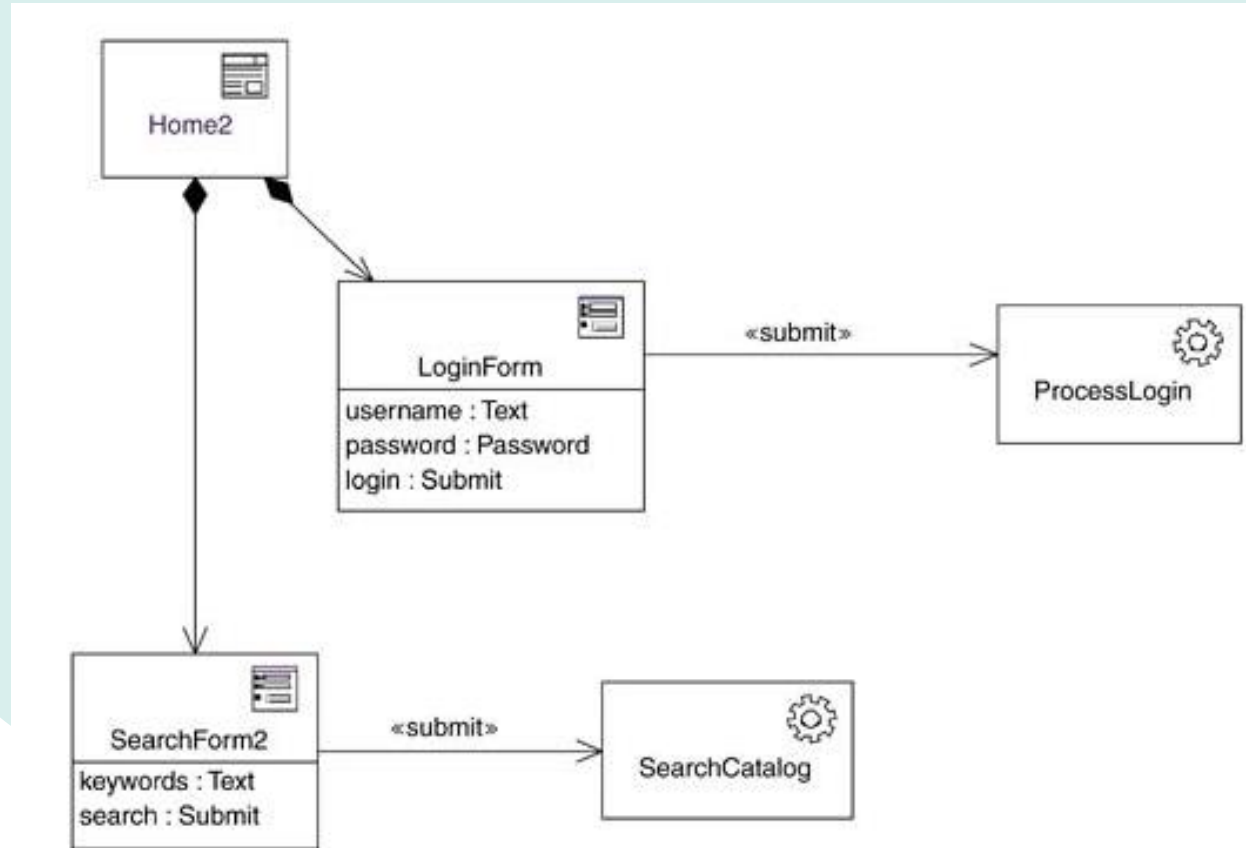
Basic relationships among WAE stereotyped elements



Association Stereotypes



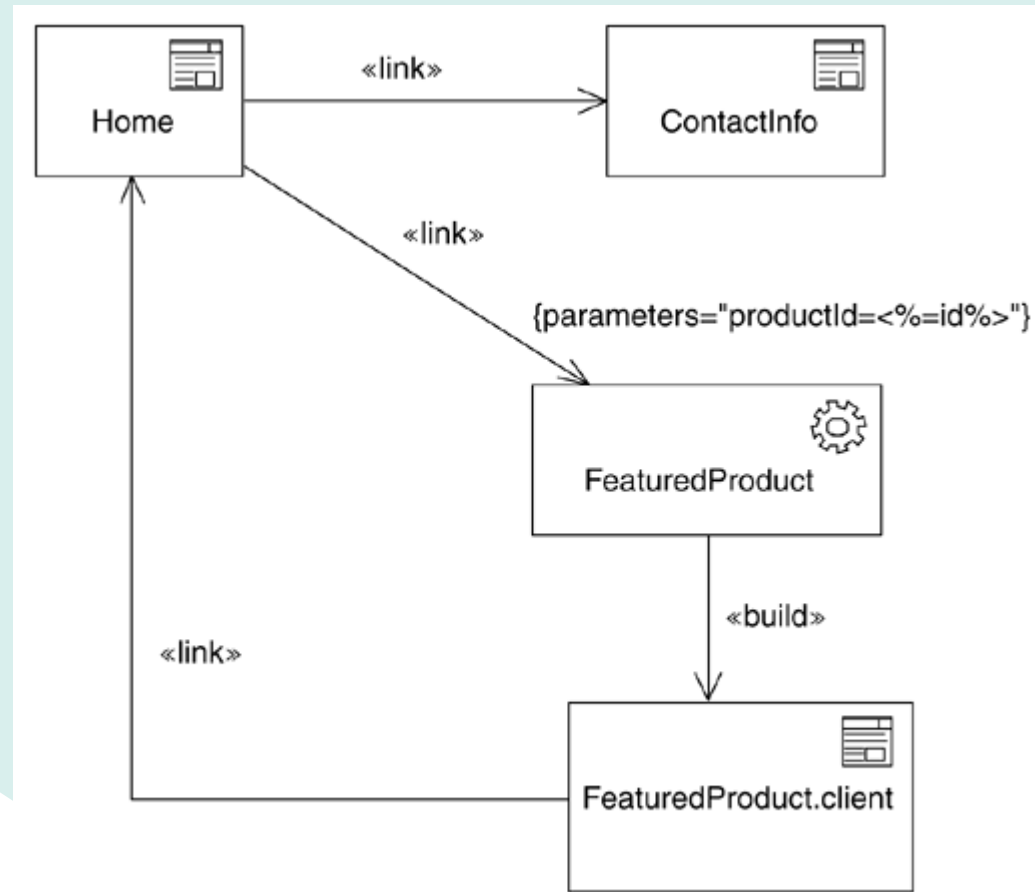
Multiple forms in client pages



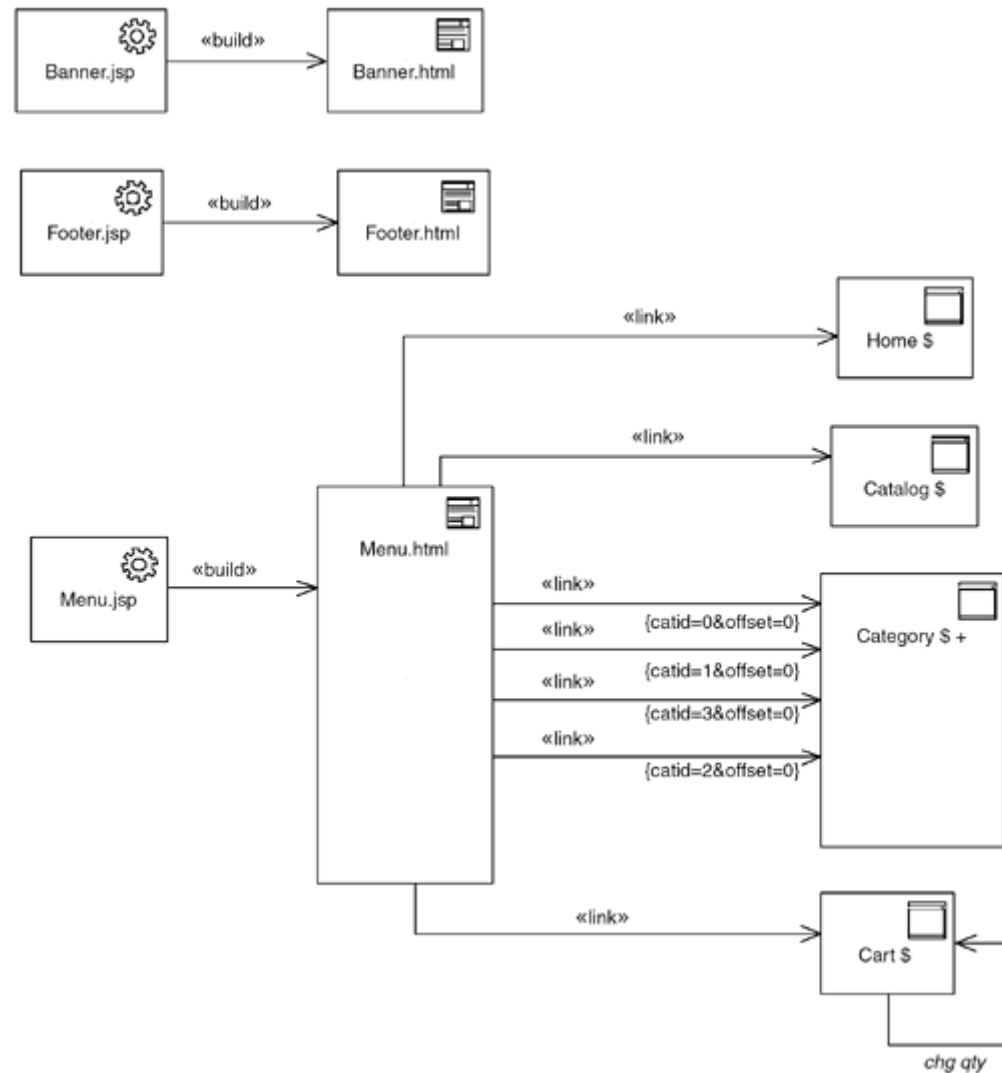
Simple client page «link» associations



Stereotyped «link» associations originating from client pages

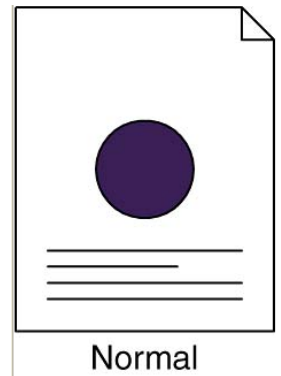


The «link» associations pointing to «screen» elements when screens are compartmentalized

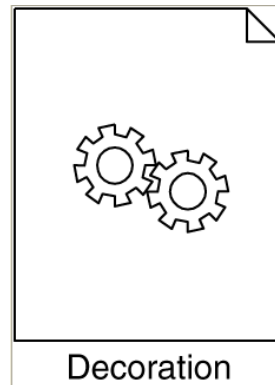


WAE – Component View

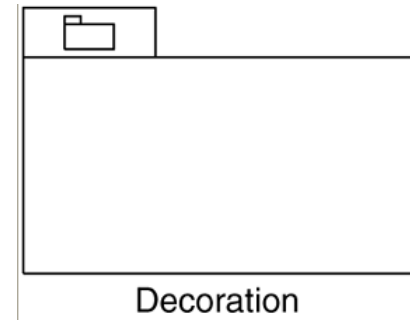
- Static Page
- Dynamic Page
- Physical Root



Static Page



Dynamic Page



Physical Root

Static Page

- Metamodel: Component
- Can be directly requested by a client browser
 - Performs no server-side execute
 - Delivered directly from the file system to the client intact

- Constraint:

Cannot realize logical components that execute on the server, that is, server pages.

Static pages can realize only client pages

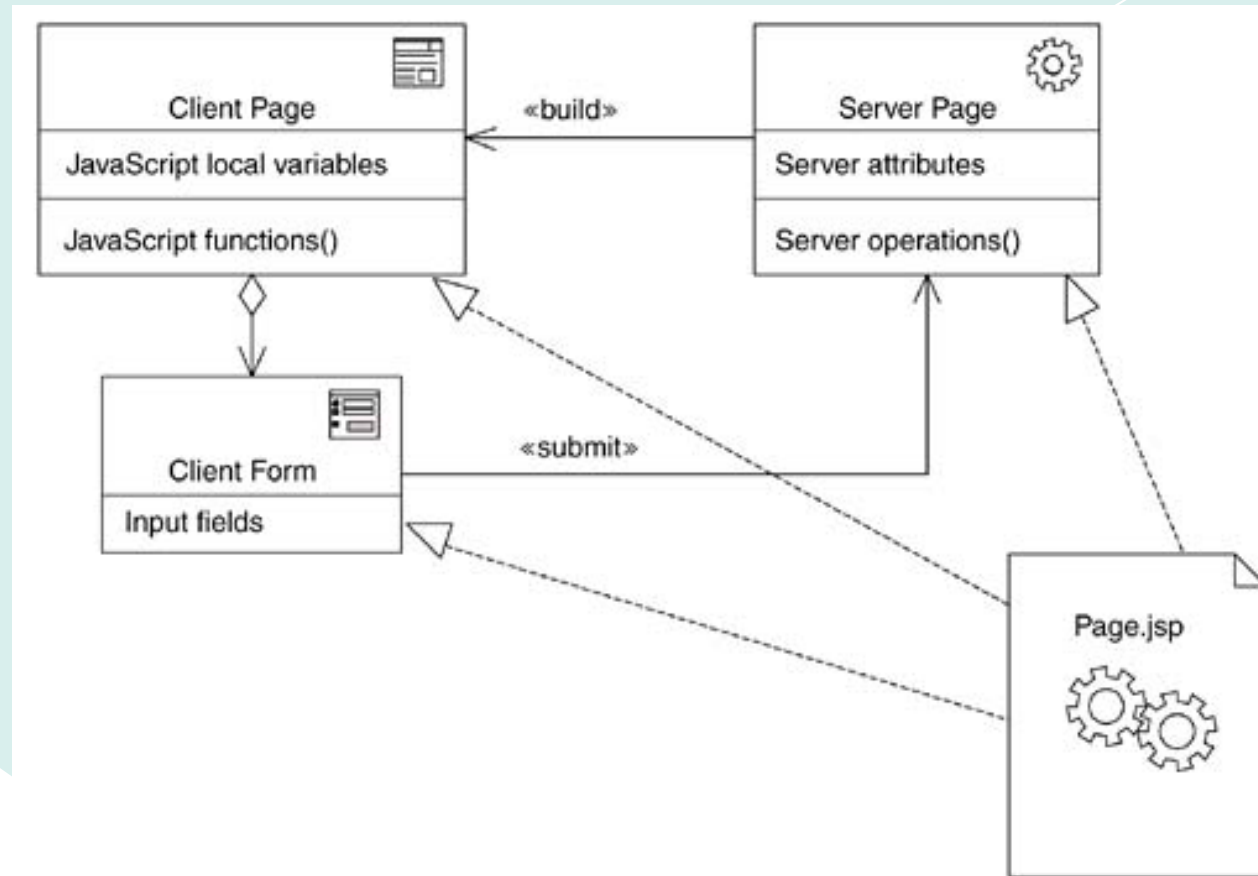
Dynamic Page

- Metamodel: Component
- Can be requested by a client browser
 - When requested or delegated to via a «forward» relationship, server-side processing takes place
 - The results of this processing can change the state of the server and be used to construct some of the HTML that is streamed out to the requesting client
- Can accept user input submitted by forms
- Constraints: Must realize a single server page

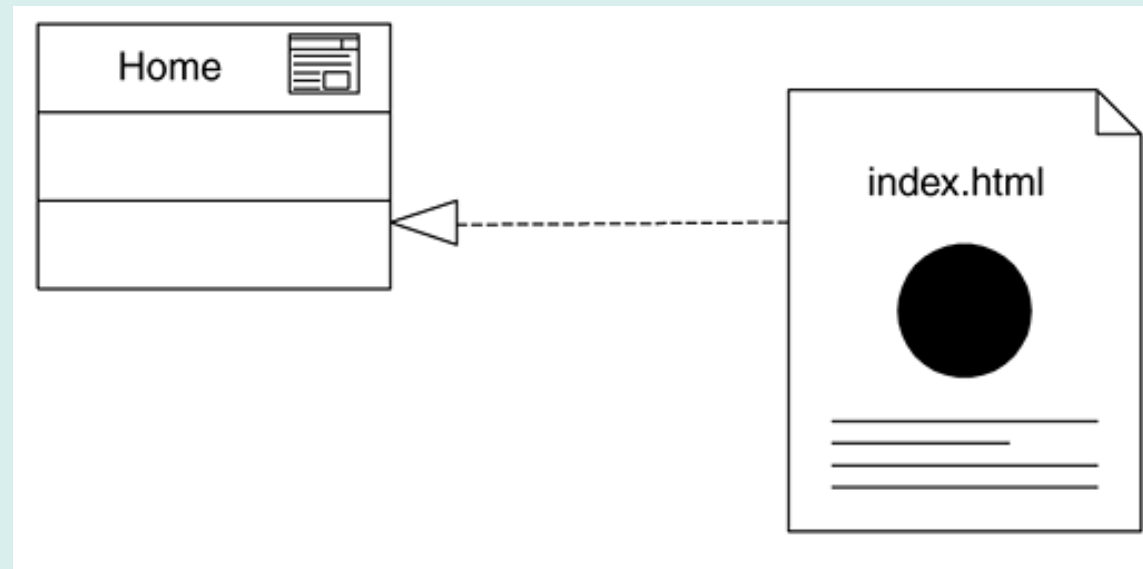
Physical Root

- Metamodel: component package
- A file hierarchy that contains requestable resources
 - Clients request static or dynamic files directly from this hierarchy
 - Maps directly to a Web server file system directory
- Tagged values:
 - Host name, the name of the host of the Web server, such as `www.mycompany.com`.
 - Context, the application context. The context appears as a top-level directory, such as `www.myco.com/appcontext`.

Logical-view classes realized by dynamic page component



Client pages realized by static page components



Designing Web App

- Most of the activities are the same as for any client/server system: partitioning the objects into the system's tiers and developing the necessary infrastructure and helper classes to add to the analysis model
- Proper partitioning of the business objects in a Web application is critical and depends on the architecture
 - **Objects may reside exclusively on the server, the client, or both**
 - Thin Web client applications place all objects behind the server, running either on the Web server or on another tier associated with the server.
 - Thick Web client applications allow some objects to execute on the client.
 - Web delivery applications have the most freedom in the placement of objects, being essentially distributed object systems that happen to use a browser.



Thick Web client Web applications

- For the most part, **persistent objects, container objects, shared objects, and complex objects all belong on the server**
 - Objects with associations to server resources, such as databases and legacy systems, also belong in the server tier
 - Objects that maintain static associations or dependencies with any of these objects must also exist on the server
- **An object can exist on the client if it has no associations or dependencies with objects on the server** and has associations and dependencies only with other client resources, such as browsers and Java applets
 - Candidate objects for the partitioning on the client are field validation objects, user interface controls, and navigation assisting controls
 - Client objects can be implemented with JavaScript, JavaBeans, applets, ActiveX (COM), or even plug-ins



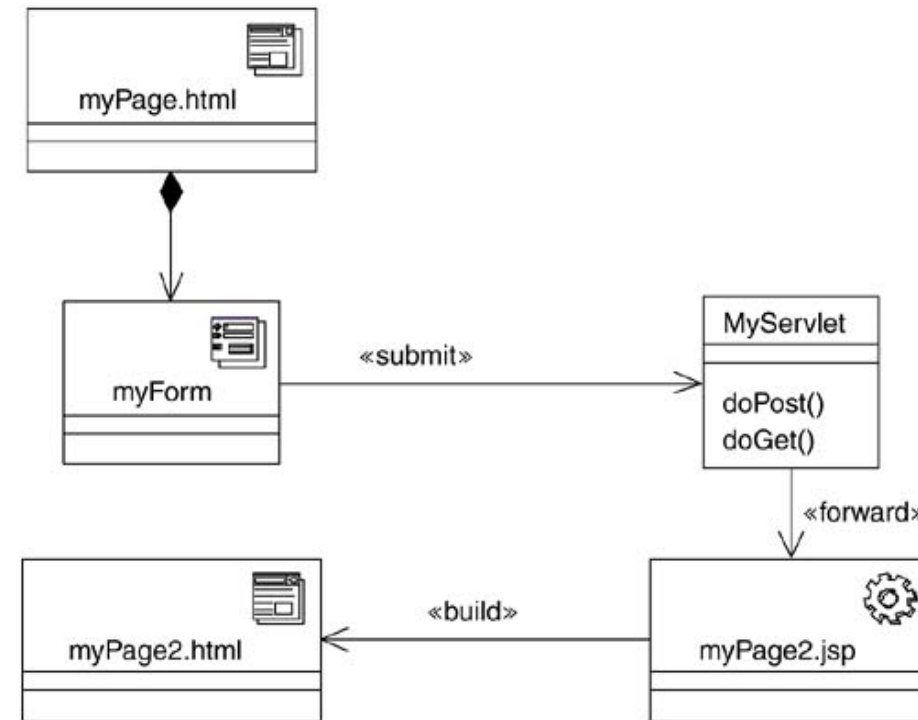
Web Delivery Web Applications

- One of the primary reasons for distributing objects to the client is **to take some of the load off the server**
- It is also natural to place objects in the part of the system where they will be most effective
 - As a general rule, **place objects where they have the easiest access to the data and the collaborations they require** to perform their responsibilities
 - If an object can exist on the client and if most, if not all, its associations are on client objects, that object is a likely candidate for placement on the client

JavaServer Page Model 2 Architecture

69

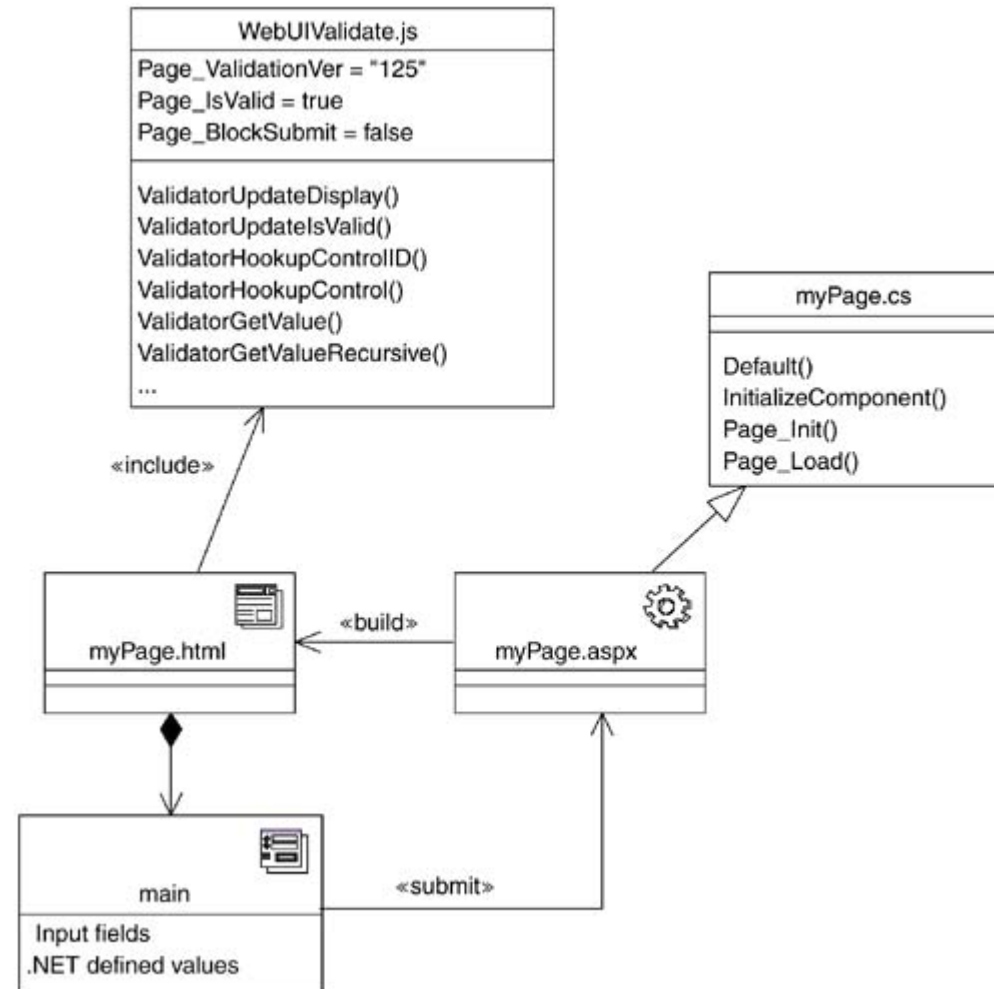
- Separating the two mechanisms for accepting and processing input and building output
- Using servlets (a completely Java-written component) for accepting all user-supplied input
- The servlet delegates the building of the response page to a JSP
 - JSP are more appropriate for building HTML output since the majority of the code in the component is often HTML.



..NET paradigm for handling user input

70

- myPage.aspx is modeled with a «server page» stereotyped class
- The C# code behind class myPage.cs is a superclass to the ASPX class and contains the majority of the event-handling code
- The general strategy is: to leave the ASPX file to focus on building the output page.



IF2250 RPL

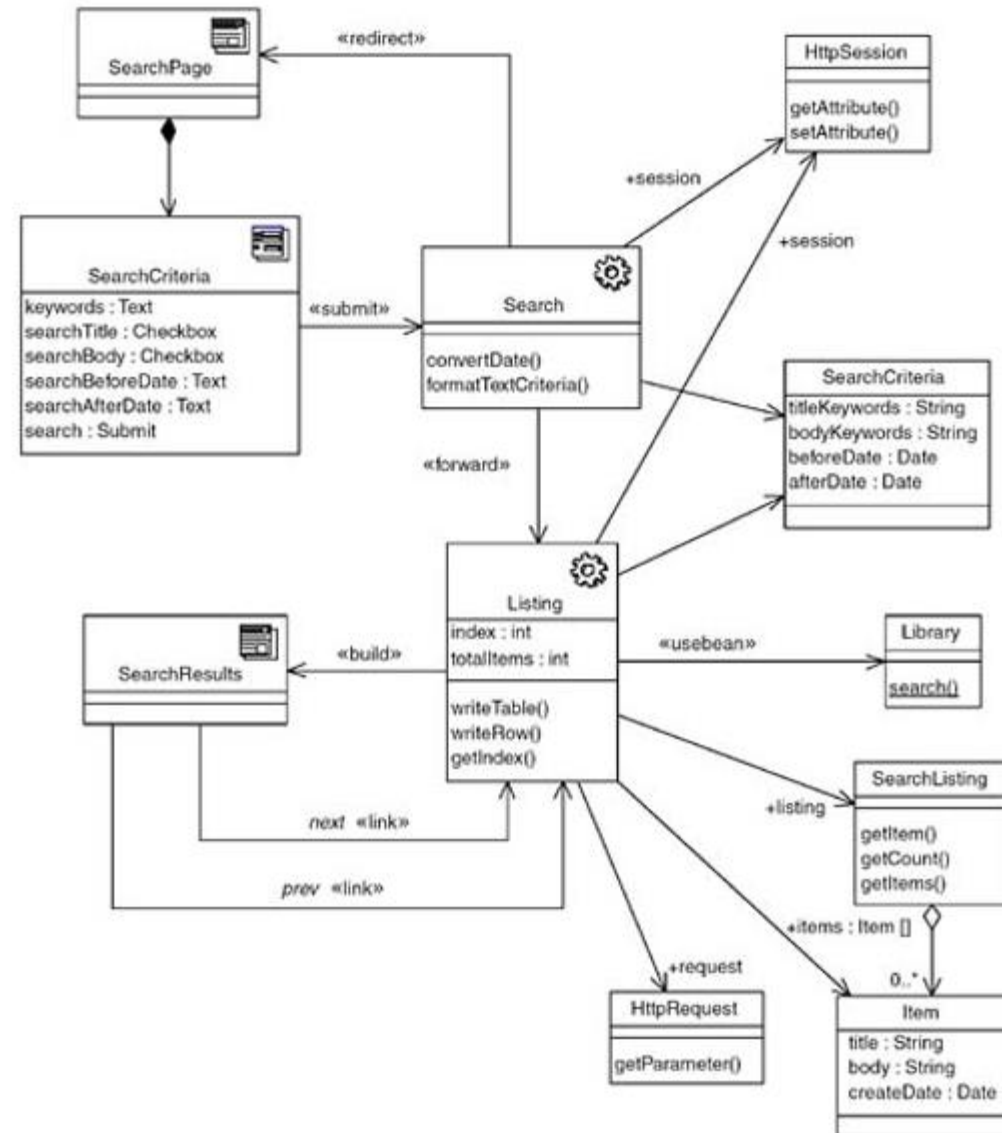
Design model fragment of catalog search functionality

71

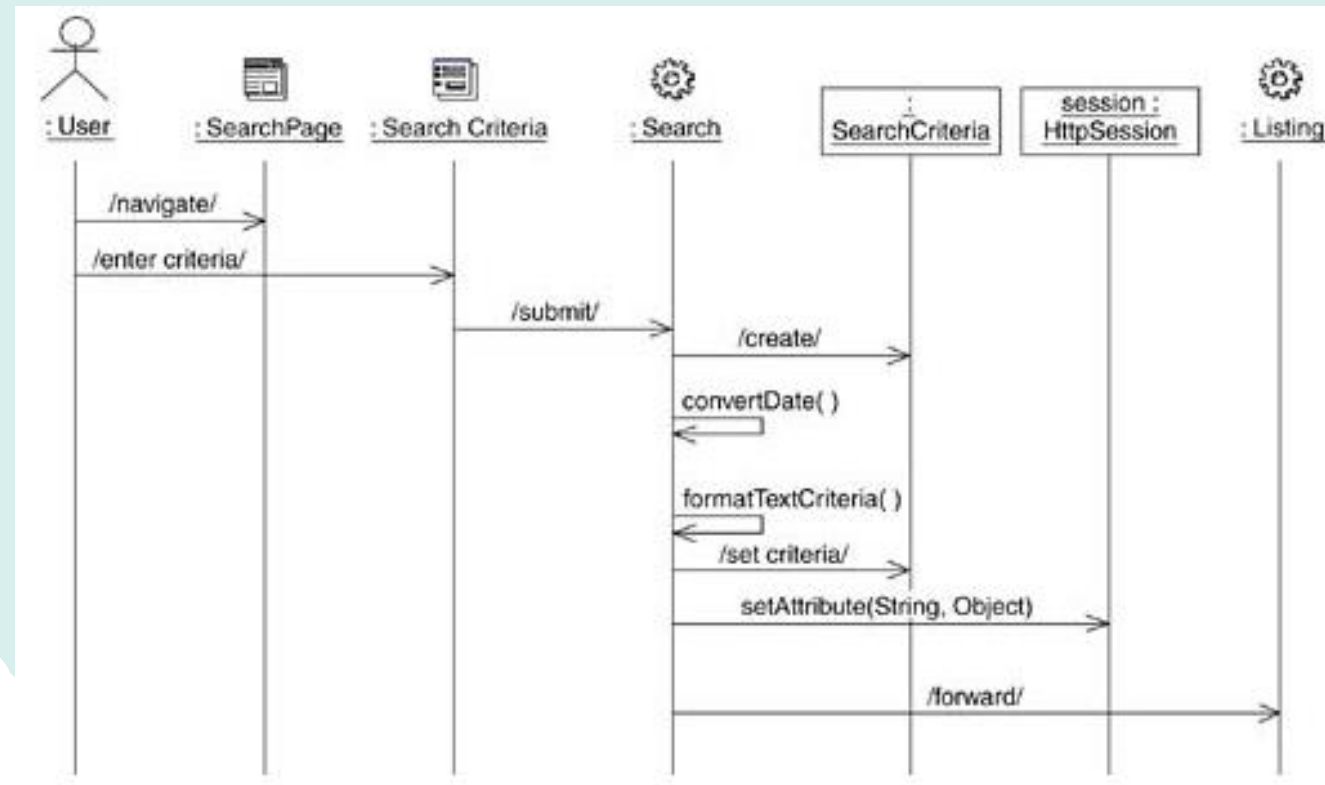
- The search form submits its data to the Search «server page», which is responsible for accepting the search request and processing
- Building the response page is the job of the Listing «server page»
- The SearchResults client page implements a page-scrolling mechanism—Paged Dynamic List, Page-by-Page Iterator—and has two «link» relationships to itself.
 - The parameter tag value for each of these links has a value to indicate which direction to scroll:

{ parameters="scroll=next" }
and

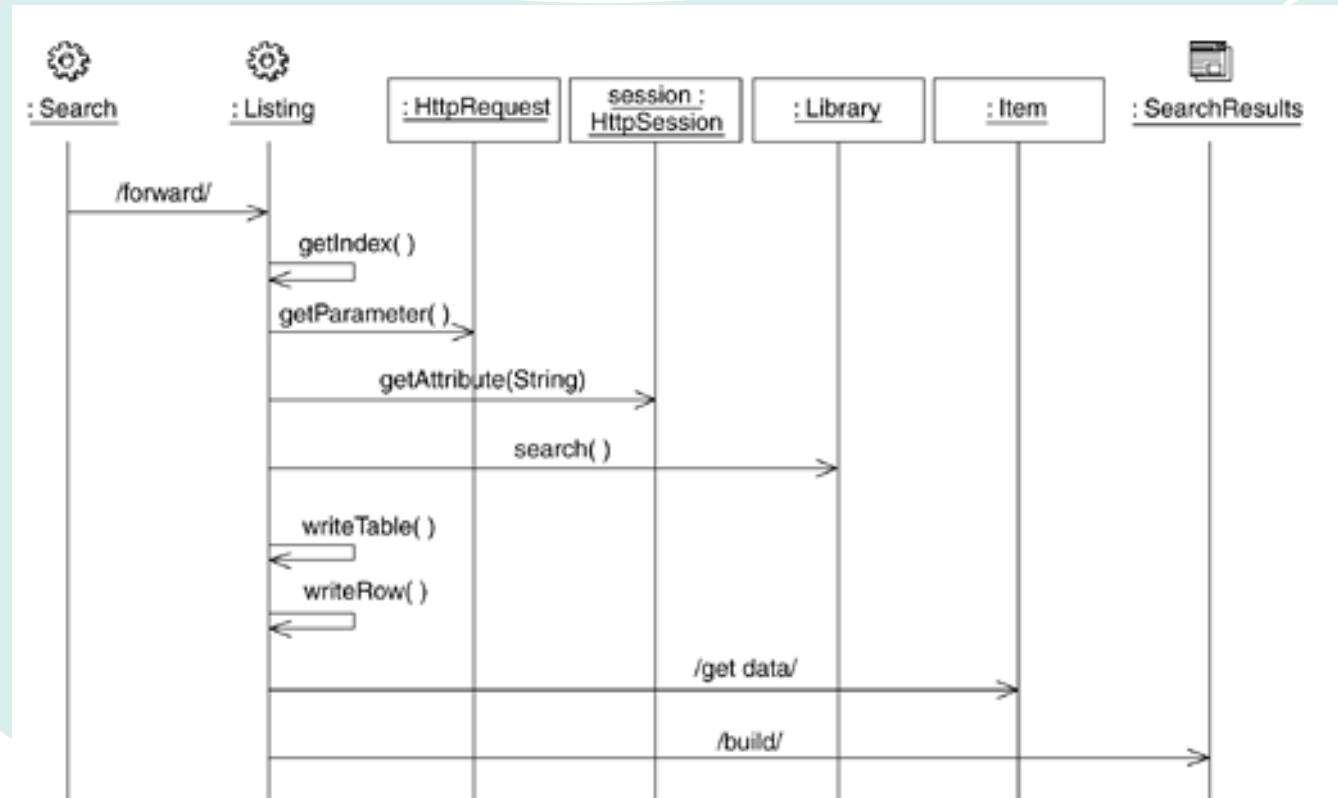
{ parameters="scroll=prev" }.



Search catalog sequence diagram using stereotyped elements



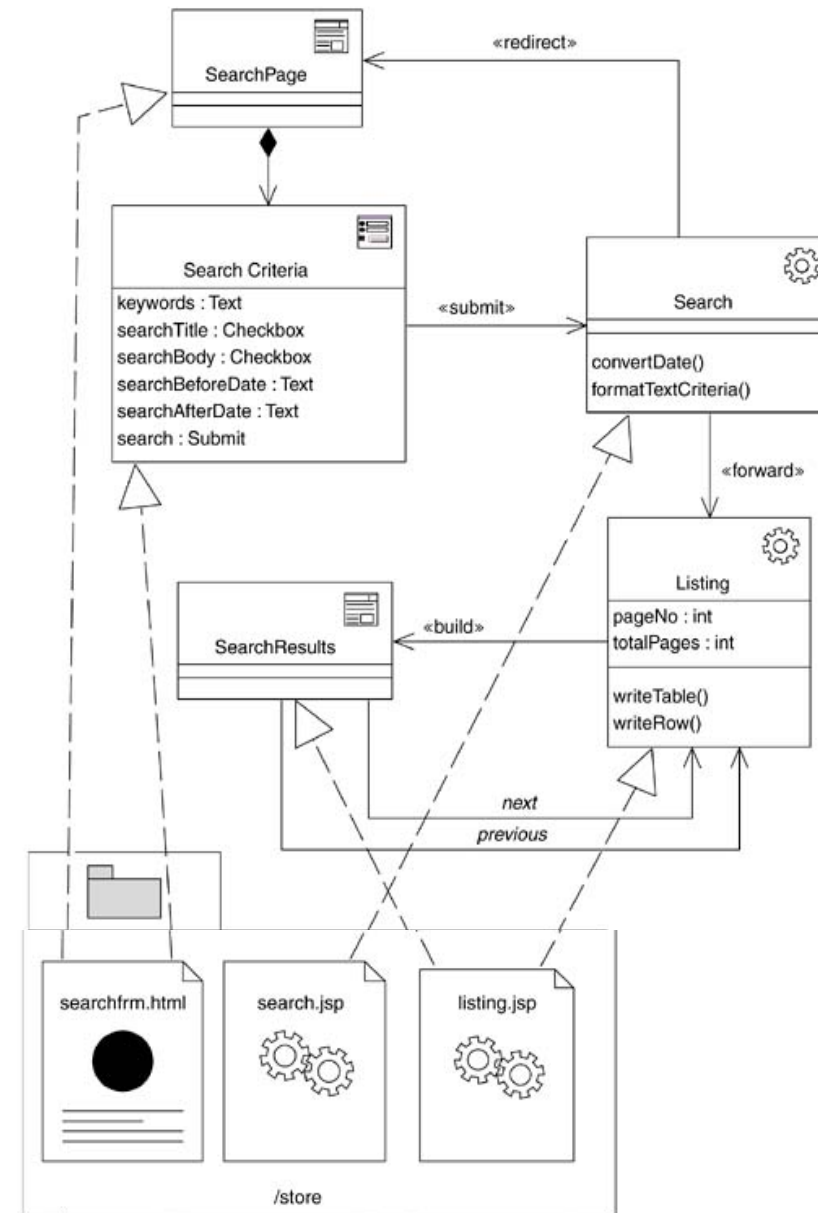
Build Listing sequence diagram



Component view realizations of logical-view classes

- The components are all under the physical root at the same level
- The physical root is named /store
- A client browser would request the search page by using the URL:

<http://localhost/store/searchform.html>.

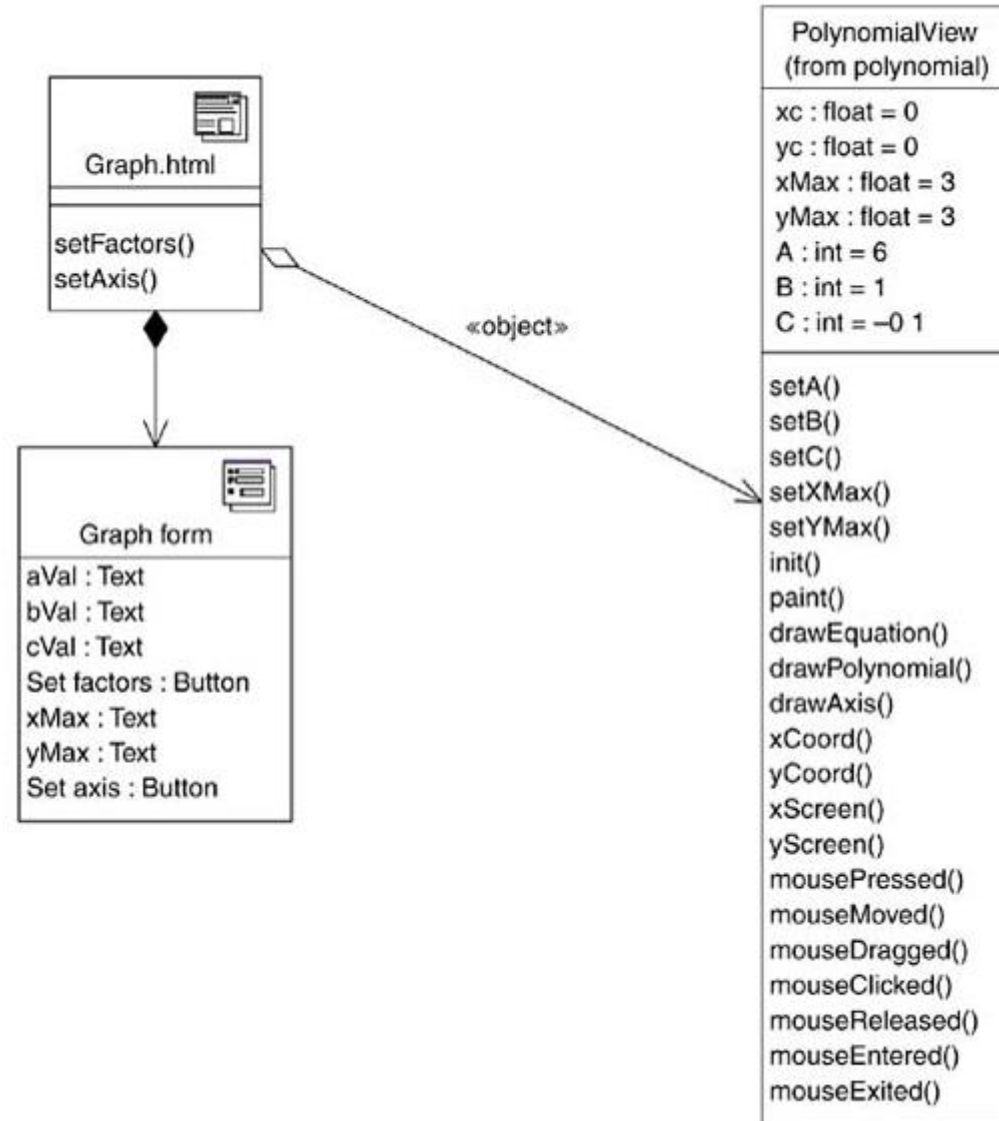


Client-Side Scripting

- requires careful attention to the partitioning of the objects

Modeling applets and other embedded controls

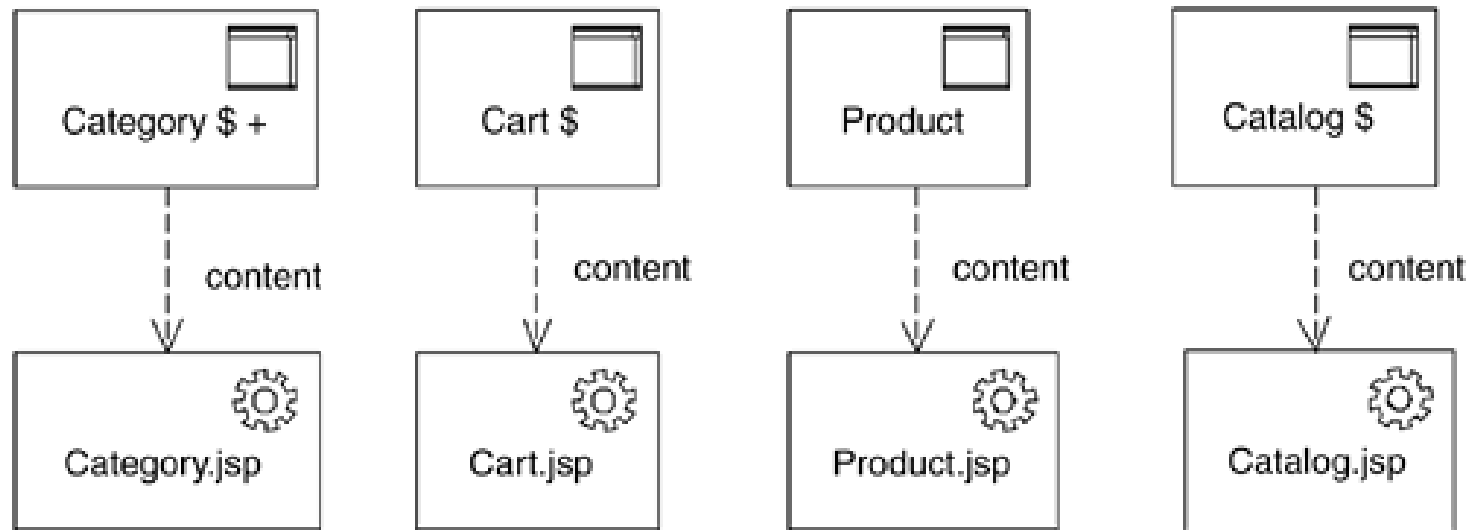
- A simple class diagram with a «client page» that has two defined JavaScript functions.
- Shows an «object» stereotyped association to the PolynomialView applet class.
- The client page also has an HTML form included



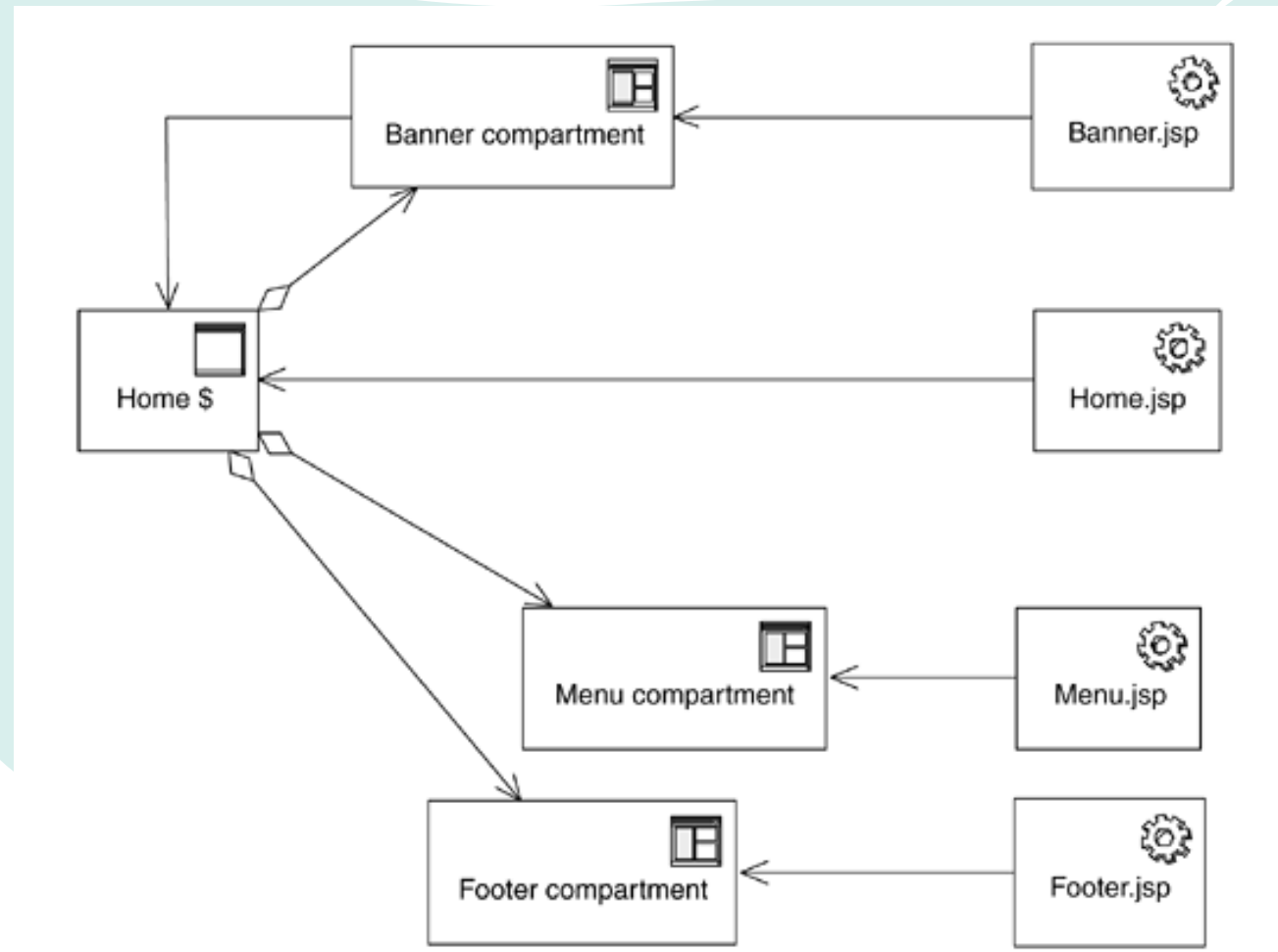
Mapping to the UX Model

- The mappings are captured in class diagrams that contain UX screens and design model classes with dependency relationships connecting them
 - show which Web page classes realize which UX model screens
 - simple architectures have a one-to-one mapping between a Web page class—client or server page—and the screen
 - in more complex architectures, Web page classes are responsible for delivering multiple screens

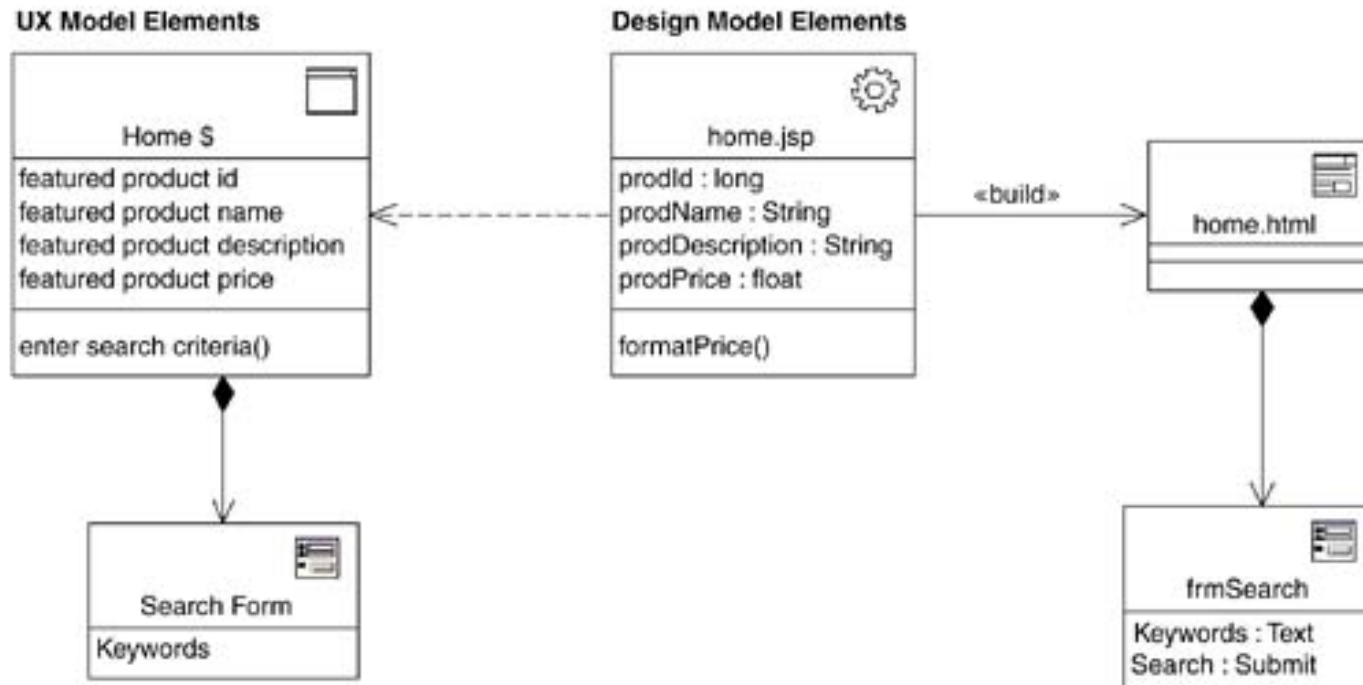
Design model mapping with UX model



Design model mappings with UX model, including «screen compartment» mappings



Design model implementing UX model contract



Guidelines for Web Application Design

- Be wary of using the latest **capabilities of browsers**
 - The browser wars continue, and it is difficult to predict which features will eventually become standard
- Think about how **the page is going to be tested**
 - Don't use visual cues to let the actor know when the page is safe to interact with unless these same cues are accessible by an automated testing tool
- Avoid the temptation to use multiple browser windows simultaneously
 - Although a useful feature for some applications, designing and maintaining two client interfaces requires more than double the effort
- Keep **the focus of server pages on the construction of the user interface**
 - Avoid placing business logic code in the server page. Use external objects to encapsulate this type of logic



Referensi

- **Building Web Applications with UML Second Edition** By Jim Conallen
- Publisher : Addison Wesley
- Pub Date : October 04, 2002

