Tapestry Design Review

Thu Apr 4 2000

Introduction

- ✔ We’re still in the dark ages of web applications
- ✔ We’re told:
  - Receive a request
  - Process it
  - Send a reply
- ✔ That’s a lie!

Introduction

- ✔ Web applications aren’t about requests
  - – They’re about interactivity
  - – Responding to user in a custom way
  - – Unifying behavior throughout application
- ✔ Most solutions focus on the request alone
  - – ASP, JSP, WebMacro, XMLC, FreeMarker
  - – Scripting languages
  - – Unique, ugly, incompatible

Introduction

- ✔ Scripting only helps for one part of interaction
  - Each page includes "potentials" … links and forms with some behavior
  - To build one page, you need to know how to invoke actions, some on other pages
  - Tricky URLs to encode action, parameters, etc.
  - Scripting only knows URLs, not actions
  - Different developers, different approaches

Introduction

- ✔ Scripting too weak
  - – Too many easy mistakes
  - – Too hard to debug
  - – Too much user-written code
  - – HTML full of “wierdness”
  - – Little possibility for re-use (non-static)
  - – Lots of plumbing for little effect

Introduction

- ✔CGI very procedural
- ✔Start here, do this, stop
- ✔APIs based on CGI very procedural
- ✔Time for objects!

Introduction

- ✔Thus, Tapestry!
- ✔Build app from component objects
- ✔Let framework do “the plumbing”
- ✔Reduce amount of code
- ✔Increase amount of interactivity
- ✔Eliminate bugs from ad-hoc solutions

Tapestry Goals

- ✔ Portable code (JDK 1.1, Servlets 2.1)
- ✔ Minimal HTML markup
- ✔ Tapestry handles building/parsing URLs
- ✔ Make difficult easy:
  - Debugging
  - Deployment
  - Localization
  - Reuse
  - Monitoring / performance analysis
- ✔ Robust exception support

Tapestry Goals

- ✔Reduce amount of Java code
- ✔Zero code generation
- ✔Work well for failover, load balancing
- ✔Good interfaces, simple implementations
  - Grow Tapestry by creating new implementations
  - Existing code works well into the future

Overview

- ✔ Components
- ✔ Parameters
- ✔ Pages
- ✔ Applications
- ✔ Application Servlet
- ✔ Request Cycle
- ✔ Application Services
- ✔ Persistent Page State
- ✔ Dynamic Page State

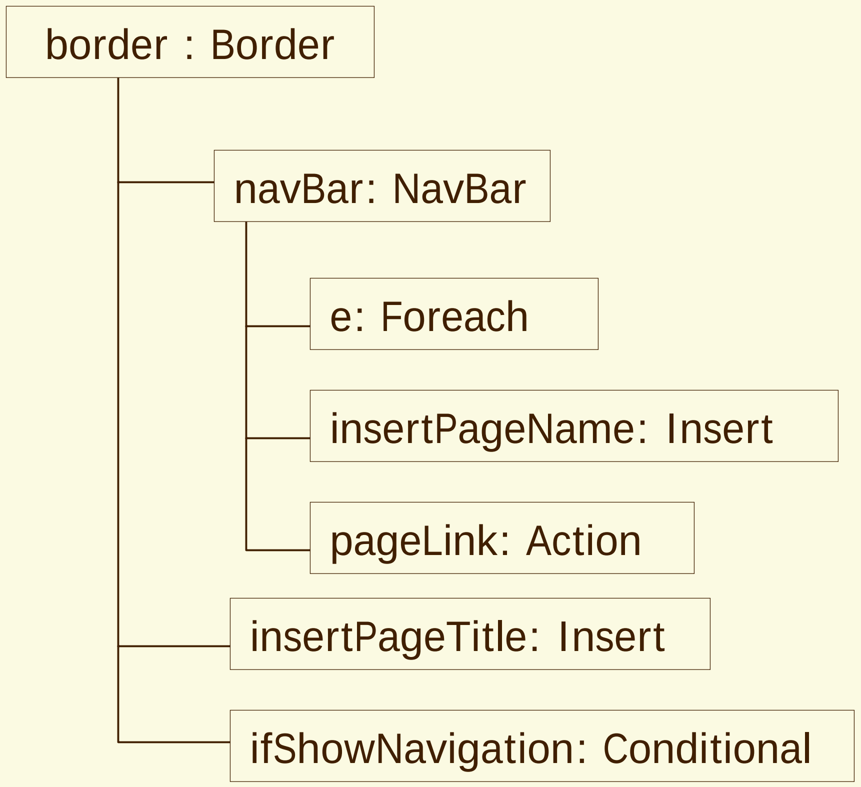
Components

- ✔ Tapestry component
  - – Specification
  - – HTML template (optional)
  - – Java Class (usually, not always)
- ✔ Component parameters
  - – Define data needed by component
  - – Ex: insert component has ‘value’
  - – Dynamic: based on JavaBeans properties

Components

- ✔Recursive
  - – Components contain other components
  - – Aggregation
  - – Part of the component specification
  - – Arbitrary depth
  - – Ex: ShoppingCartEditor contains form, textfield, insert, foreach, conditional, ...

Components


border : Border
navBar: NavBar
e: Foreach
insertPageName: Insert
pageLink: Action
insertPageTitle: Insert
ifShowNavigation: Conditional

Components

- ✔ Re-usable
  - Parameters to adapt component to page and application
  - “Black box” design

Parameters

- ✔ Parameters are “plugs” in the Component black-box
- ✔ Mostly, data “pulled” into Component
  - Insert: value to insert into HTML
  - Conditional: value to evaluate
  - Action: listener to notify if action triggered
  - form components: initial value for form fields

Parameters

- ✔ Sometimes, data “pushed” out from Component
  - – Foreach: current value
  - – form components: updated value when form submitted

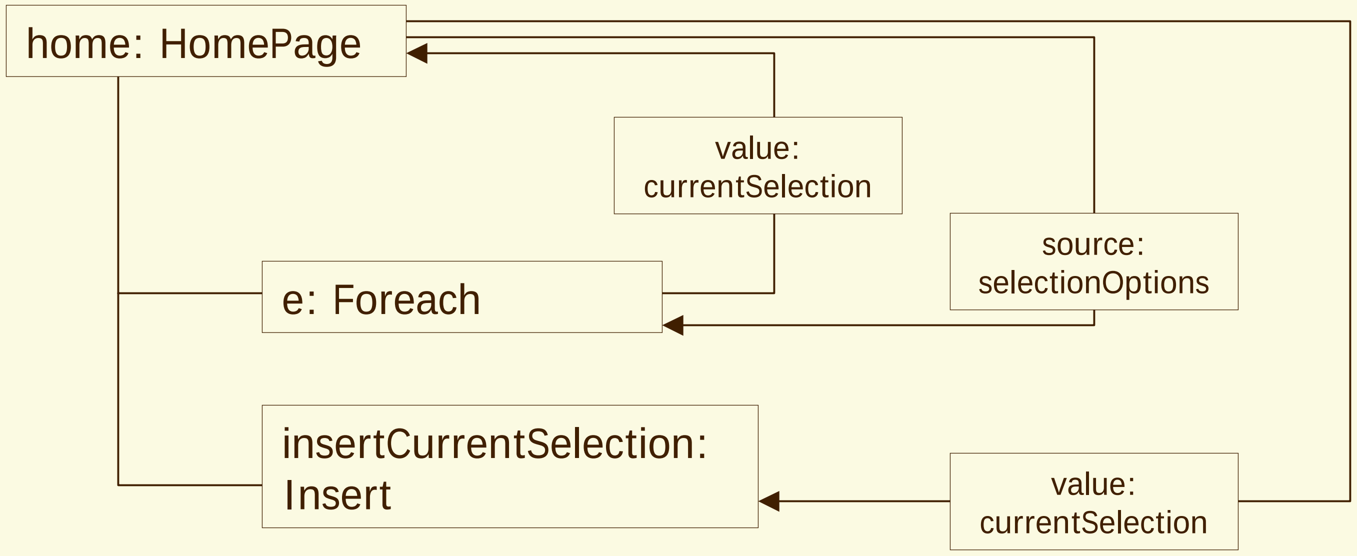
Parameters

- ✔ Bindings provide values for parameters
- ✔ Specified in containing Component's specification
- ✔ Static bindings
  - – Fixed string value
  - – Often coerced to int or bool
  - – read-only

Parameters

- ✔ Dynamic bindings
  - Specifies JavaBeans property to get or set value
  - Relative to containing Component
  - read, write, read/write -- if Component implements accessors, mutators
  - Can use property path
    - Ex: page.application.specification.name

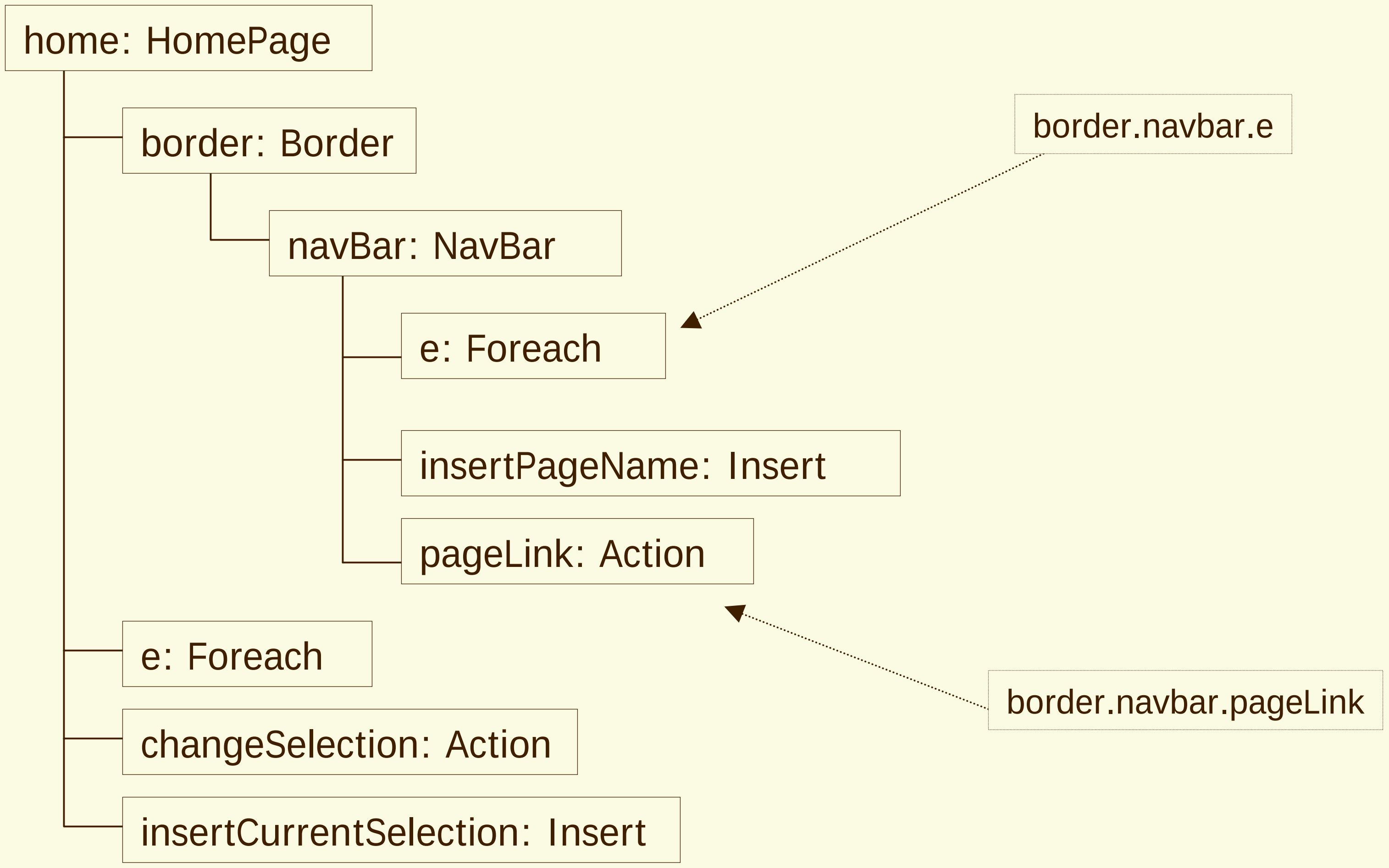
Parameters


home: HomePage
value: currentSelection
source: selectionOptions
e: Foreach
insertCurrentSelection: Insert
value: currentSelection

Pages

- ✔Specialization of Component
- ✔Point of interaction with application
- ✔Focus for persistence of server-side state
- ✔Contain other components
- ✔Specific Locale for localization
- ✔No parameters - no containing Component

Pages


home: HomePage
border: Border
navBar: NavBar
e: Foreach
insertPageName: Insert
pageLink: Action
e: Foreach
changeSelection: Action
insertCurrentSelection: Insert
border.navbar.e
border.navbar.pageLink

Applications

- ✔ Provide support to everything else
  - Page recorder for each page
  - Page source: pool of reusable pages
  - Application services (for building URLs)
  - Runs the request cycle
  - Access to templates & specifications
- ✔ One instance for each client, stored in HttpSession

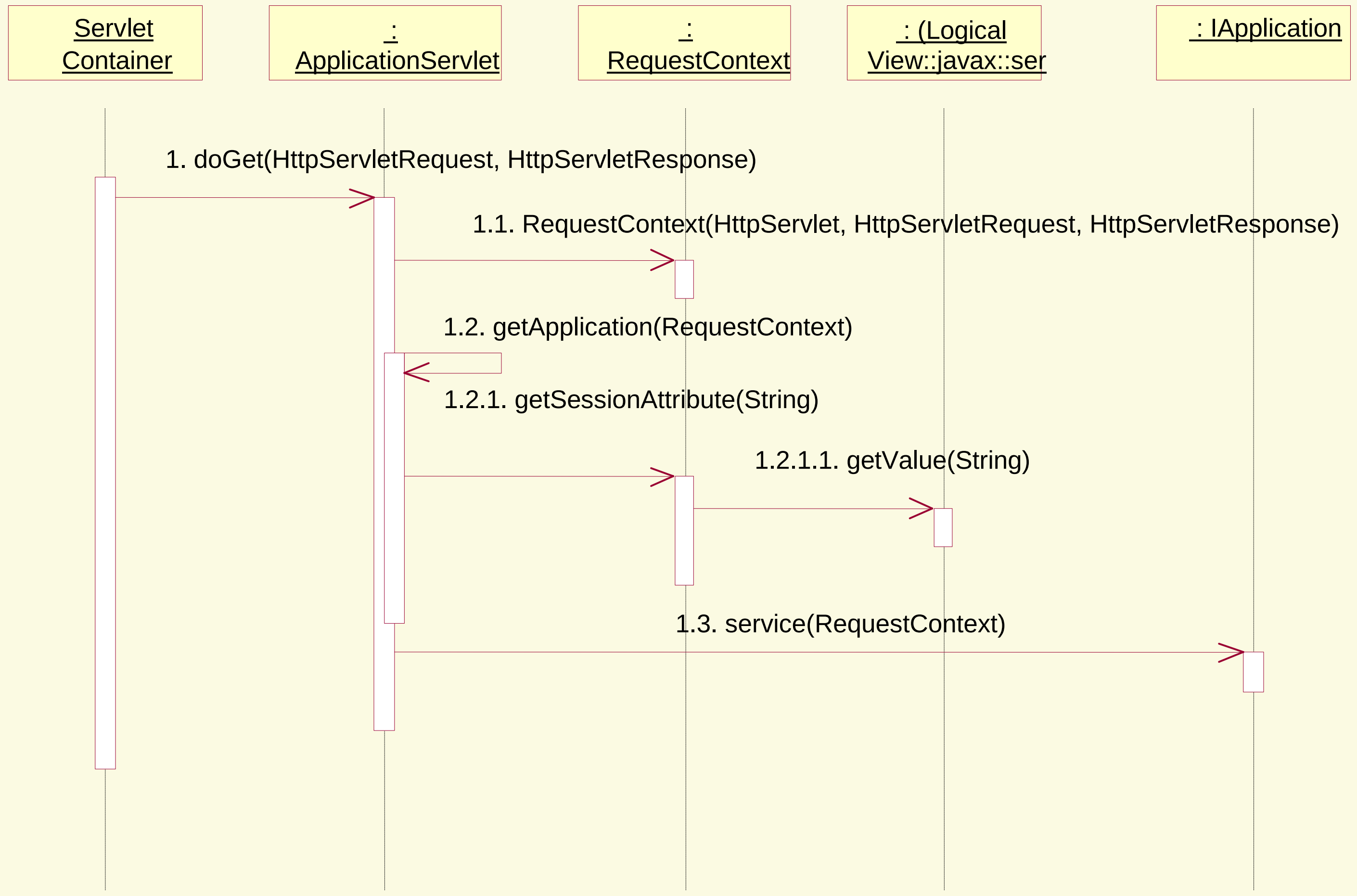
Applications

- ✔Central location for common values and logic
- ✔Serializable: may move to a different JVM because of failover or load balancing
- ✔Maps page names to page components
- ✔Provides framework for handling exceptions
- ✔Can provide new services

Application Servlet

- ✔ Single servlet for entire application
- ✔ Very little code: locates the Application object in the HttpSession, or creates it
- ✔ Delegates everything else to the Application

Application Servlet


Servlet Container
: ApplicationServlet
: RequestContext
: (Logical View::javax::ser
: IApplication
1. doGet(HttpServletRequest, HttpServletResponse)
1.1. RequestContext(HttpServlet, HttpServletRequest, HttpServletResponse)
1.2. getApplication(RequestContext)
1.2.1. getSessionAttribute(String)
1.2.1.1. getValue(String)
1.3. service(RequestContext)

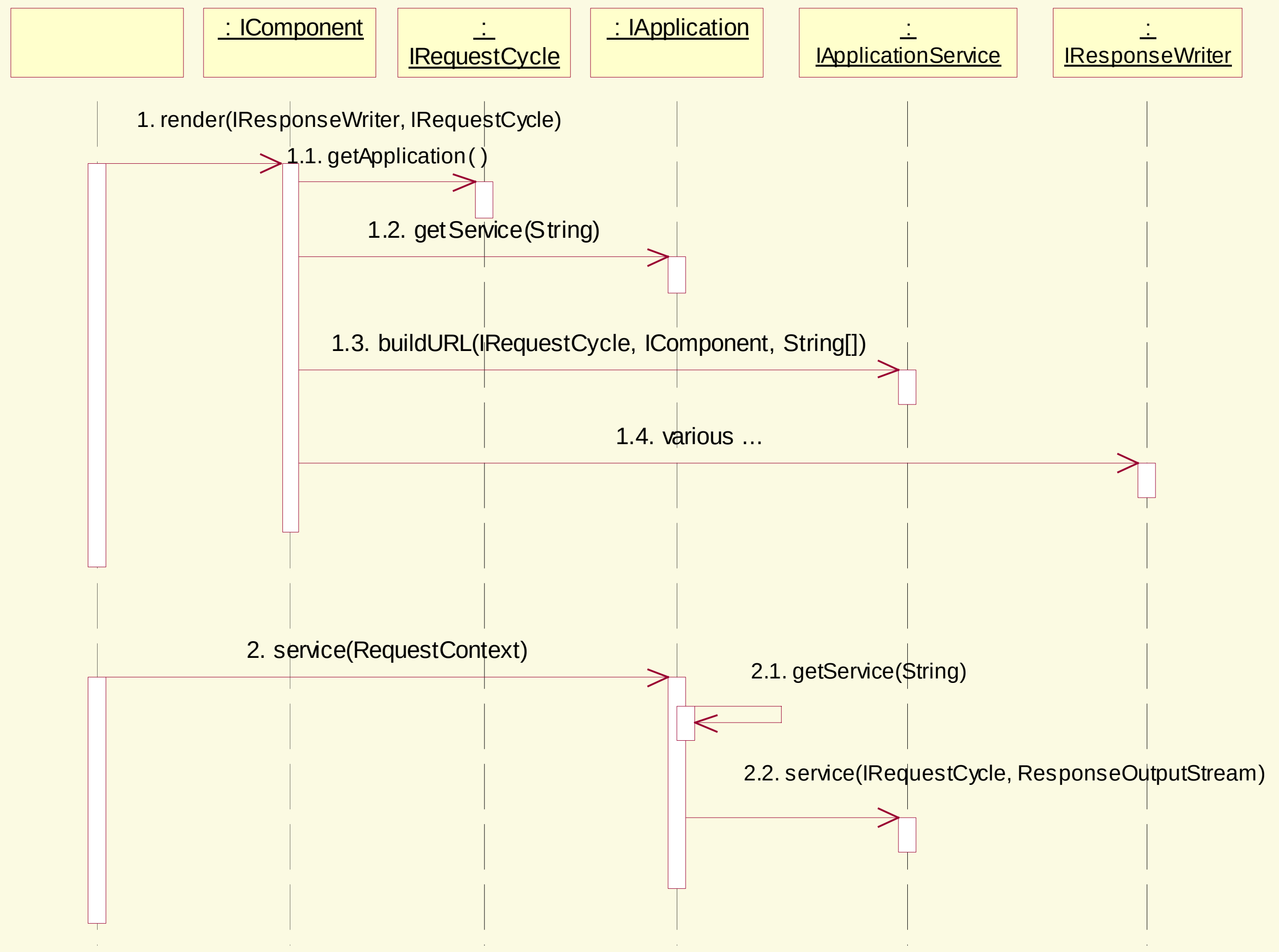
Request Cycle

- ✔Represents processing a single request and rendering a response HTML page
- ✔Tracks state of components
- ✔Knows ‘where on the page’ during renderring … needed to build URLs

Application Service

- ✔Builds URLs for components
- ✔Later, parses URL and kicks off request cycle
- ✔URLS:
  - *servlet path / service name / service info*
  - Each service defines its own service info
  - Ex: Page service, info is name of page
- ✔Usually linked to a specific Component

Application Service


: IComponent
: IRequestCycle
: IApplication
: IApplicationService
: IResponseWriter
1. render(IResponseWriter, IRequestCycle)
1.1. getApplication( )
1.2. getService(String)
1.3. buildURL(IRequestCycle, IComponent, String[])
1.4. various ...
2. service(RequestContext)
2.1. getService(String)
2.2. service(IRequestCycle, ResponseOutputStream)

Persistent Page State

- ✔ Pages have server-side state
- ✔ Data specific to state
- ✔ Lifespan is same as session
- ✔ Examples:
  - – Show / hide details
  - – Form data, or errors in submitted form
  - – Navigation through complicated data
  - – Handles of EJBs, database connections, etc.
- ✔ Rich state for rich interaction

Persistent Page State

- ✔ Traditionally (Servlets, JSPs)
  - – A lot of “plumbing”
  - – Stored as session attributes, cookies, hidden form fields, encoded URLs, etc.
  - – Lots of ‘ad-hoc’, buggy solutions
  - – Life span of data hard to control

Persistent Page State

- ✔ Stored in instance variables of page
- ✔ Problem:
  - – Pages are complicated to build … whole tree of components, bindings, templates …
  - – Pages are pooled between requests
  - – Pooled pages shared between client sessions
  - – Need to separate page state from instances of pages

Persistent Page State

- ✔Page Recorders
- ✔Notified of changes to persistent properties

```
public void setActiveSelection(String value)
{
    activeSelection = value;
    fireObservedChange("activeSelection", value);
}
```

- ✔Can rollback a page to a prior state
- ✔Components can use their page’s recorder

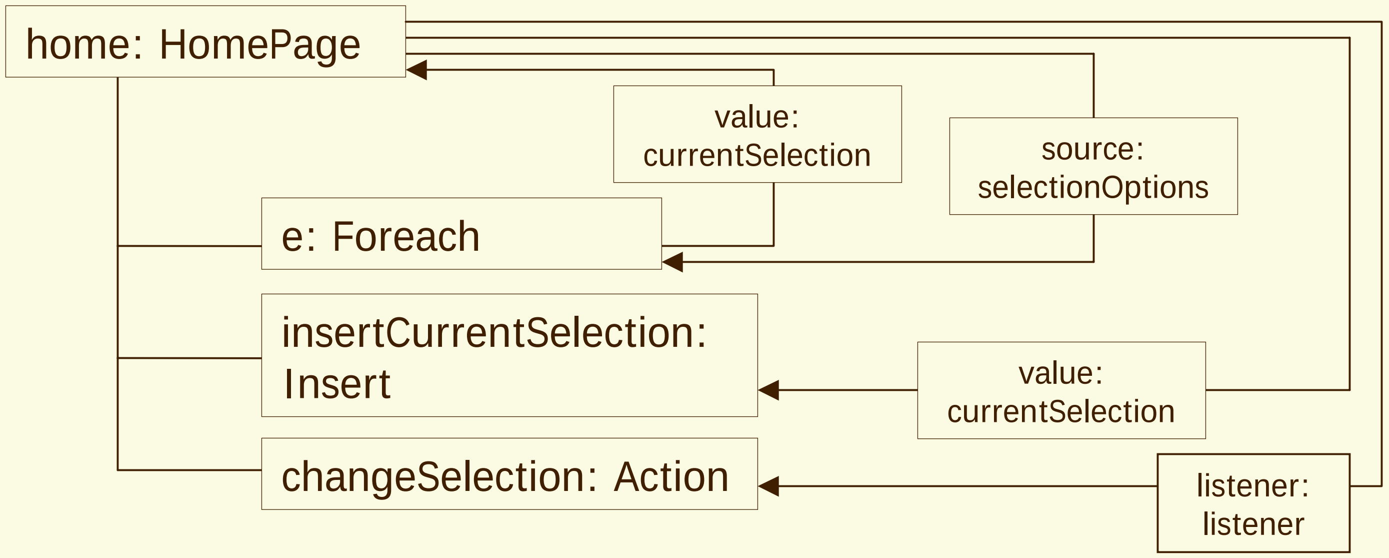
Persistent Page State

- ✔Page Recorders have a version number
- ✔Incremented every request cycle if a property changes
- ✔Incorporated into URLs
- ✔Used to identify “stale links”

Dynamic Page State

✔ State that changes during renderring a page
✔ Examples:
  – Iterating a list of line items in a shopping cart
  – Building an option list of answers in a survey
✔ Same components used multiple times in same render
✔ Parameters different each time

Dynamic Page State



- ✔ Problem: actions
  - Knowing component id not enough
  - What is dynamic state (ex: currentSelection)?

Dynamic Page State

- ✔ Solution: encode in URL info needed to restore dynamic state
- ✔ How?
  - – Allocate *action ids* during render
  - – Simple ascending sequence
  - – Automatically accounts for Foreach, Conditional, etc.

Dynamic Page State

- ✔ Restoring page state:
  - – Roll back page state
  - – Re-render page, discarding output
  - – Action ids allocated again
  - – When current action id matches encoded action id, state has been restored
  - – Action component invokes actionTriggered() on its listener
  - – Called “rewind stage”

In Detail

- ✔ Understanding the Request Cycle
- ✔ Application Services
  - page
  - direct
  - action
- ✔ HTML Templates
- ✔ Component Specification
- ✔ Application Specification

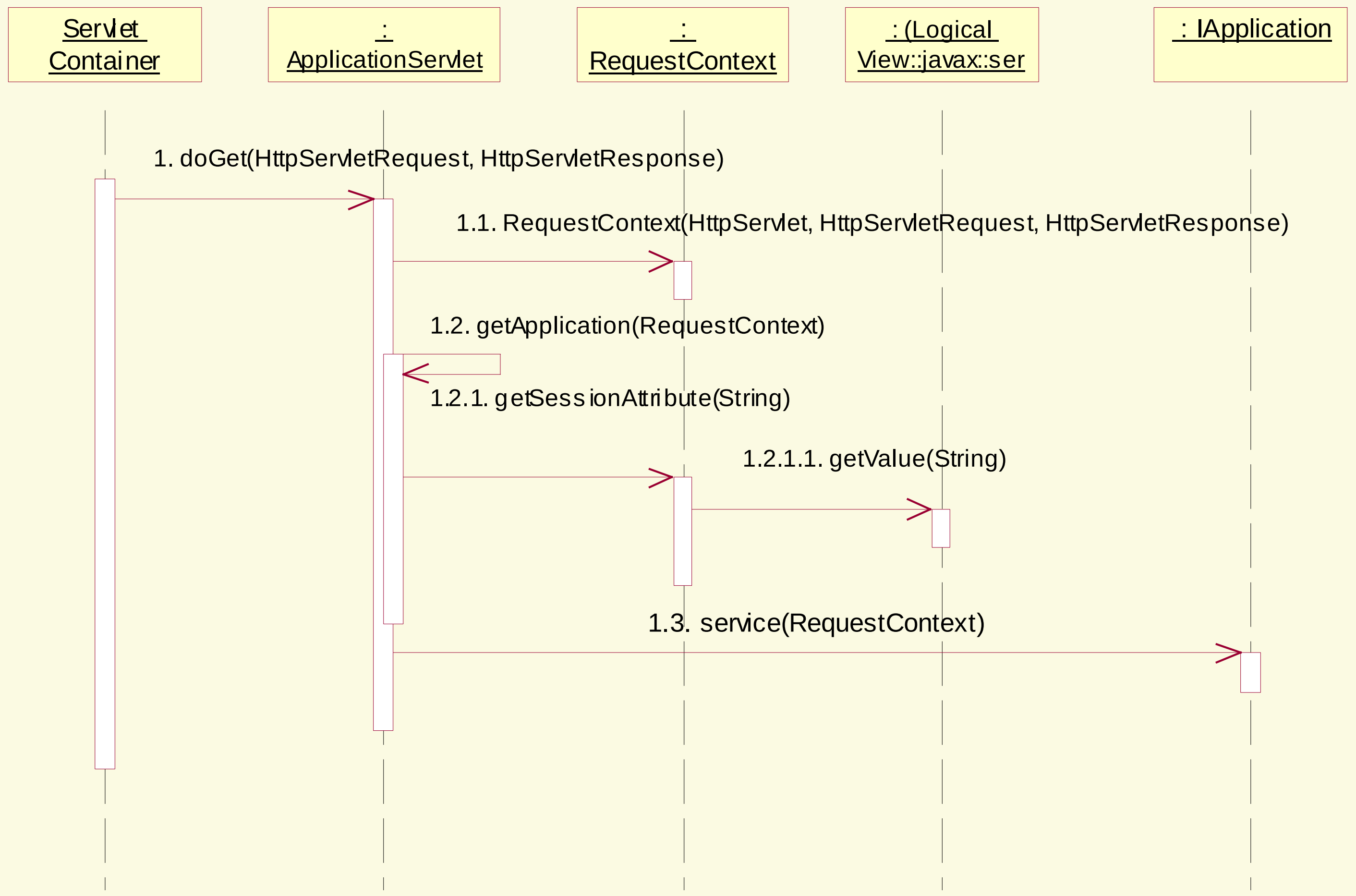
Understanding the Request Cycle

- ✔ Servlet Container invokes Application Servlet
- ✔ Servlet locates/creates Application object, invokes service()
- ✔ Application digs service out of URL
- ✔ Application finds correct Application Service, invokes service()

Understanding the Request Cycle

- ✔ Service runs rest of request cycle
- ✔ Generally:
  - Load page, restore its state
  - Find component identified in URL
  - Invoke component's listener's actionTriggered()
  - Render a result page

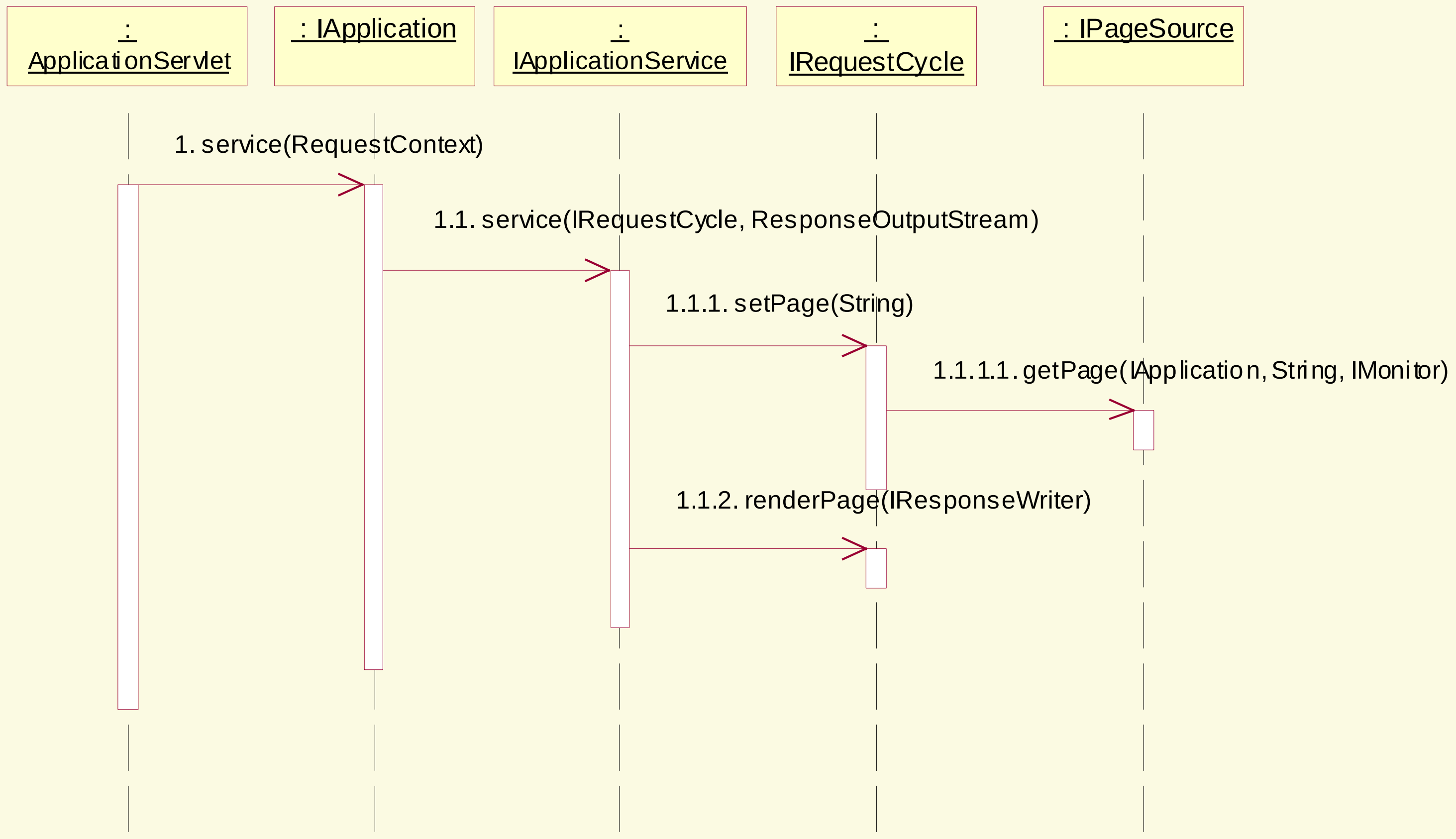
Understanding the Request Cycle


Servlet Container
: ApplicationServlet
: RequestContext
: (Logical View::javax::ser
: IApplication
1. doGet(HttpServletRequest, HttpServletResponse)
1.1. RequestContext(HttpServlet, HttpServletRequest, HttpServletResponse)
1.2. getApplication(RequestContext)
1.2.1. getSessionAttribute(String)
1.2.1.1. getValue(String)
1.3. service(RequestContext)

Application Service: page

- ✔ Simple, used for basic navigation
- ✔ URL:
  servlet path / **page** / *page name*
- ✔ Restores page's persistent state
- ✔ Doesn't need to find component, just renders response page

Application Service: page

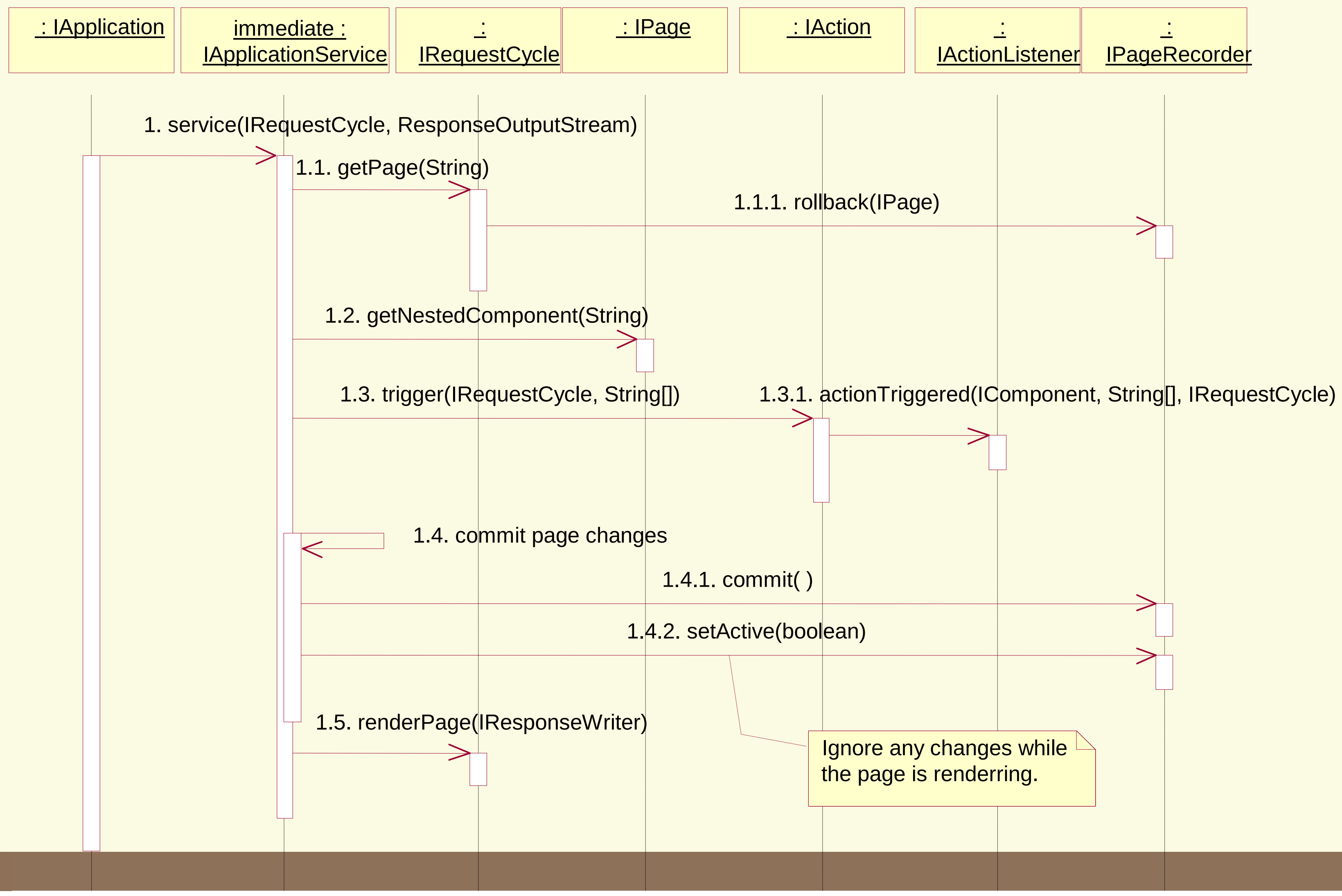

: ApplicationServlet
: IApplication
: IApplicationService
: IRequestCycle
: IPageSource
1. service(RequestContext)
1.1. service(IRequestCycle, ResponseOutputStream)
1.1.1. setPage(String)
1.1.1.1. getPage(IApplication, String, IMonitor)
1.1.2. renderPage(IResponseWriter)

Application Service: direct

- ✔For links & buttons on page that don't rely on dynamic state of page
- ✔URL:

  servlet path / **direct** / *page name* / *page version* / *component id path* / *additional parameters*
- ✔Invokes trigger() on the component
- ✔Component invokes actionTriggered() on its listener
- ✔Can carry additional parameters in URL

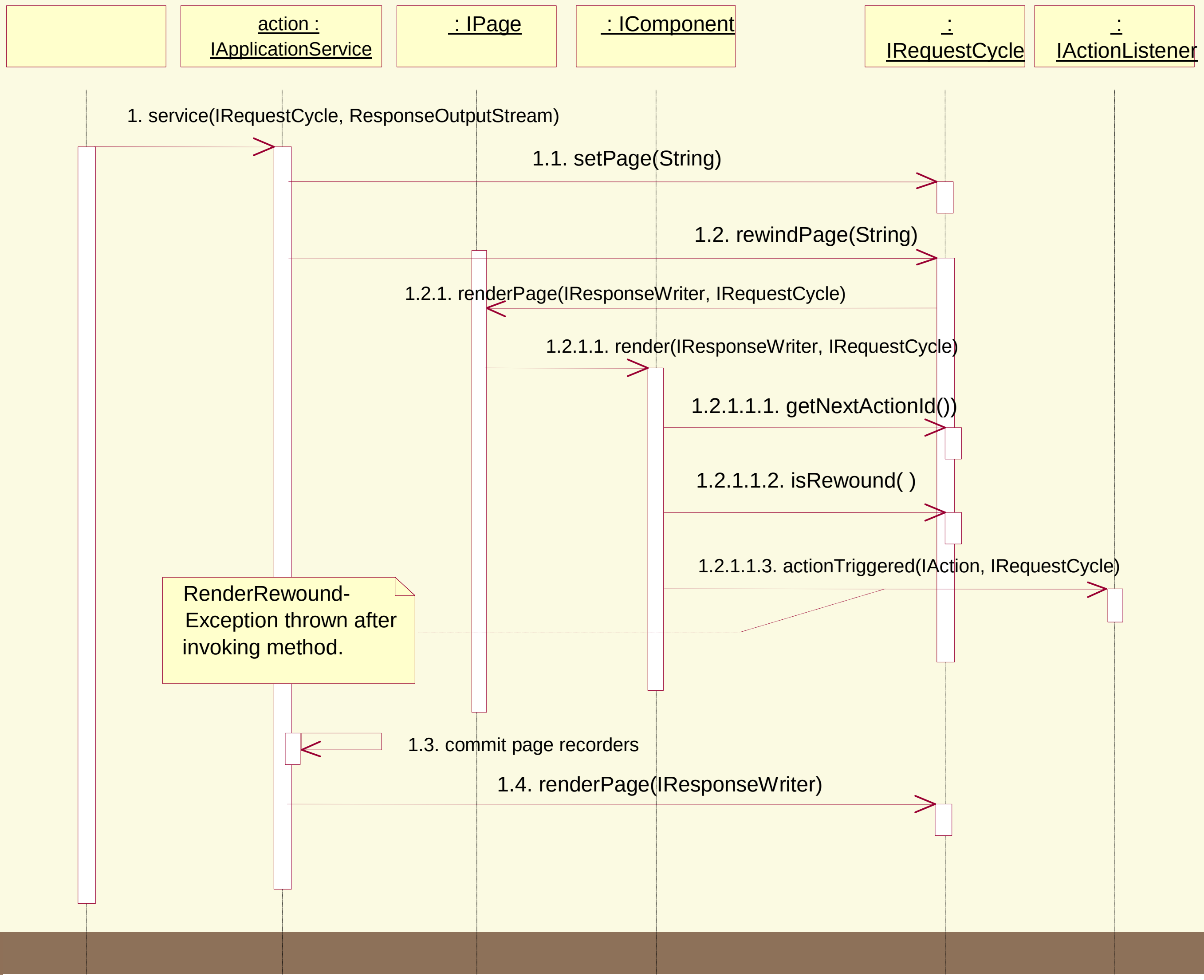
Application Service: direct


: IApplication
immediate : IApplicationService
: IRequestCycle
: IPage
: IAction
: IActionListener
: IPageRecorder
1. service(IRequestCycle, ResponseOutputStream)
1.1. getPage(String)
1.1.1. rollback(IPage)
1.2. getNestedComponent(String)
1.3. trigger(IRequestCycle, String[])
1.3.1. actionTriggered(IComponent, String[], IRequestCycle)
1.4. commit page changes
1.4.1. commit( )
1.4.2. setActive(boolean)
1.5. renderPage(IResponseWriter)
Ignore any changes while the page is renderring.

Application Service: action

- ✔Used with forms and with actions sensitive to dynamic page state
- ✔URL:
  servlet path / **action** / *page name* / *page version* / *action id*
- ✔Rolls the page back, then rewinds it
- ✔The action component invokes actionTriggered() on its listener

Application Service: action


action : IApplicationService
: IPage
: IComponent
: IRequestCycle
: IActionListener
1. service(IRequestCycle, ResponseOutputStream)
1.1. setPage(String)
1.2. rewindPage(String)
1.2.1. renderPage(IResponseWriter, IRequestCycle)
1.2.1.1. render(IResponseWriter, IRequestCycle)
1.2.1.1.1. getNextActionId())
1.2.1.1.2. isRewound( )
1.2.1.1.3. actionTriggered(IAction, IRequestCycle)
RenderRewound-Exception thrown after invoking method.
1.3. commit page recorders
1.4. renderPage(IResponseWriter)

HTML Templates

- ✔Goals:
  - – Easy localization
  - – Minimal HTML markup
  - – Efficiency
- ✔Each component has a single template
- ✔Template may be chosen based on Locale
- ✔Templates are resources packaged in the JAR with classes and specifications

HTML Templates

- ✔Templates are excerpts of standard HTML documents
- ✔Add a single element:
  <jwc>
  Java **W**eb **C**omponent
- ✔Just a placeholder for the location of of the component

HTML Templates

- ✔ Usage:
  - <jwc id="*component id*"> … </jwc>
  - <jwc id="*component id*"/>
- ✔ Components may *wrap* static HTML and other components
- ✔ Affects the order in which components are renderred (the component trace)

HTML Templates

HomePage.html

```
<p>Change it to:
<ul>
<jwc id="e">
  <li><jwc id="changeSelection">
        <jwc id="insertCurrentSelection"/>
      </jwc>
</jwc>
</ul>
```

HTML Templates

- ✔Each component id in the template matches against a contained component in the specification
- ✔Pages have templates, but are not contained inside other components
- ✔Pages do contain other components

Component Specification

- ✔ Specifications are XML files
- ✔ Stored with class in JAR file
- ✔ Describe type of component
- ✔ Describe parameters of component
- ✔ Describe components contained within component

Component Specification

✔ Structure:

```
<specification>
    <class>class name</class>
    parameters
    components
    assets
</specification>
```

Component Specification

- ✔Class is the Java Class to instantiate
- ✔Classes which use a template don’t specify the path to the template … it is assumed to be a neighbor of the specification (with the .jwc extension changed to .html).

Component Specification

✔ *Parameters*

```
<allow-body>boolean</allow-body>
<allow-informal-parameters>boolean
    </allow-informal-parameters>

<parameter>
    <name>name</name>
    <type>type</type>
    <required>boolean</required>
</parameter>
```

Component Specification

- ✔ <allow-body>
  - – Some components may not wrap other elements
  - – ex: textfield, insert
  - – Defaults to true if not specified

Component Specification

✔ <allow-informal-parameters>
- Allows additional bindings beyond the defined parameters
- Each becomes one attribute of the tag created by the component
- Only makes sense when the component maps directly to a single HTML element
- Used for JavaScript, CSS
- Defaults to true if not specified

Component Specification

- ✔ <parameter>
  - – One for each parameter
  - – Name should be a valid JavaBeans property name (alphanumeric)
  - – Type is a class, or omit for Object (I.e., match any)
  - – Required defaults to false; if true then binding must be specified

Component Specification

✔Embedded components:

```
<component>
  <id>id</id>
  <type>type</type>
  bindings
</component>
```

✔Id must match <jwc> tag in template

✔Type is either a specification path, or a well known alias

✔Built in components all have aliases

Component Specification

- ✔ Components have bindings that match their parameters
- ✔ Three types:
  - dynamic
  - static
  - inherited

Component Specification

- ✔ Dynamic binding:

```
<binding>
    <name>name</name>
    <property-path>property path
   </property-path>
</binding>
```

- ✔ Property path is relative to containing component
- ✔ May be read, write or read/write depending on component

Component Specification

- ✔Static binding:

```
<static-binding>
    <name>*name*</name>
    <value>*value*</value>
</static-binding>
```

- ✔Value will be the String value for the parameter
- ✔Read-only
- ✔Often coerced to int or boolean

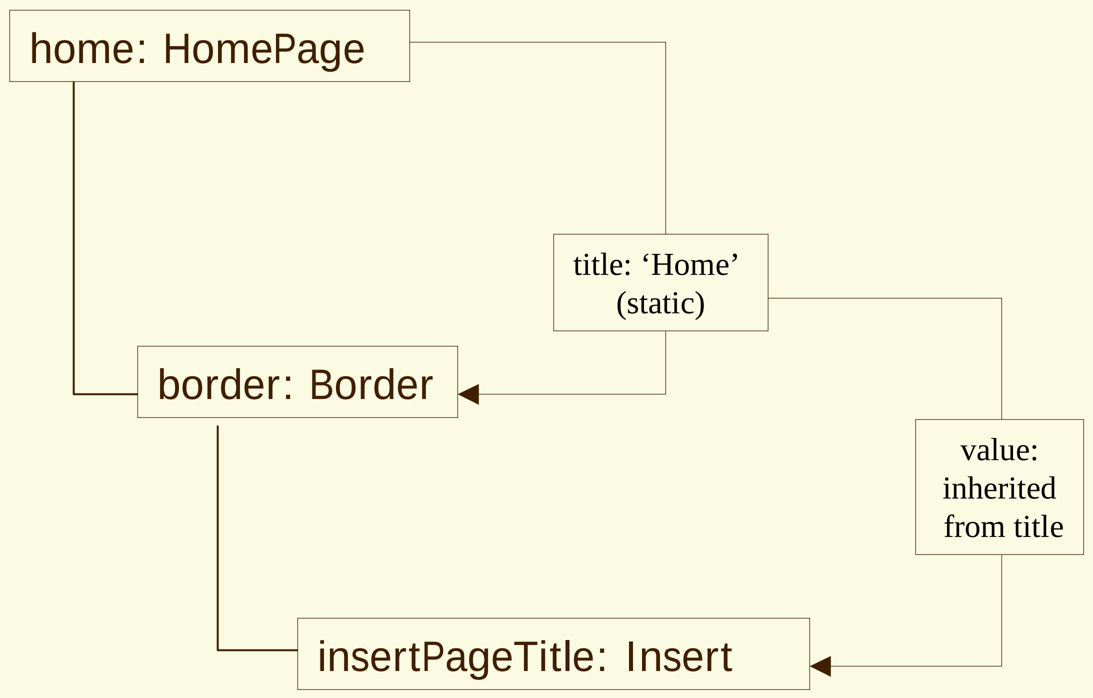
Component Specification

- ✔Inherited binding:

```
<inherited-binding>
    <name>name</name>
    <parameter-name>parameter name
      </parameter-name>
</inherited-binding>
```

- ✔Used often with aggregation
- ✔Contained component shares a parameter with its container

Component Specification



Component Specification

- ✔ Assets
  - – Allow images, sounds, etc. to be packaged with a component
  - – Supports re-use
  - – Described elsewhere

Application Specification

- ✔ Another XML file
- ✔ Specifies
  - – Name of application
  - – Map from page name to page component
  - – Short aliases for common components
    - ex: ‘NavBar’ instead of ‘/tests/tapestry/NavBar.jwc’