UI Design and Development, a Case Study

Bryan Basham
StillSecure, Inc.
b_basham@yahoo.com
Final Edits

The slides you see here are edited to included content from the three sessions at the Colorado Software Summit 2007. The slides with titles in this orange-brown color have been added.

In some cases, the new diagrams represent additional design relationships that were explored during the presentation.

In other cases, content has been added or restructured from bullet-point slides to mind-maps. These mind-maps can be used to illustrate changing information as time progresses. In the case of this talk, most of these mind-maps are evolved from each of the four phases that the Cobia project is transforming into.
Other Presentations

- Choosing a JVM Web Framework
  - by Matt Raible
- JavaServer Faces
  - by Gary Murphy
- Extending JSF
  - by Bryan Basham
- Introduction to Grails
  - by Scott Davis
Goals of the Presentation

The goals of this presentation are roughly two-fold:

- Describe my experience with integrating a UI design and development team
- Solicit the experience of the audience on the topics raised during each session
The Cobia project has two very high-level goals. First, the UI must be user-focused and as a consequence have a rich UI experience. In phases one and two, these goals were managed by the Customer Experience group (the UI designers). The second goals is that Cobia architecture must promote 3\textsuperscript{rd}-party module development and as a consequence Cobia must have a software development kit (SDK). The SDK is managed by the Engineering group.
User-Driven Development

- User-focused
- Observe use of current system
- Pain points of current system
- User interviews
- Make developers use the system
- Usability testing
- Prototyping
- Use case simulations
- Tools
- "Click stream" part of Open Symphony
- JTA
- Domain knowledge
- Business processes and rules
- User cases
- Requirements
- User-driven development

Engage real users: wants and needs
Engage users to test the system

UDD (User-Driven Development)
Topics Mind Map

The Cobia Project

UI Design & Development

Phase One

Phase Two

Phase Three

Phase Four
What is Cobia?

- Cobia is a platform for modules:

```
Admin  Firewall  Router  DHCP  IPS (StrataGuard)  WiFi  ...
```

- Structure of modules:

```
Modules have Managers
  perform
  maintain state

Jobs  Model
```
What is Cobia?

- Tiers of the module design:
  - Cobia Appliance
  - Domain Model
  - Manager
  - Service
  - Meta Data
  - Database
  - Config file
  - UI
What is Cobia?

- Tiers again...

- User interacts with the Screens
- UI
- UI manipulates the Domain Model
- Managers keep configuration state in the Domain Model
- Manager
- UI sends Jobs to the Managers
- UI sends Jobs to the Managers
What is Cobia?

I usually spent a few minutes on the previous diagram. I would ask the group if they had heard of Ivar Jacobson's Analysis model elements? Few did so I showed the relationship between the previous tiered architecture with Jacobson's model. Here is that diagram:

I renamed Jacobson's original “Control” component to “Service” to be aligned with modern terminology of these fundamental concepts.
Analysis Model and MVC

In one occasion this led to a question about the relationship of this architecture based upon Jacobson's model to the Model-View-Controller architectural pattern.

As you can see a Boundary component is comprised of both View and Controller aspects. The Entity and Service components comprise the Model aspects of MVC. Note that Entity and Service components are usually implemented by entity and session beans (and MDBs to handle asynchronous services).
What is Cobia?

- A rich user experience:
Example Screen Flow
Example Domain Model

RouteState

RipProtocol

StaticRoutes

BgpProtocol

BgpPeer

RipInterface

EthernetIntf
(from Admin module)
Example Manager

```
«interface»
ServiceManager
{from Cobia BASE} {abstract}
+startService():JobResult
+stopService():JobResult

«interface»
RouterManager
{from Cobia BASE} {abstract}
+initialize():JobResult
+getRipModel():JR
+setRipModel(RipProtocol):JR
+getBgpModel():JR
+setBgpModel(BgpProtocol):JR

AbstractManager
{from Cobia BASE} {abstract}
+<T>saveState(T state):void
+<T>restoreState():T

AbsServiceManager
{from Cobia BASE} {abstract}
+startService():JobResult
+stopService():JobResult
+isServiceRunning():boolean
// other service methods

XorpManager
+initialize():JobResult
+getRipModel():JR
+setRipModel(RipProtocol):JR
+getBgpModel():JR
+setBgpModel(BgpProtocol):JR
```
The Cobia Team

Mitchell Ashley
Business Manager

Paul P.
UI Designer

Leslie B.
UI Designer

KC Berg
Lead Architect

Bryan B.
SW Engineer

Paul C.
SW Engineer

Joel E.
SW Engineer

Heang L.
SW Q/A

Alex E.
SW Q/A
After the Cobia demo...

At this point in the presentation I would give a brief demo of the UI features of the Cobia system. The rest of the presentation I ignored the slides and create an atmosphere of interactive discussion that tracked the evolution of these topics iteratively in four phases of the Cobia project. There are the elements consider in each phase:
The Phases

For each phase, I provide the goals of the phase. From the goals the team made new technology and process choices. These choices led to known and sometimes unknown consequences.

The first phase was the beginning of the Cobia project. Most of the staff were new to the company. It was my first JavaServer Faces (JSF) project. We were all learning about StillSecure and specifically the existing libraries (such as CommonLib). This phase finished in July of 2007.

The second phase is marked by the creation of the Cobia UI component library. This phase finished when the first cut of the component library was completed in October 2007.

The third phase is marked by when the UI designers begin using the Cobia UI component library. This phase is just beginning.

The fourth phase is marked by greater independence of UI designers while building screens using the Cobia UI component library.
Topics Mind Map

The Cobia Project

Phase One
- Raw UI design with standard JSF components.
- Back-end SDK

Phase Two

Phase Three

Phase Four

UI Design & Development
Phase One Goals

The Cobia project started around May-June of 2006; most of the team was hired specifically for the project. I was hired as the UI developer. The primary goal of phase one of the project was to build modules and the Cobia infrastructure as fast as possible; the forming and storming phases. My personal goals were to learn all of the technologies that were involved in UI development and to code the UI screen designs pretty much as-is except where I had to augment with JavaScript were necessary. Later, I was able to discover new UI abstractions.
The UI designs are very detailed HTML, CSS, and some JavaScript. These screen designs are used for usability testing. On the UI development side I inherited two technology choices: JavaServer Faces (JSF) using JSPs for views and an internal UI framework. This framework included a small library of simple components and a major framework for doing multi-screen validation and breadcrumb management. We called this library CommonLib.
UI design was separate from UI development. The UI design team was formed first to begin scoping out prototypes of the first few Cobia modules: admin, router, DHCP, DNS, firewall. My job as UI developer was to code the screens as fast as possible. I did this by injecting the least amount of JSF and CommonLib components into each view. I made no attempt to create new abstractions, but I gradually began to find these abstractions, which became phase two.
Phase One Consequences

The results of these choices of technology and process. There were plenty of positive aspects of the team, technology, and process; such as, the creation of high-quality modules and our ability to meet aggressive iteration schedules. But for this discussion, this mind map focuses on the negative aspects. These are discussed in the next three slides.
Phase One Consequences

Over the first year of the Cobia project there were several problems with how the team as a whole (UI designers and engineers). For this discussion, we focus on UI design and development issues.

First, over time (from module to module) the raw HTML/JS/CSS design of certain types of UI widgets drifted. I was left with the choice of coding only the new widgets with the new design or refactor all of the existing screens that use this widget. Such drift is human nature. Designs naturally drift as the designer sees better ways of designing a specific widget. This type of drift happens in code as well; and no one is pointing fingers merely observing that it happens.

Second, there was significant redesigning of screens when development attempted to match up what the UI designer envisioned for the module with what the backend service was actually capable of producing. In many cases we sent screens back for redesign, which in some cases also required additional work/refactoring by the UI developer. (See next diagram)
Phase One Consequences

The problem with the *storming* UI design cycle was that the designers were often months ahead of the development process. When we discovered problems (primarily with the mismatch of the UI and what the backend service offers) this pulled the designers from the current work and also forced the UI development to have to recode the changes. In a few cases, this required from days to weeks worth of re-work.
Phase One Consequences

Third, I have observed that coding each screen by hand takes the UI designer about 80% of their time creating screens versus roughly 20% designing and creating new UI components (widgets). It occurs to me that the UI designers focus should be more on widget design rather than screen design. Screen design should be as easy as plopping down the required widgets in the desired layout, not by coding each widget by hand.

Fourth, the UI design team was often weeks or months ahead of the development team in creating screen designs for future modules. In phase one this had a much needed benefit of showing the vision of the future during this storming phase. So this consequence has both pros and cons.

Fifth, the design of new UI components were almost never complete (relative to the needs of UI development) nor fully testable. Many issues come up when taking a component design (in raw HTML and JS) and making it work with a server-side framework such as JSF.

Lastly, UI development itself was rather difficult. Knowing how to place the right JSF tags into the raw HTML in the JSP files was rather tricky. And identifying component design drift by comparing HTML of the design with the JSP files was also difficult. Furthermore, creating the JSF backing bean was also rather difficult and some code was highly redundant, such as the code required to interact with an data table of entities.
UI Design & Development

The Cobia Project
- Raw UI design with Cobia UI framework and components.
- Screen SDK

Phase One
- Raw UI design with standard JSF components.
- Back-end SDK

Phase Two
- Raw UI design with Cobia UI framework and components.

Phase Three

Phase Four
Phase Two Goals

At the end of the Cobia v0.2 release, I asked my management to allow me to build the Cobia UI Framework and component library. The technical goals of this effort were to create a library of reusable UI widgets, simplify UI development by eliminating raw HTML and JavaScript from JSP files (actually JSPX format), and to prevent the UI design drift problems.
Phase Two Technology

The biggest technical change in phase two was the creation of a nearly complete set of custom JSF components for Cobia-specific UI widgets, containers, input components, and so on. We also began making some use of existing JSF libraries such as RichFaces and Tomahawk. We also began make more use of AJAX and selected AJAX4JSF (A4J) from RichFaces. Lastly, we developed a framework that extends what CommonLib had done for us; thus splitting from that library.
Phase Two Process

Phase two saw the creation of the UI framework and component library. The greatest change was that UI development now included the discovery and creation of new UI components; this is not a trivial exercise. On the plus side, JSP development was far easier when using high-level JSF component tags with no HTML/JS. In this short phase (6 months), the UI design team did not change their process.
Phase Two Consequences

The greatest change in phase two was to simplify UI development. The V0.3 release of the Cobia SDK now includes the current UI component library and framework. Creating components also reduces the risk of UI design drift by encapsulating the rendering of any given component in a single renderer class. Abstracting UI widgets into real JSF components permits full life cycle testing of UI components. JSP development has become more easy, but creating the backing beans is just as hard.
Topics Mind Map

UI Design & Development

The Cobia Project

Phase One
- Raw UI design with standard JSF components.
- Back-end SDK

Phase Two
- Raw UI design with Cobia UI framework and components.
- Screen SDK

Phase Three
- UI design focused on building new UI components; not screens.
- Component SDK

Phase Four
Phase Three Goals

Phase three began as soon as Cobia V0.3 was released. The primary goal of phase three is to promote the use of the component library during screen design. This has two subgoals (a) to reverse the 80/20 UI design split and (b) to get the UI designers creating the screen designs (and all corresponding files: CSS, JS, text properties) in the Cobia codebase directly.
In phase three, the UI designers will need to use new tools to work with the component library. The designer can still use Dreamweaver which is capable of importing JSP/JSF tag libraries. The bigger hurdle are the tools required to build and deploy a JSF application: Ant, Eclipse, and a web container such as Tomcat.
Phase Three Process

In order to use the Cobia UI component library, UI designers will need to use a new process to design new screens by using some combination of Ant or Eclipse and Tomcat.
Phase Three Process

The larger change that this phase might require is the closer coordination of UI design and developer resources. Specifically, every screen in Cobia requires both a JSP page (now being created by the UI designer) and a JSF backing bean for that page. The latter is written in Java and therefore requires a UI developer to work with designers. This might be a blessing in disguise. Before phase three, the distribution of labor was linear; UI designers created screens and then sent these screens to development. Like this:

On the other hand, in phase three the work of designing and developing the screen code are done together. While both resources are tied together, it is theoretically possible that this team effort can be more efficient and effective. Like this:
Phase Three Consequences

The hope of phase three is primarily to reverse the design 80/20 and to reduce redesign problems by having the UI designs be created using the component library. Another side benefit might be having better prototypes for widget and usability testing.
Topics Mind Map

The Cobia Project

Phase One
- Raw UI design with standard JSF components.
- Back-end SDK

Phase Two
- Raw UI design with Cobia UI framework and components.
- Screen SDK

Phase Three
- UI design focused on building new UI components; not screens.
- Component SDK

Phase Four
- Screen design and domain modeling in parallel.
- Modeling SDK

UI Design & Development
Phase Four Goals

Phase four is gazing into the crystal ball. But ideally, we would like to provide UI designers with greater autonomy and to provide greater ease of use for UI developers.

Goals

- Code the UI design as-is
- Learn JSF
- Learn CommonLib
- Discover abstractions
- Create UI components
- No raw HTML/JS in JSP files
- Prevent UI design drift
- Reverse 80/20
- Designs become part of codebase
- UI designers using the component library
- Provide greater ease of development
- Give UI designers greater autonomy
As discussed in the sessions, code generation from the JSP files could solve both of the goals. First, code generation of common or redundant backing bean code could ease development. Second, code generation of domain models and demo data could support making UI designers more autonomous.
Phase Four Process

The use of code generation will have a definite impact on the design/development process, but mostly likely these process changes will be hidden behind Ant and Eclipse.
Phase Four Consequences

Clearly the desired consequences of using code generation are to each UI development and support greater UI designer autonomy. But until we actually try the changes recommended above, we will never know.

- UI designs too far ahead of development
- UI components were incomplete
- UI development was hard
- UI design drift

- 80/20 backwards
- redesign issues

Phase 1

- Using components reduces design drift
- Components go through testing
- Eases JSP development but not for backing beans

Phase 2

- reverse 80/20

Phase 3

- greater coordination reduces redesign issues
- better prototypes for usability testing

Phase 4

- More UI design autonomy
- Continued ease of UI development
[Image of a mind map titled "Topics Mind Map"]

**The Cobia Project**

- **Phase One**
  - Raw UI design with Cobia UI framework and components.
  - Screen SDK
  - Back-end SDK

- **Phase Two**
  - Raw UI design with standard JSF components.
  - Modeling SDK

- **Phase Three**
  - UI design focused on building new UI components; not screens.
  - Component SDK

- **Phase Four**
  - Screen design and domain modeling in parallel.
  - Modeling SDK