Airlift Framework for Java

Technology Overview

May 2005

Goals and Objectives of the Airlift Framework

The overriding objective of the Airlift Framework for Java can be stated as follows:

> ***To promote, enable and maximize the portability, reusability and longevity of developed application and business logic.*** *This includes persistence models, persistent objects and related artifacts, coded user interaction and validation logic, business process logic, access control logic, and persistent data processing logic.*

This stated high-level goal can be broken down into the following 2nd tier objectives:

1. *To promote and enable* ***a clean separation of presentation logic from underlying application/business logic****, such that the choice of existing or emerging presentation technologies, at various points along the lifespan of a developed application, can be made with minimal impact upon underlying application logic. Furthermore, multiple varying presentation technologies and configurations should be deployable simultaneously using the same shared application/business logic.*

2. *To promote and enable* ***a clean separation of application/business logic from underlying object persistence technologies****, such that the choice of existing or emerging persistence technologies, at various points along the lifespan of a developed application, can be made with minimal impact upon application logic. Furthermore, multiple varying persistence technologies and configurations should be deployable simultaneously using the same shared application/business logic.*

3. *To promote and enable* ***a model-driven approach to*** *development and maintenance of* ***an application's persistent objects*** *and related object persistence mappings and infrastructure.*

4. *To promote and enable* ***a model-driven approach to*** *development and maintenance of* ***an application's core user interaction and business process objects,*** *including interaction model composition, declarative query and service integration, access control point definitions, and generation of presentation layer controller mappings to application layer components.*

Areas outside of the scope of Airlift

1. ***Airlift does not seek to provide or replace core presentation technology**, but instead leverages third-party technologies for implementation of supported user interfaces through controller interfaces to the application layer. These technologies include the Struts Framework, Java Server Faces (JSF) technologies, the Java Swing (rich-client) API, and in future releases possibly SWT, WML, (and for web-service interfaces) SOAP, etc.*

2. ***Airlift does not seek to provide or replace core persistence technology**, but instead leverages third-party technologies for implementation of object persistence through persistence layer interfaces for use by the application layer. These technologies include the Hibernate version 2, Hibernate version 3, and in future releases, EJB 3.0, JDO, etc.*

3. ***Airlift does not seek to provide a "pure POJO" environment.** Because of the heavy-weight and intrusiveness of first and second generation J2EE technology (EJB 1.x and 2.x containers specifically), the industry seems to have reacted now with a pendulum swing toward what one might call "POJO Nirvana". While the Airlift team has been focusing on light-weight containers for many years and has intentionally avoided reliance on EJBs, Entity Beans, and heavy weight containers, we do not hold that POJOs are the answer to all the world's development problems. Just as XML became the buzz in the late 90s and was soon being over used and extended past it's practical, we think it is possible that now POJOs are being over emphasized to the point that technology developers are going to great lengths and building complex behind-the-scenes "code-generation magic" in order to make POJOs enterprise capable. We are suggesting that balance is in order with respect for POJOs, just as balance has proven to be in order with respect to XML.*

Patterns, Methodologies and Best Practices within Airlift

1. ***Compile-time checking is emphasized***. *Instead of relying upon external XML-based configuration and application integration meta-data (as does EJB, Spring, and other frameworks), Airlift makes intentional use of generated and/or hand-written Java code to wire application components together. The compile-time checked nature of Java code allows the developer to detect and correct deployment and configuration settings at coding-time or build-time instead of having to wait until the application is deployed and executed to discover configuration errors. While some external configuration is necessary (primarily in the form of the airlift-config file for example), external configuration and XML "application wiring" is minimized by design.*

2. ***Use of XML meta-data is intentionally minimized.***

3. ***Annotations and AOP will be selectively integrated***. *While the emphasis in Airlift will remain upon explicit, compile-time checked Java code, some strategic use of Annotations and Aspect Oriented Programming constructs will be introduced in select areas. For example, the use of annotations and AOP will be introduced as an alternative approach to the placement of transactional demarcation boundaries around Component methods which may require or mandate an active transaction block as advice for the method call, etc. In these cases the framework will provide the needed advice implementations and supporting meta-data needed for implementation using the chosen underlying AOP technology (JBoss AOP, etc).*

4. ***Model-driven "object graph navigation" artifacts are auto-generated and utilized for application "wiring"***. *Such artifacts can be used to create compile-time checkable declarative paths through graphs of persistent objects (EntityGraphs), app-layer model (Component) hierarchies, and implementation-independent Query graphs. For example, the UML Entity Model generator auto-generates static navigation instances which represent Entity fields and associations. These objects can then be chained together in Java code in order to declaratively map components and sub-components (e.g. Table Columns, Edit form fields, etc) down through complex Entity Graphs to the field level. If changes are made to an application's model such that coded entity graph paths are no longer valid, then the Java compiler will immediately detect and flag any application configuration code which utilizes object navigation paths which have either changed or been removed. In contrast, XML-based meta-data which has become out of date by referencing removed or altered*

classes, entities, etc, would probably not be detected as incorrect until a runtime exception is thrown.

5. ***Presentation Layer implementations are thought of a "faceplates" for application Components**. Imagine if you will a car stereo system which has a removable face-plate. The face-plate is the Presentation Layer and the stereo component sans-face-plate is what the Application Layer component hierarchy looks like. All the switches, buttons, and LCDs are there, but are only operational and functional when some face-plate is attached. However, many completely different looking face-plates can be attached, and entirely new ones invented for use on any component.*

6. ***"Dynamic Presentations" are highly valuable**. Furthermore, it is possible to create dynamic face-plates which morph themselves automatically to provide all the buttons and readouts that are provided by an underlying component. We call this an auto-generated presentation implementation because it completely builds up a default presentation to match, button-for-button and switch-for-switch, with each underlying app-layer component it discovered and interrogates for its features.*

7. ***Unit tests are just another presentation layer implementation.** A unit test is a form of face-plate, as well as a set of JSF components, a set of Struts actions and forms, as well as a dynamically runtime-generated Swing GUI. Unit test should obviously fully exercise the buttons, switches, readouts, and inputs of the Components they are designed to test.*

8. ***Application layer focus**. Our focus in Airlift is not the design, creation, layout, or selection of face-plates, that is left up to the various UI designers using various technologies. Our focus is on designing, modeling, coding, and unit testing a robust set of app-layer," faceless" components which are intelligently integrated, highly reusable, and pluggable into various use-cases across potential suites of applications.*

Integration with Java Technologies

(todo)

Component Functionality Matrix

Aspects \ Component Classes							
Content Caching						■	
Content Assignable			■	■		■	
Session Identity /Component Key					■		
Management by AppSession					■		
Presentation Entry Point					■		
Compositing / Sub Components		■					
Savable /saveChanges() impl		■					
Conversation Propagation					■		
State History Management / Awareness							■
State Maintained Across Conversations						■	
Context Sensitive					■		
Context Propogation					■		
Label / Title	■						
ToolTip	■						
GetSetValue				■			
EntityFieldWrapper				■			
Is Selected			■				
Option List				■			
ValidationStatus				■			
Actions	■						
Access Control	■						

Component Hierarchy

- Component
  - CompositeComponent
    - SelectableComposite
      - MenuItem/Row/TreeNode

- Component
  - FieldBasedComponent
    - ValueSelectionField
    - ValueEditField

- Component
  - CompositeComponent
    - ManagedComponent
      - ContentCachingComponent
        - StateNavigableComponent
          - TableComponent
          - FormComponent
            - SearchForm
            - EditForm
          - PagingResultsList
        - CustomComponents

- Component
  - CompositeComponent
    - ManagedComponent
      - SearchResultsComposite
      - ListDetailComposite

Comparison with other frameworks and related technologies.

The following spreadsheet is an attempt to compare the Airlift framework with other frameworks and APIs including the Spring framework, and EJB 2.x and 3.x. As there are some (unknowns at least to the author) with EJB 3.0 etc, there are some guesses and "?" where insufficient information was readily available at this writing. Updates will be forthcoming.

Framework/API Feature Comparison	AirliftJ	Spring	EJB 2.x	EJB 3.x
Transactional Support				
Declarative Transaction Demarcation through XML	NO	YES	YES	?
Declarative Transaction Demarcation through Annotations	PLANNED (through AOP)	YES (Spring AOP)	NO	YES
Declarative Transaction Demarcation through Callback wrapper	YES	YES	NO	?
Declarative Transaction Demarcation on Actions	YES			
Compile-time checking of Transaction Demarcation settings	YES	NO (in most cases)	NO	checked annotations?
Two-phase Commit Support	PLANNED (JTA TransMgr)	YES (through JTA)	requires JTA	YES (through JTA)
Conversion of Exceptions to Transaction Rollback	YES	YES	YES	YES
Unchecked Exceptions bubble to Transaction Blocks	YES	YES	?	YES?
Explicit Rollback support	YES	YES	YES	YES
Hibernate 2 support	YES	YES		
Hibernate 3 support	YES	YES		YES
JDO support	as needed	YES		
JTA support	PLANNED (JTA TransMgr)	YES	YES	YES
Others (Toplink, etc)	as needed	YES		
Distributed Transactions	must use JTA	must use JTA	must use JTA	must use JTA
TransactionManager available using ThreadLocal	YES	YES		
Object Persistence				
Pluggable/Abstracted Global Identity (GUID) generator	YES			?
Pluggable/Abstracted ORM implementation	YES		through CMP	through CMP
Pluggable/Abstracted Query implementation	YES			
Compile-time checking of Query construction	YES			
Access to native ORM impl query language	YES (HQL)			through EJBQL
Access to native SQL	through JDBC	through JDBC	through BMP	through BMP

		wrapper		
Lookup Entity by GUID	YES		through BMP/CMP	through BMP/CMP
Support for Entity and Dependent Objects	YES		YES	YES
Pluggable/Abstracted JVM & Session-level Cache support	YES			
Declarative mapping of Constraints to Queries	YES			
Passivate and Activate Queries	YES			
Compile-time checked EntityGraph declarations	YES			
Remoting				
Remote method calls	through IoC Commands	through Proxies	EJB/JCA	EJB/JCA
Aspect Oriented Programming (AOP) Support				
Provided AOP		Spring AOP		Depends on Annotations
Third-party AOP integration	PLANNED (JBoss AOP)			Container-provided (JBoss, etc)
Deployment and Configuration				
XML-based configuration of dependencies	optional/limited	YES	YES	YES
Code-based (compile time checked) dependency configuration	YES	optional/limited		
Multi AppContext configurations per JMV	IN DEVELOPMENT	YES		
Annotation-based configuration	PLANNED (through AOP)	YES		YES
Emphasized Patterns, Methodologies & Best Practices				
Focus on POJOs	POJO neutral	YES	NO	YES
Lightweight Container Emphasis	YES	YES	NO	YES
XML-centric deployment and configuration	NO	YES	YES	NO
Focus on Compile-time checkable deployment / object-wiring	YES	NO	NO	
Model-driven business objects with base-class functionality (Entity)	YES			
Model-driven user-interaction models (Components) with stock fns	YES			
Dependency Injection (setter)	some	YES	some	some
Dependency Injection (constructor)		YES		
Configurable Factories	YES	YES		
Configurable Factories with complex "object wiring"		YES		
IoC through Service Locator	YES			

Use of Template pattern for reusable stock functionality	YES			
Security and Access Control				
XML-centric declarative Access Control points	NO	YES	YES	
Annotation-oriented declarative Access Control Points	PLANNED (through AOP)	third-party??	NO	YES
Compile-time checked declarative Access Control Points	YES		NO	NO?
Model-driven declarative Access Control Points	PLANNED			
Pluggable/Abstract access control point (permission) checking	YES	third-party??		
Access Permissions integrated with UserProfile and Authentication	IN DEVELOPMENT	third-party??		
Data-oriented Constraints integrated with User Profile	IN DEVELOPMENT			
Compile-time checked data-oriented access-control Constraints	YES			
Automatic propagation of access control Contraints to Queries	IN DEVELOPMENT			
Localization / Internationalization				
Message Bundles integrated with (request context) Locale	IN DEVELOPMENT	YES		
Request context Locale integrated with User Profile	IN DEVELOPMENT			
Application Layer Infrastructure / Support				
Business components abstracted from Presentation code	YES	YES	as EJBs	as EJBs
Lifecycle support for app-layer components / business objects	YES	YES	as EJBs	as EJBs
Templated pattern impl for common use-case component wiring	YES			
XML-oriented component/object wiring	NO	YES		
Compile-time checked component wiring	YES			
Data-oriented Constraints auto-propagate between wired components	IN DEVELOPMENT			
Selection-oriented Constraints auto-propagate between wired comp	IN DEVELOPMENT			
Managed Components provide Presentation entry points	YES	through Bean Factories	EJB lookup	EJB lookup
Declarative navigation through Component hierarchies	YES	through Java Bean methods		
Presentation Layer Integration / Support				
Struts controller access to App Layer components	thru Template Actions/Forms	through Factory wrappers	EJB lookup	EJB lookup

JSF controller access to App Layer components	thru Template Backing Beans	through Factory wrappers	EJB lookup	EJB lookup
Model-driven generation of JSF backing beans	PLANNED			
JSF controllers auto-generated (at runtime) from components	PLANNED			
Swing presentation locally integrates with app-layer components	IN DEVELOPMENT	integration possible	NO	integration possible
Rich Swing model integration with stock component interfaces	IN DEVELOPMENT			
Swing presentation auto-generated (at runtime) from components	IN DEVELOPMENT			
Model-driven generation of Swing controllers	PLANNED			
WebWork integration		YES		
Tapestry integration		YES		
Unit Testing Integration / Support				
Base Unit Test classes to support context setup	YES	YES		
Base Test Suite classes to support context setup	YES			