Air Arabia

Design Guideline Document

Document Version 1.0

Document Summary

Project Title	Air Arabia
Project Code	AA_RES
Document ID	~~<DocID>~~ AA_020
Document Version	1.0
File Name	'AA~~A~~A_RES_DesignGuideline.doc
Template ID	T105
Template Version	1.0
Project Repository	Air Arabia on Domino_NTT
Date of First Release	~~01~~15-Jul-2005

	Name	Date
Prepared by	Mohamed Nasly, Chandana K	30-Jun-2005
Reviewed by	Kasun C. Kumara, Nipunika Edirisinghe	06-Jul-2005
Reviewed by	~~Kasun C. Kumara, Nipunika Edirisinghe~~Chandana Kithalagama	~~06~~12-Jul-2005
Approved by	Kasun C. Kumara	~~dd~~12-~~mmm~~Jul-~~yyyy~~2005

Table of Contents

1 Introduction

Design Guidelines is a product of architecture definition. It describes the guidelines to be followed during design, architectural design, and implementation. It is an artefact created and maintained by the software architect, and serves as a communication medium between the software architect and other developers.

1.1 Purpose

The purpose of this document is to communicate the design standards and conventions to be used in the design of the system.

1.2 Scope

The document specifies guidelines for Air Arabia project. Provides design guidelines reference for Module designers and developers.

1.3 References

This section provides a complete list of all documents and other sources of information referenced in the Software Architecture Document

Document ID	Description	Comments
Guidelines for Database Design - G028	Database design guidelines	JKCS standards for database design
~~UI specification~~		
~~SRS~~		
~~User messages guide.doc~~	~~Presentation related messages~~	
~~Oracle Database Designing Guidelines_V2.doc~~	~~Database design guidelines~~	

1.4 Definitions, Acronyms and Abbreviations

Definitions: A concise explanation of the meaning of a word, phrase, or symbol.

Acronyms and Abbreviations: A word formed from the initial letters of a multi-word name E.g. DIP Development in Progress

Definitions, Acronyms and Abbreviations	Description

2 Design Phase Artefacts

Software Architecture Document
Design Use Case Documentations [at module level]
Use Case Realizations (Sequence Diagrams, Collaboration Diagrams) [at module level]
State Diagrams [at module level]
Logical Diagrams [at module level]

3 General Design and Implementation Guidelines

Designing and implementing to provide modularity is a key to success of the project. The overall software architecture is decomposed into a set of modules. In general, modules are self-contained collection of codes, scripts and configuration information that are logically and functionally coherent. Modules interdependencies must thoroughly be managed to maximize the reuse. Modules are uniquely identified by a name, which should be configurable. Each module has its own source area, independent build and documentation. A module

provides its services through a well defined set of interfaces. Modules are assembled into applications along with the supported configurations.

3.1 Mapping from Design to Implementation

Modules are grouped into layers where lower level layers provide system services and upper level layers provide domain specific services with modules in upper level layers only accessing functionalities provided by modules in a lower level layer. Within in each module following categorization is maintained: service layer, delegation layer, remoting layer, business logic layer and persistence layer. Modules are made remotely accessible by providing a remoting layer over the modules' business services implementation layer. Transparent access to module's services to clients is provided by implementing a delegation layer over the remoting layer. In general persistence layer is implemented through Object Relational Mapping techniques. Inter-module communication and modules' lifecycle are managed by a separate module, the platform.

3.2 Specify Interfaces on Subsystems

The application is composed of multiple subsystems (modules) each of which exposes a well defined set of interfaces that other subsystems and clients use to obtain services from the subsystem.

At the highest level, a subsystem exposes its services through a well defined module's service interface. The subsystem's service interface acts as the factory for creation of subsystem's business delegates which exposes subsystems business services. Each module provides two types of delegates: remote logic accessible delegates and local logic accessible delegates. Thus delegates help providing location transparency to the clients with maximization of module distribution among physical nodes. The subsystem's model and exceptions are also exposed to the client.

In summary, a subsystem exposes following interfaces: service interface, business delegates, models and module exceptions.

3.3

~~3.3~~3.4 Document the Operations

Operations are named using standard java method naming conventions. Each operation should be described what it accomplishes and the arguments to the operation should be documented specifying any constraints that apply. The operation's results and exceptions that may be raised during the operation should clearly be mentioned. If the operation involves complex logic, the operation specification should carry the pseudo-code for the operation. The constraints, pre-conditions and post-conditions of the operation are also documented.

~~3.4~~3.5 Document the Messages

All the presentation tier related messages are documented in UI technical specifications. Each message is defined with a unique message Id and carries a brief description.

~~3.5~~3.6 Detecting, Handling, and Reporting Exceptions

The problems that occur at runtime must be handled through some contract that allows the originator of the error to pass appropriate information to the caller who will know how to handle problem properly. Exceptions occurring in each of the following layers should be handled in a uniform way in all the modules. The module level design should maintain the uniformity.

Persistence layer
Business logic layer
Remoting layer
Service layer

In general, following guidelines are used in handling exceptions;

Exceptions occurring within a module should be propagated to the client wrapping the exception in module's API level exception. i.e. Each module should expose only one checked exception and one unchecked exception to service requestors.

Each checked exception should subclass generic checked exception defined by the framework, and each unchecked exception defined at module level should subclass generic unchecked exception defined by the framework.

If the service requestor has no clue of recovering from the exception and exceptional condition occurred is fatal, wrap the exception in the module's unchecked exception throw to the requestor.

Exceptions occurring at lower level layers of a module should be chained and propagated to the caller as a module's exception so that caller has the clue of exception occurred.

~~3.6~~3.7 Memory Management

Java Garbage collector automatically handles this.

~~3.7~~3.8 Software Distribution

Software is distributed as one enterprise archive and three web archives. Business services modules are packaged along with system services in an enterprise archive which may be deployed in multiple nodes. The web modules are packaged with module clients and the system services in a web archive which may be deployed in multiple nodes. The communication between web tier and backend is achieved through Java RMI-IIOP through Enterprise Java Beans technology. Inter-module communication can be remote or local depending on the node(s) the modules deployed.

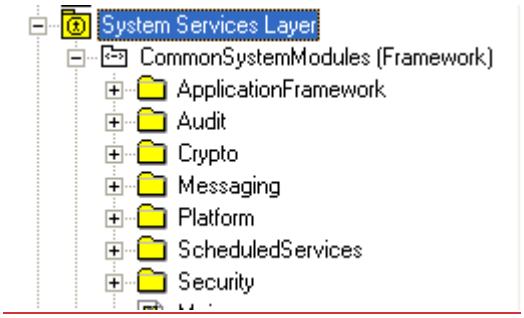
~~3.8~~3.9 How to Represent Reusable Components

Project uses following third-party open source frameworks:

Jakarta Struts	For implementing MVC in presentation tier
Hibernate	For implementing ORM
Spring	To manage modules and modules' configurations through IoC Uses the AOP engine for implementing cross-cutting concerns
Quartz	Scheduling engine
Velocity	For templated email message generation
JUnit	For unit testing functionalities
xDoclet	For generating EJB interfaces and hibernate mappings
Ant	Build automation and unit test automation

For all the development dependent third-party components a package would be shown to depict the dependency in relevant diagrams.

All other reusable components should be grouped under a System Services package and would typically contain following components.



Application Framework - module initialization, services for inter-module communication,
Platform – shared objects used by business and system services.

3.93.10 Designing Persistent Classes

Persistence layer is designed on top of hibernate ORM implementation. The lifecycle of the persistence objects are managed through hibernate. The ORM implementation provides an API for performing basic CRUD operations on objects of persistent classes and also provides a language or API for specifying queries that refer to classes and properties of classes. The design should ensure the CRUD operations are performed with maximum efficiency. The techniques for the ORM implementation to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions should be used to maximize the efficiency.

3.103.11 Transaction Management

Transaction management can be achieved using enterprise transaction management services provided by the EJB container or the transaction management facilities provided by the spring framework. By default, container provided transaction management will be used.

At persistence layer, transactions are managed declaratively using springs declarative transaction management facilities.

3.113.12 Program Structure

The project follows General Sun Java Coding standards. Each module follows a well defined module packaging structure. Each operation should be documented using javadoc conventions.

Structure	Remarks
com.isa.thinair	
<module>	all modules have similar structure to the following
api	
service	BD interfaces and the module interface
mod	
el	entities
modelException.java	model exception class
cor	
e	
service	
bd	BD implementation
remot	
e	
ejb	Service bean implementation
bl	any business logic classes (optional)
persistance	
dao	DAO interfaces
hibernat	DAO hibernate
e	implementation
<module>	
ibwe	
b	public web application

logi
n
registration
reservation

xbweb — web application for CC staff and travel agents
ccres
agentres
rebook
refund

adminweb — web application for ROP and system admin
pricing
schedule
aircraft

com.isa.thin

- <module 1> all modules have similar structure to the
  - api
    - servic BD interfaces and the module interface
    - model entities
    - modelException.java model exception class
  - core
    - servic
      - bd BD implementation
      - remot
        - ejb Service bean implementation
      - bl any business logic classes (optional)
      - persistance
        - dao DAO interfaces
        - hibernate DAO hibernate implementation
- <module 2>
- <module 3>
- ibweb public web application
  - login
  - registration
  - reservation
  - …
- xbweb web application for CC staff and
  - ccres
  - agentres
  - rebook
  - refund
  - …
- adminweb web application for ROP and system
  - pricing
  - schedule
  - aircraft

```
com.isa.thin
├── <module 1>        all modules have similar structure to the
│   ├── api
│   │   ├── servic        BD interfaces and the module interface
│   │   ├── model         entities
│   │   └── modelException.java        model exception class
│   │
│   └── core
│       └── servic
│           ├── bd        BD implementation
│           ├── remot
│           │   └── ejb    Service bean implementation
│           ├── bl    any business logic classes (optional)
│           └── persistance
│               ├── dao            DAO interfaces
│               └── hibernate     DAO hibernate implementation
├── <module 2>
├── <module 3>
│
├── ibweb         public web application
│   ├── login
│   ├── registration
│   ├── reservation
│   └── …
├── xbweb         web  application  for  CC  staff  and
│   ├── ccres
│   ├── agentres
│   ├── rebook
│   └── refund
│       …
└── adminweb      web application for ROP and system
    ├── pricing
    ├── schedule
    └── aircraft
```

~~3.12~~3.13 Algorithm Guidelines

Algorithms used throughout the system can be divided into two categories. They are standard and business specific. The standard algorithms are encryption algorithms used for encrypting user password, credit card information, etc. The business specific algorithms are used in seat allocation, seat availability checking, etc. All the complex business algorithms should be given in pseudo code in the design documentation. Recommendation of which standard algorithm to use for each case is given below.

Use case	Algorithm	Remarks
Encrypting user password	PBEWithMD5AndDES	Tried and tested algorithm, used internally
Encrypting credit card information	PBEWithMD5AndDES	Same above.

~~3.13~~3.14 System Modification and Build Guidelines

The CVS manages source code versioning. The developers are responsible for keeping the code repository up to date. Each function introduced or modified should thoroughly be tested before committing the changes to the CVS repository. The project uses Jakarta Ant for automating build process and the unit test execution automation. Each module should be able to be built on its own and tested. Each module's build should make use of globally defined build utilities.

~~3.14~~3.15 System Diagnostic Guidelines

As application level diagnostic mechanisms, application logging and tracing will be adopted. A Memory profiler will be adopted in order to diagnose system issues and Oracle dictionary views will be used to diagnose database problems.

4 Database Design Guidelines

Project follows database design guidelines specified in JKCS standard for database design documentation (Guidelines for Database Design - G028~~Oracle Database Designing Guidelines_V2.doc~~).

4.1 Mapping from persistence classes to database structures

Follows persistence class mapping strategies defined by Hibernate framework.

4.2 Mapping of design class attributes to database primitive data types.

Follows type mapping conventions specified by Hibernate framework.

5 Architectural Design Guidelines

Refer to the AA_ModularArchitecture.doc document for details on architectural design guideline document.

6 Mechanism Guidelines

N/A