Web Responsive Design (Web RD) – Conductor Framework – White Paper

Sunil Panore, Lead – Software

April 2016

Version

Date	Version	Author	Description
April 11, 2016	1.0	Sunil Panore	Initial Draft
April 18, 2016	1.1	Asma Doni	Updated document as per the latest template and reviewed and formatted the same
May 03, 2016	1.2	Sunil Panore	Updated the document and sent to Asma Doni for further review
May 04, 2016	1.3	Asma Doni	Re-reviewed and completed the document

Reviewed By

Date	Name
April 18, 2016	Asma Doni
Month DD, YYYY	Charudatt Garud

Contents

Introduction

Web Responsive Design (Web RD) - Conductor framework provides set of features and tools useful for creating scalable, Multi-Tenant, customizable, and single-page Web-based responsive design applications.

Following are the features and tools provided by Web RD - Conductor framework:

- Page flow framework for creating Responsive User Interface (UI)
- Admin Console Tool for managing and creating lightweight UI [Manage data for Page flow framework]
- Conductor framework for managing workflows
- Support for Multi-Tenancy
- Pipeline for managing deployments

Limitations with Legacy Systems

Following are the limitations found in legacy systems:

- Legacy systems are not compatible to run on other devices such as Mobiles and Tablets because of heavy UI layer
- Separation of concerns is not well maintained
- Unit testing for components is difficult
- Reusability and extension is very limited
- No Support for Multi-Tenancy
- Manual deployments are error prone

Objectives

The key objectives of developing this framework are listed as below:

- Standardization across and quick development for new systems
- Easily maintainable UI – achieved using Admin Tool
- Customizable user experience
- Reusable business workflows which can be further extended for use with Mobile Apps
- Isolated UI and Business Logic for better unit testing
- Support for Multi-Tenancy
- One-click deployments

Importance of Web RD – Conductor

Web RD - Conductor originates from challenges of Legacy Systems. For creation of Lightweight UI, Separation of concerns to promote easy unit testing, support for Multi-tenancy and one click deployments to save time / manual efforts is the demand of modern technology. Legacy systems had limited or no support for the above mentioned areas. This is where the tools and features provided by Web RD - Conductor framework prove useful. WEB RD - Conductor forms the technology core of web based responsive design systems.

Normally developers end up spending hours for making page structure. Using Web RD - Conductor this is well managed with the help of Admin Console Tool which can be used for managing pages as well as the content. This makes the application dynamic and helps avoid manual efforts for UI changes. This acts as a base framework for making UI changes in Web RD - Conductor.

Support for Multi-Tenancy and reusability plays an important role. Multi-tenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client-organizations (tenants). Multi-Tenant feature add business as well as technical benefits to the systems.
Separation of Concerns which refers to Unit testing, plays an important role to achieve product quality. The solution is to have loosely coupled layers which would promote unit testing at different layers so that issues could be separated in terms of layers.

Deployment part comes for all systems usually at the end of the development process. Traditionally, this process has been manual intervention which is error prone and time consuming. To overcome this Web RD - Conductor framework comes with the concept of build pipeline which supports one click deployments and automatic creation of packages.

Topic of this paper and its scope

The scope of this paper covers the following topics:

- Web - RD solution
- Integration and information flow between different layers of Web RD – Conductor framework
- Multi-Tenancy
- Deployment Pipeline

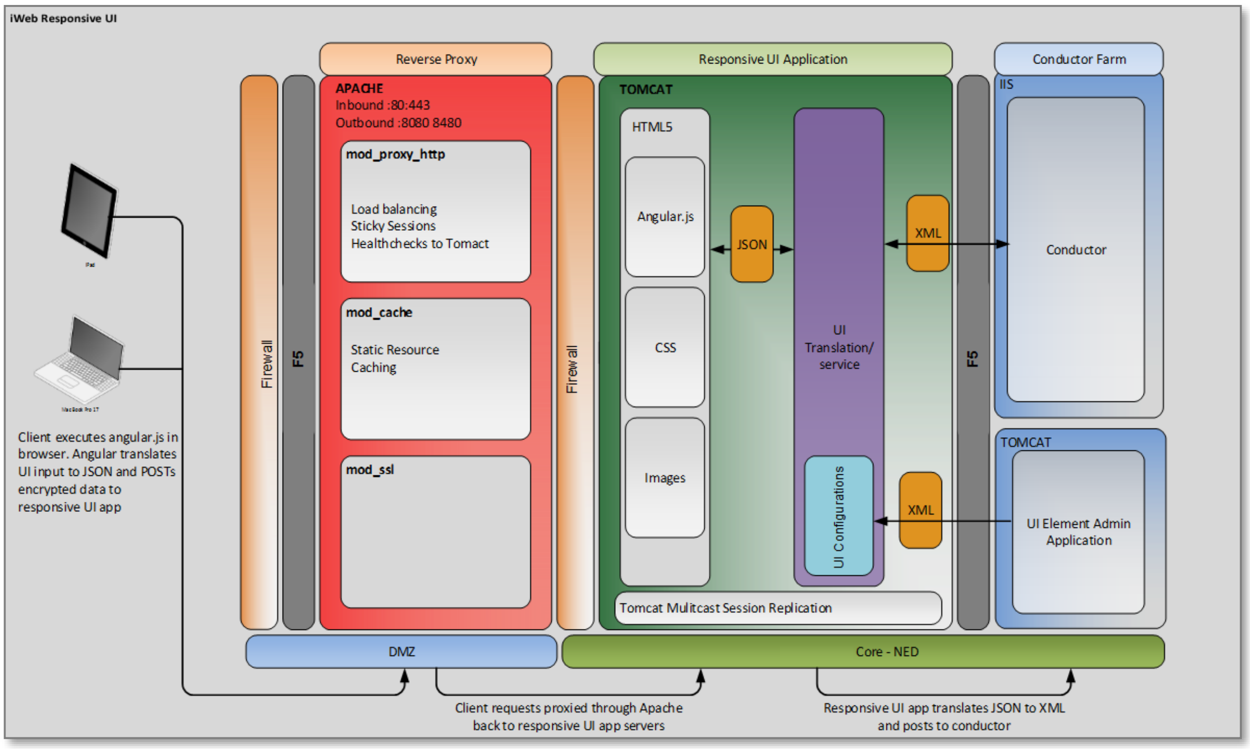
Solution Offering By Synechron

Synechron has developed a framework called Web RD - Conductor that provides solution to the preceding limitations with Legacy Systems.

Web RD - Conductor framework operates on the inner side of the firewall, which forms the following inner two layers:

- Responsive UI
- Conductor

Apart from the above two layers, there is a pipeline setup that acts as a back bone of deployment for various environments.



iWeb Responsive UI
Reverse Proxy
APACHE
Inbound :80:443
Outbound :8080 8480
mod_proxy_http
Load balancing
Sticky Sessions
Healthchecks to Tomact
mod_cache
Static Resource Caching
mod_ssl
Firewall
F5
Responsive UI Application
TOMCAT
HTML5
Angular.js
CSS
Images
JSON
XML
UI Translation/ service
UI Configurations
Tomcat Mulitcast Session Replication
Conductor Farm
IIS
Conductor
TOMCAT
UI Element Admin Application
DMZ
Core - NED
Client executes angular.js in browser. Angular translates UI input to JSON and POSTs encrypted data to responsive UI app
Client requests proxied through Apache back to responsive UI app servers
Responsive UI app translates JSON to XML and posts to conductor

Fig. 1: Web RD - Conductor implementation

Web RD Framework

Multi-tenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client-organizations (tenants). Multi-tenancy contrasts with multi-instance architectures where separate software instances (or hardware systems) operate on behalf of different client organizations. With a multitenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance.

At a very high-level, there are three different parts of the Web RD framework, they are as follows:

- **a.** User Interface
- **b.** Conductor
- **c.** One-Click Deployment (using build pipeline)

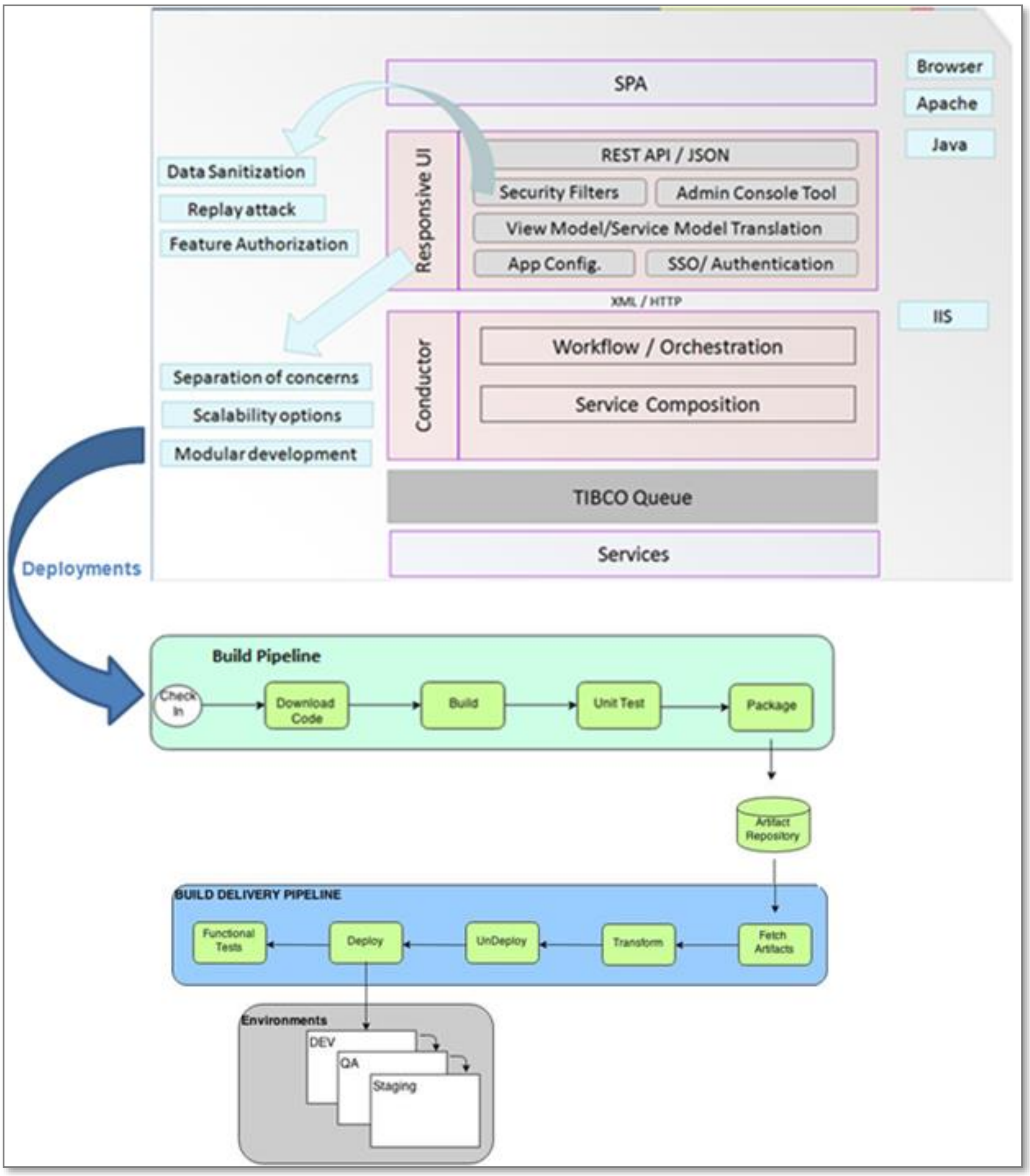


Fig. 2: Web RD – Conductor Framework

Following are the features of Web RD - Conductor framework:

- Page flow framework [Responsive UI]
- Admin Console tool for managing and creating lightweight UI (manage data for Page flow framework)
- Conductor framework for managing workflows (Business Flows) and Service orchestration
- Multi-Tenancy Support
- One-click deployments using Pipeline

a. User Interface Layer (UI)

The User Interface Layer of the framework used for managing pages and contents for the applications. There are two different parts of UI, i.e., Page Flow Framework and Admin Console.

- **Page Flow Framework (PFF)**

  PFF forms the User Interface structure, which acts as a baseline for all the UI objects. It also supports Single Page Application (SPA) and the details are as below.

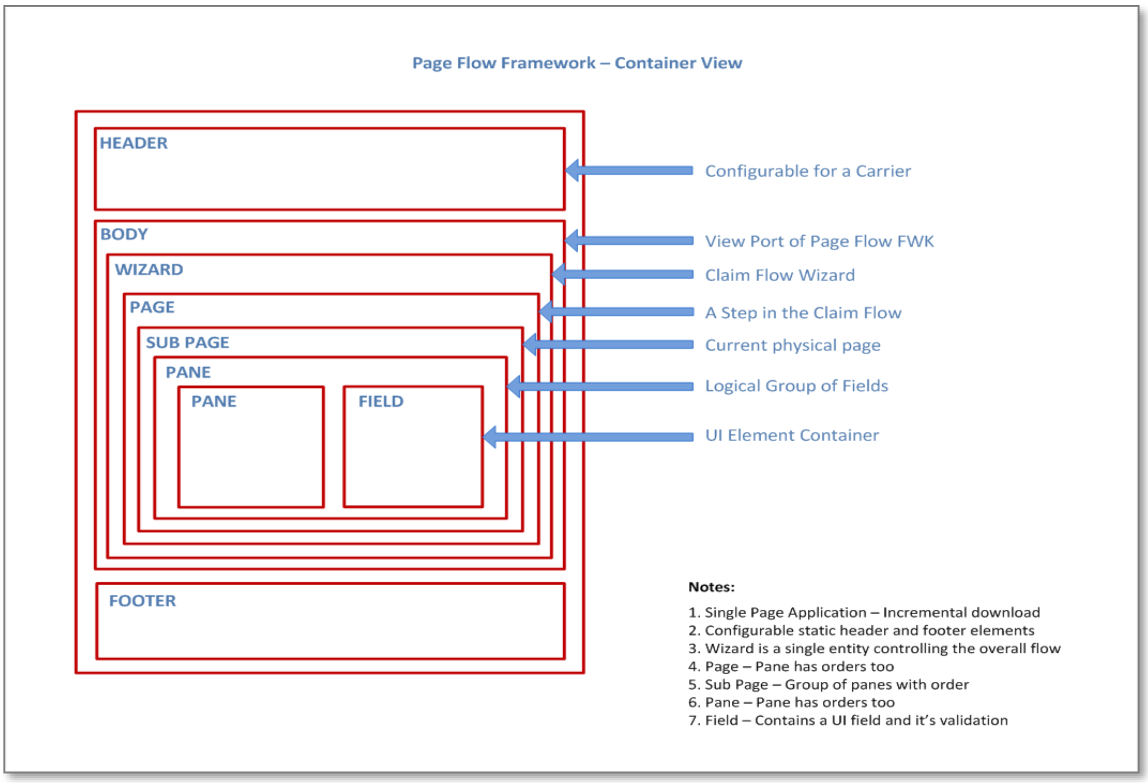

Page Flow Framework – Container View
HEADER
BODY
WIZARD
PAGE
SUB PAGE
PANE
PANE
FIELD
FOOTER
Configurable for a Carrier
View Port of Page Flow FWK
Claim Flow Wizard
A Step in the Claim Flow
Current physical page
Logical Group of Fields
UI Element Container
Notes:
1. Single Page Application – Incremental download
2. Configurable static header and footer elements
3. Wizard is a single entity controlling the overall flow
4. Page – Pane has orders too
5. Sub Page – Group of panes with order
6. Pane – Pane has orders too
7. Field – Contains a UI field and it's validation

Fig 3: Page Flow Framework – Container view

  - ✓ Header act as a common area for all the pages, which is configurable for the client.
  - ✓ Body forms the dynamic content for the application, which contains configurable Wizard and Pages.
  - ✓ Page is one important part of the framework'
  - ✓ Pages can further have Subpages which in turn have Panes, which forms part of page or subpage.
  - ✓ Fields are granular levels controls on the page.
  - ✓ New pages or subpages are loaded under the same wizard who makes the application responsive.

- **Admin Console tool**

  Containers in above figure are configurable and this can be achieved with the help of Admin Console tool.

  At a very high-level, Admin tool performs following two functionalities
  - ✓ Setting up Page flow framework User Interface for clients

- Setting up carriers
- Setting up wizards
- Creation of Page Structure
- Create Sub Pages
- Create Panes and Fields
- Create Locale and Error Locale
- Setting up Headers and Footers dynamically

✓ JSON export for different environments

- Export the JSON content in the form of scripts for optimization

Multi-Tenancy, is only possible if there is a support to configure different carriers in the same framework. Admin tool helps setup new carriers or modify existing carriers as per the need. Once the carrier is created we can configure the Wizard, Page structure, etc. as part of it using Admin Console tool.

Admin Console tool provides support for creating and update for the pages.

This tool also helps achieve optimization on POD / Production environments. The content scripts can be generated in the form of JSON which is key value pairs and pushed to environment as needed. This makes the system perform faster as the same can be cached on the Web Servers.

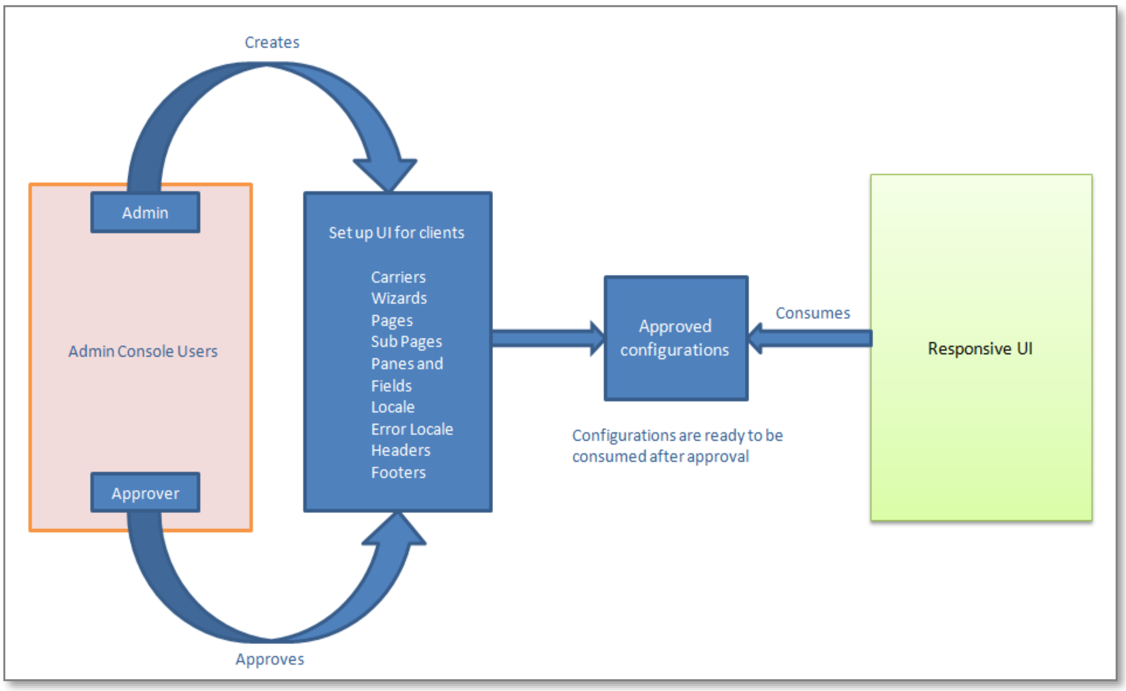


Fig. 4: Setting up Page flow framework UI for clients using Admin Console Tool

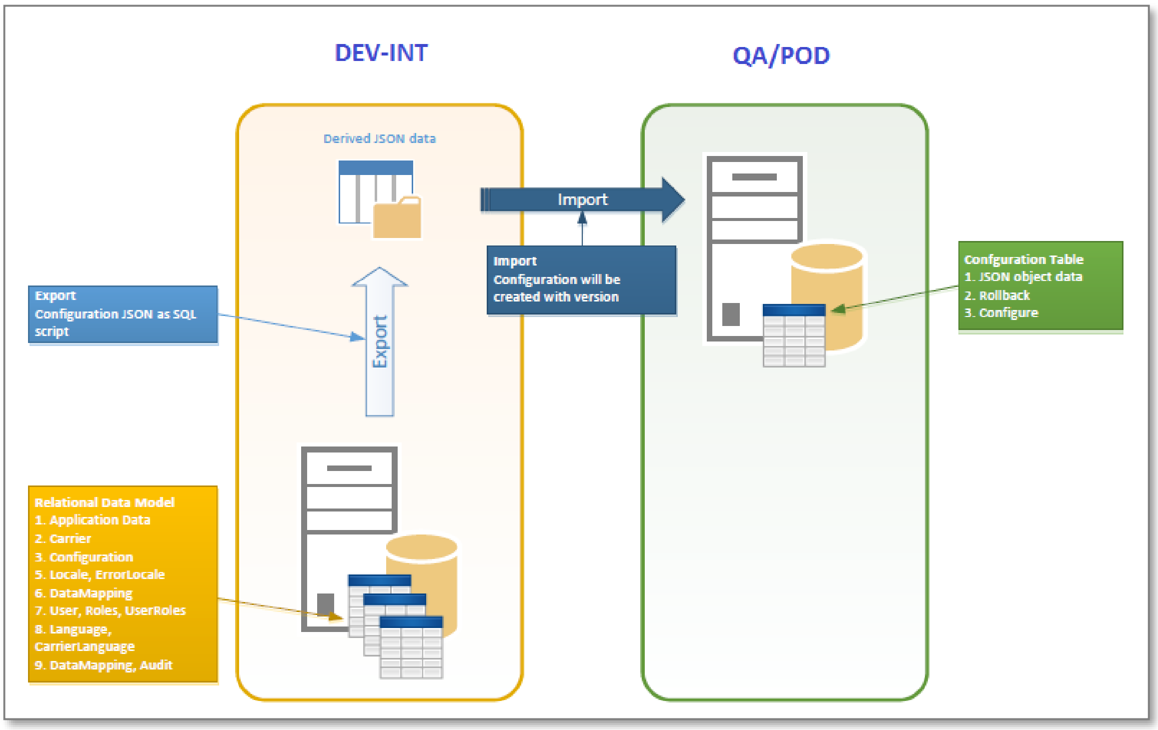


Fig. 5: JSON export of scripts through admin console for different environments

- **Detailed UI flow**

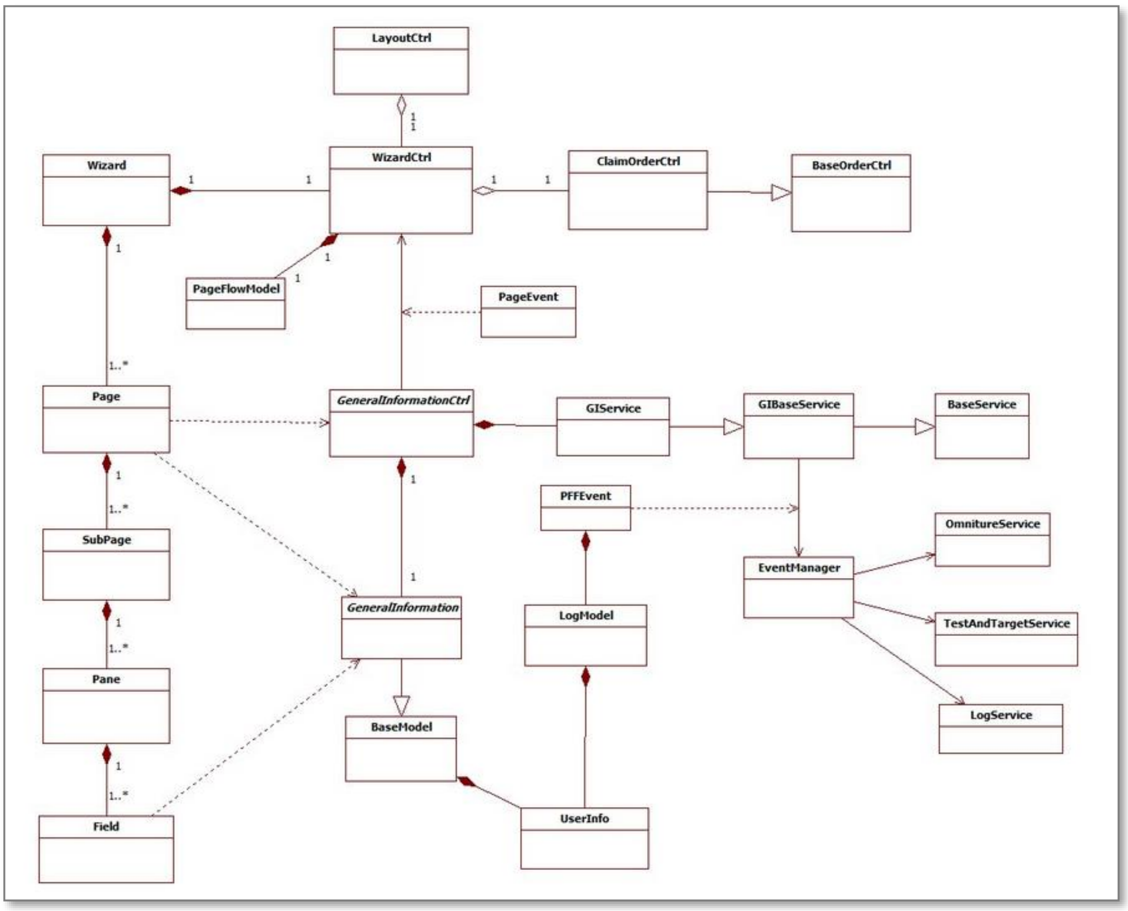
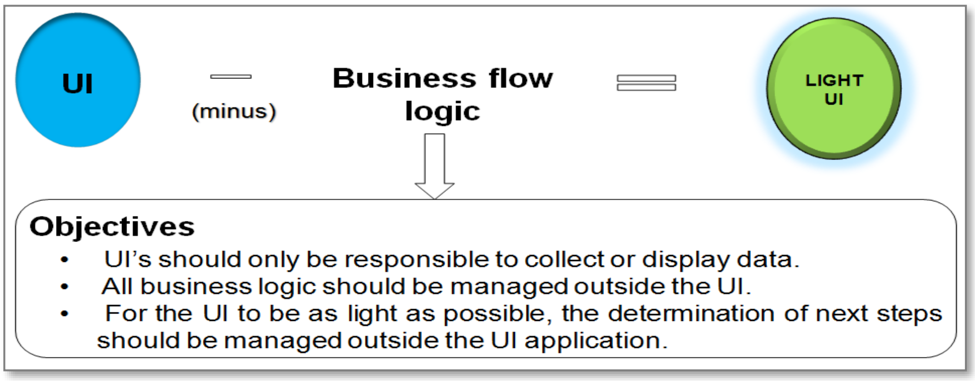


Fig. 6: Detailed UI Flow

UI is responsible for Data capturing and validation. State of the UI is stored on the client (Browser), this improves performance and the responsiveness of the application. Reduced memory and resource usage on the server and therefore scalable. The application becomes Easier to debug and maintain as very little server affinity issues. To achieve Use platform agnostic message formats such as JSON or XML.



Objectives

- UI's should only be responsible to collect or display data.
- All business logic should be managed outside the UI.
- For the UI to be as light as possible, the determination of next steps should be managed outside the UI application.

Fig. 7: How Web RD – Conductor is different from Legacy applications

b. Conductor

Conductor helps in creating dynamic business workflows using Goal driven approach. Conductor acts as a body which Decouple or Detach Business Logic from UI. It also takes care of service orchestration and simplifies interaction with backend services. Conductor also has the capability to share data between related applications.

Conductor forms the workflow base of all the applications and thus provides Single design for all clients based on Configuration, this forms one of the crucial factors while developing Multi-Tenant applications. Changes at the service level for both versions and contracts can be easily adopted using Conductor. Flow management is easy and time saving when it comes for modification or creating any new business flows.

For UI to be as light as possible, the determination of the next step should be managed outside the UI (application) this is achieved by using goal driven approach as Conductor empowers the application by taking all business decisions this ensures that UI does not have to bother about making any decisions and based on the conductor goals. This ensures that UI can run into simulation mode and unit testing for UI and Conductor is Isolated which in turn promotes parallel development.

Conductor makes Calls to necessary services to retrieve the data required by the UI. Finally it also determines the next Goal or module the UI should perform within a Process.

At the bottom level Conductor uses handy configurations which Minimizes code changes. UI's should only be responsible to collect or display data.
The below diagram shows what changes are introduced by bringing Web RD - Conductor to replace Legacy applications.

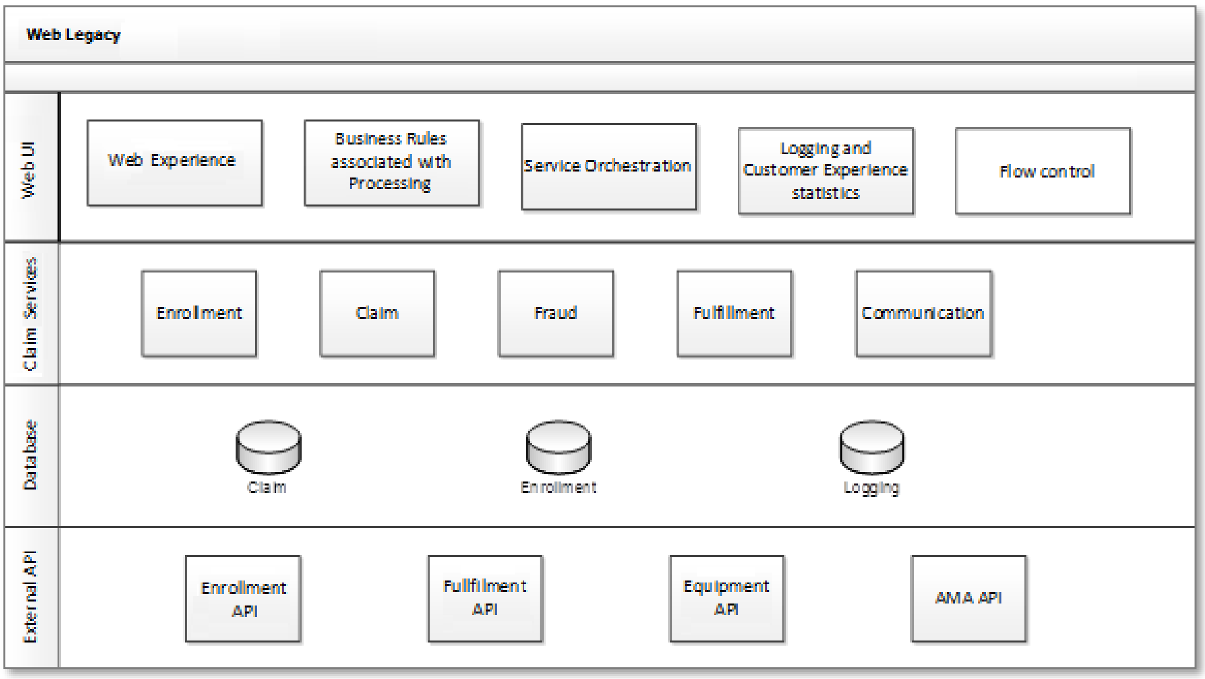
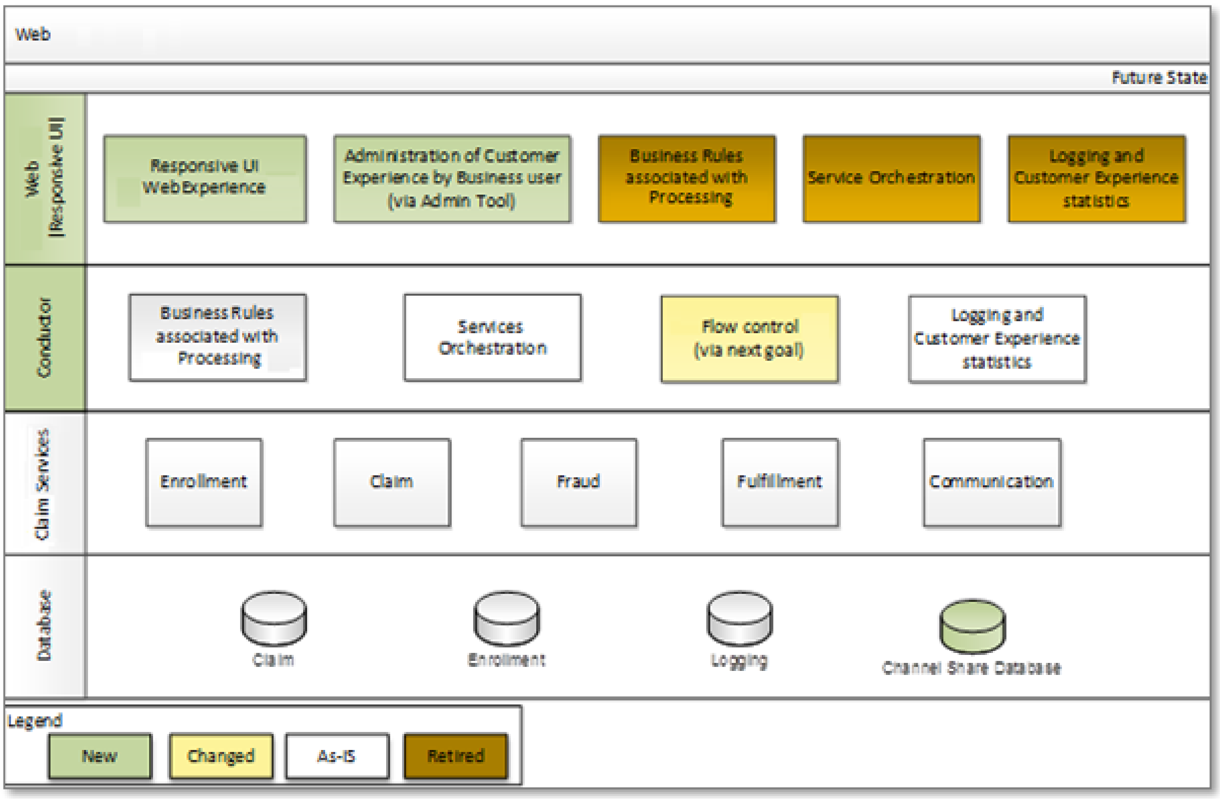


Fig. 8: Legacy system



Web
Future State
Web [Responsive UI]
Responsive UI Web Experience
Administration of Customer Experience by Business user (via Admin Tool)
Business Rules associated with Processing
Service Orchestration
Logging and Customer Experience statistics
Conductor
Business Rules associated with Processing
Services Orchestration
Flow control (via next goal)
Logging and Customer Experience statistics
Claim Services
Enrollment
Claim
Fraud
Fulfillment
Communication
Database
Claim
Enrollment
Logging
Channel Share Database
Legend
New
Changed
As-IS
Retired

Fig. 9: Web RD - implementation with Conductor

- **Detailed Conductor Flow:**

  The calls to conductor always start from UI which hits the conductor gateway [Endpoint exposed on the UI]. After this, Conductor loads any shared data from Channel Share Database which holds common data for various applications. After this the evaluation process starts for the process which in turn has States, Behaviors and Activity. This is the point where the workflow executes for the request. If it is a first request then conductor would first create the ground area which includes loading of any shared data based on the request.

  Any goal call from UI hits conductor in the form of State which is part of the actual workflow. Based on the configurations defined for a carrier the Workflow starts executing and during this activity there may be data exchanges or Service / API calls. The data received from UI or services may be later useful in the workflow hence this needs a storage at the conductor side. This storage is called as Activity Data.
  Once the flow is executed there are two things which are returned on the UI this includes

  ✓ Goal Name
  ✓ Goal data [WHICH is consumed by the UI]

  Based on the Goal Name, UI will perform the necessary redirection or loading of a Page/Subpage. This data is posted to UI in the form of XML which is converted into JSON by a rest service.

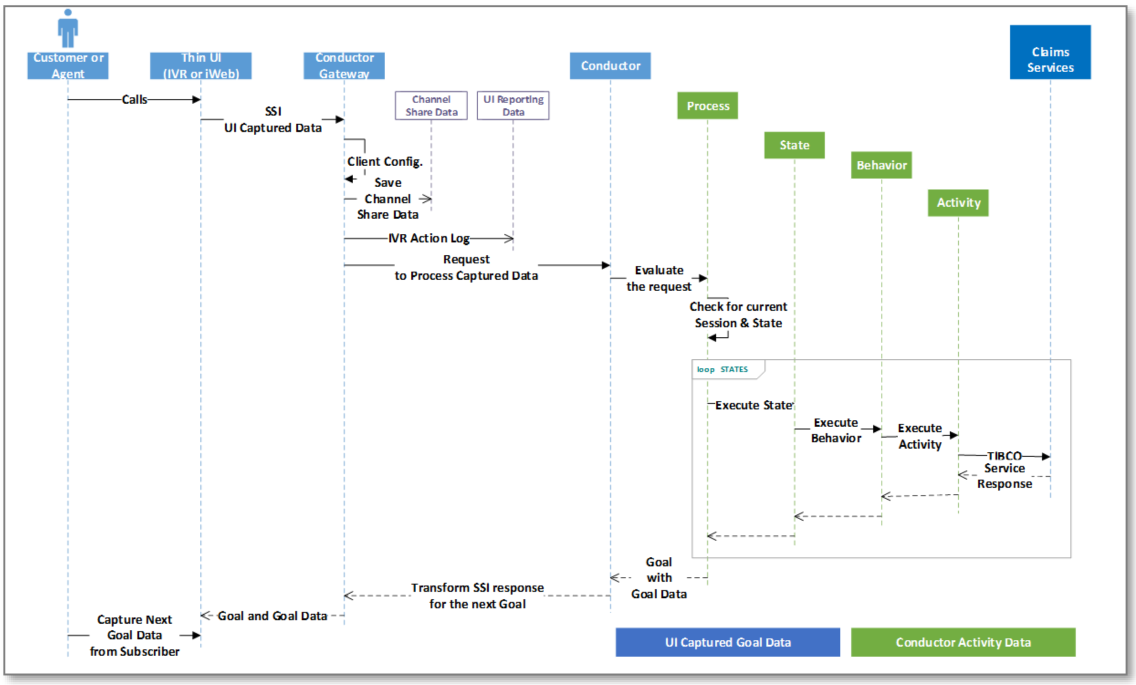


Fig. 10: Sample request from UI served by conductor

- **Key Conductor components**

Conductor has the following Key components

✓ Endpoint
✓ Channel Share Data
✓ Agent Services
✓ Cache
✓ Goal builder
✓ Core workflow engine
✓ Logger

Endpoints are exposed to UI, every request from any of the clients end up consuming endpoint in order to communicate with the conductor. This can be setup using type of client.

Channel share data is used for storing any shared data for different applications which end up using conductor. Usually before serving any request conductor would look out for data under CSD.

Agent services are used for communicating with different services and API's from conductor. This forms a point of communication from Conductor to external world.

Cache is used for storing data locally for any request sessions. During a process there is an exchange of data between conductor and other layers this is managed by using Activity data. This Activity data resides under the Cache.

Goal Builder forms the backbone of the goal driven architecture. As iterated earlier workflow decisions are taken by conductor. Goal builder is the conductor component which manages Goal creation for the UI.

Core workflow engine is the component which is responsible for execution of business workflows. Engine communicates with Goal Builder to generate the next goal.

Logger performs the function of logging the activity data, UI requests, service request / responses, API request / responses and any exception that might occur during Conductor initialization or workflow execution.

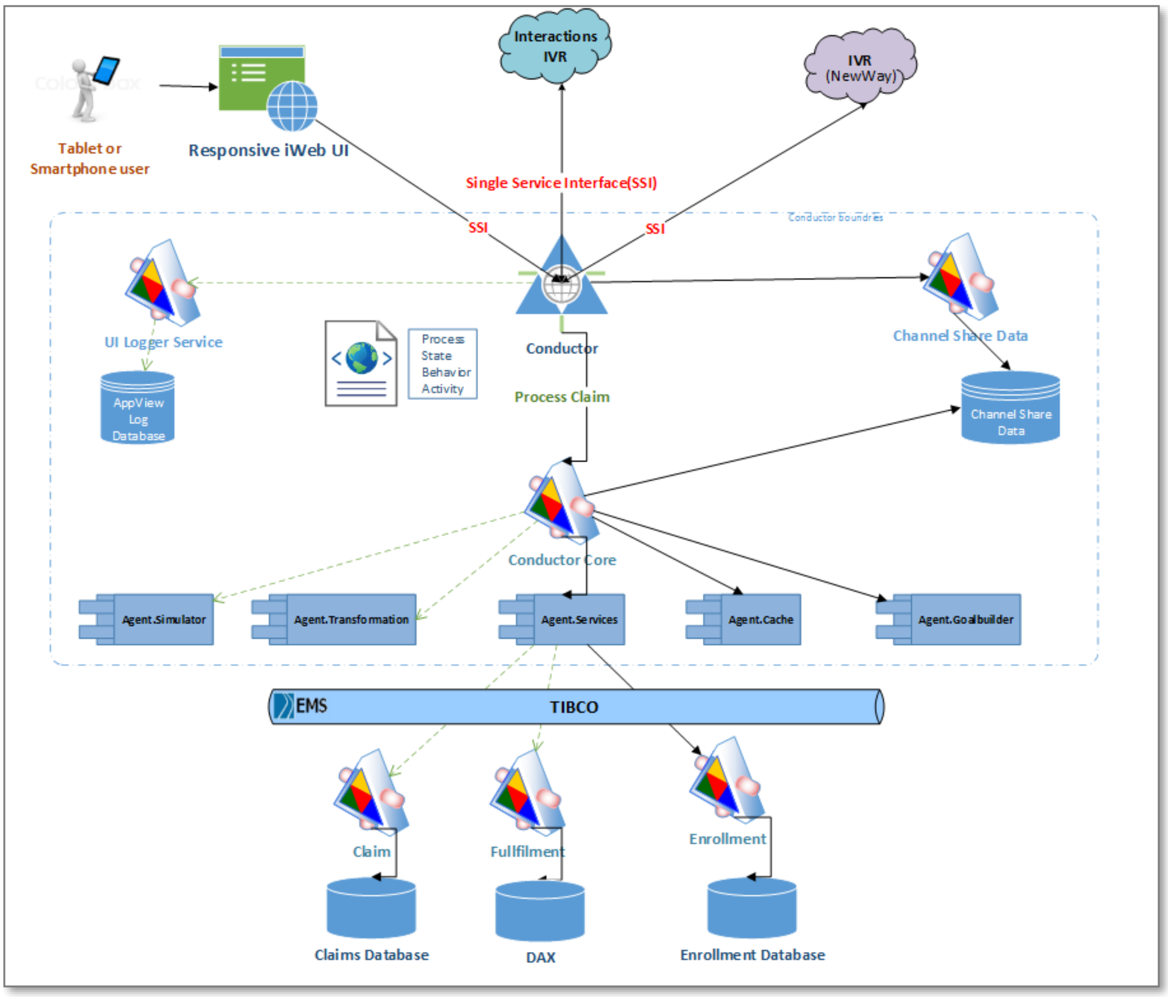


Fig. 11: Key Conductor Components

c. One Click Deployments

ANY business user/administrator can select a target environment (which kicks in an automated transformation process that results in the environment keys of the target environment) with appropriate access can do a One-Click deployment using this tool.

This suite of tools that was developed for the purpose of:

- ✓ To ensure the deployments are quick and automation reduces deployment errors.

- ✓ To provide an eco-system to abstract and maintain all the environment specific configuration and transform configuration files based on deployment environment
- ✓ To fasten the feedback loop between the Development team and the business regarding the quality of deployments.
- ✓ To add more quality gates to reduce the cost of issues post deployment

This part of the framework provides the following features:

- ✓ Listens for code changes and initiates a build based on any code change
- ✓ Runs the automated unit tests
- ✓ Checks for compliance for a set of rules like code coverage.
- ✓ Uploads the build that has complied with all metrics into an artifact repository
- ✓ As a part of Continuous delivery, any build can be selected and deployed
- ✓ Builds are downloaded from Artifact repository and deployed into selected environment.

- **Continuous Integration Pipeline:**

  Jenkins listens for any code changes checked in made by developers. The CI pipeline is invoked when a check in is made by the developer. There is an interval between the actual code check in and the Jenkins pipeline being invoked. In case there are other Check INs made during this period, they are queued and executed in order of the check ins.

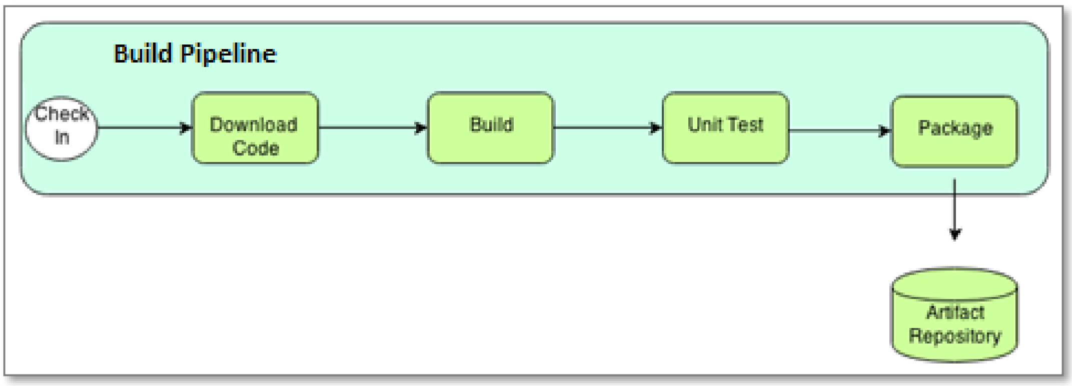


Fig. 12: CI Pipeline [Conceptual view]

Above mentioned steps are individual jobs in the Jenkins pipeline. The pipeline only differs in the internal implementation of the jobs. Internally, these jobs call into ANT scripts/executables which carry out the operation required by the job. Therefore, there is NO logic written in the Jenkins job.

The Jenkins CI pipeline is responsible for the orchestration of these jobs and the result of the CI pipeline is a package- a deployable unit. This package is stored in an Artifact repository. The Continuous Delivery pipeline picks up the required package from this Artifact repository and deploys the package to the required environment.

- ✓ Developers do check out code in their own workspace workspaces. When completed, they commit changes to repository using commit policy configured in SVN.
- ✓ The CI server monitors the repository and checks out changes when they occur.
- ✓ The CI server builds the system and runs unit/integration tests.
- ✓ The CI server uploads deployable artifacts to artifact repository.
- ✓ The CI server assigns a build label to the version of the code it just built.
- ✓ The CI server informs the team of the successful build.
- ✓ If the build or tests fail, the CI server alerts the team.
- ✓ The team fixes the issue at the earliest opportunity.
- ✓ Continue to integrate and test throughout the project.

- **Continuous Deployment Pipeline**

  The Continuous Delivery pipeline is responsible for fetching the package(s) from the Artifact repository and deploying the package(s) to the required environment. In theory, we should be able to deploy every version of the software present in the Artifact Repository but it's rarely the case that every new build produced as a result of a check in will actually be deployed. Typically, deployments are made at specific intervals and therefore the Continuous Delivery (CD) pipeline needs to be manually invoked to deploy a particular version from the Artifact repository to a specific environment. Here, environment refers to DEV, QA, Staging and Production.

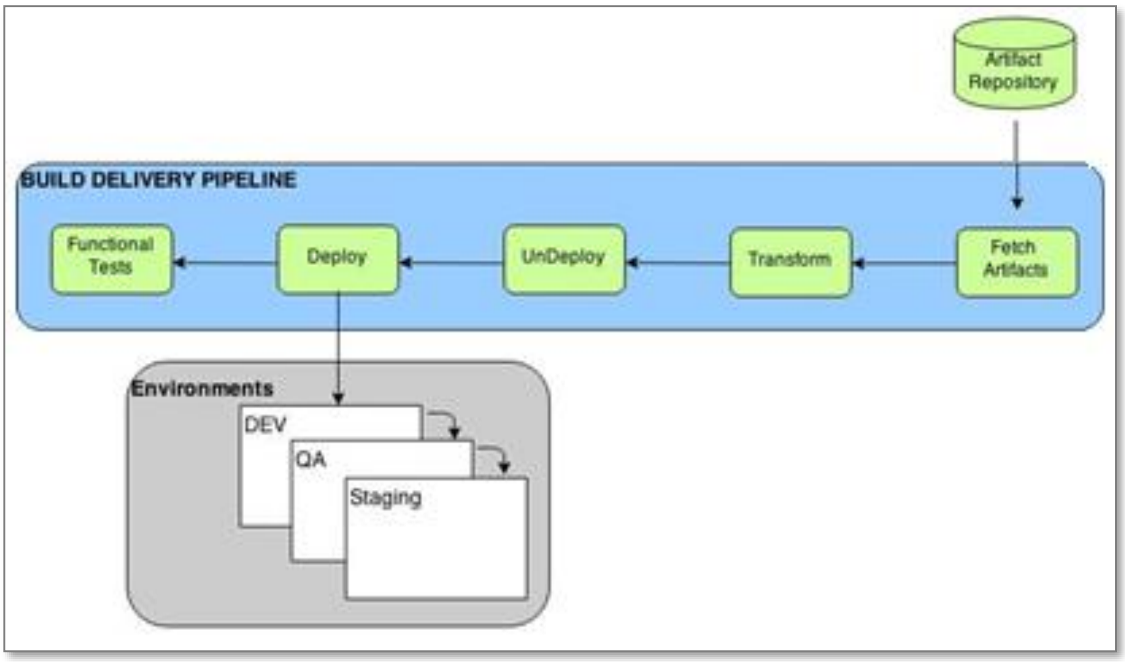


Fig. 13: CD Pipeline [Conceptual view]

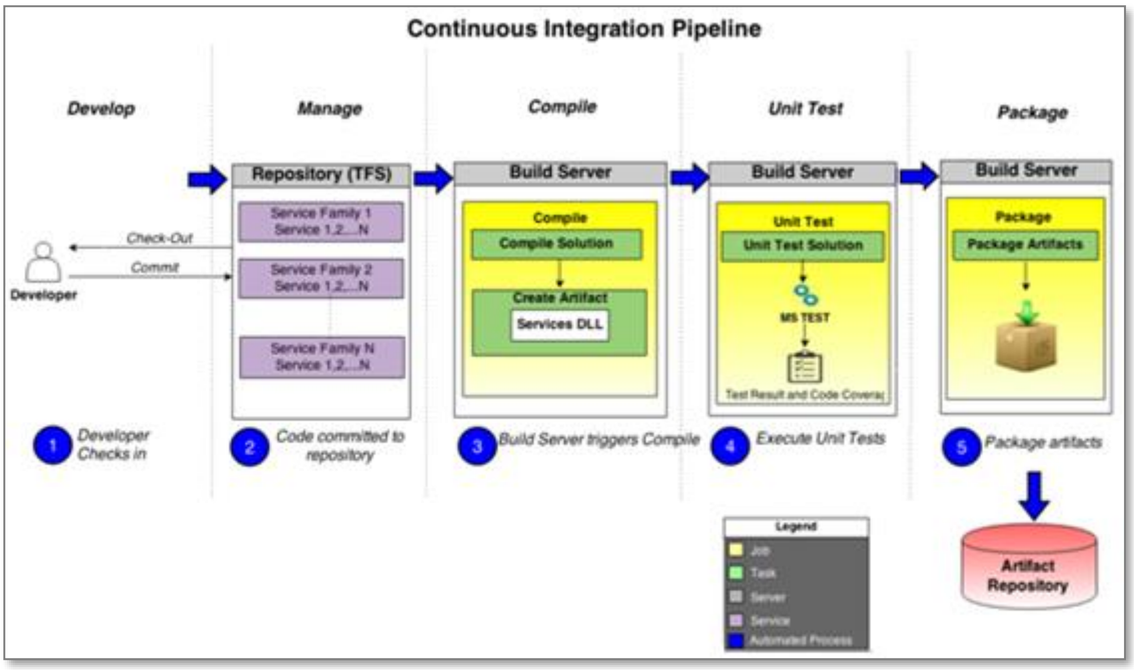


Fig. 14: Continuous Integration pipeline

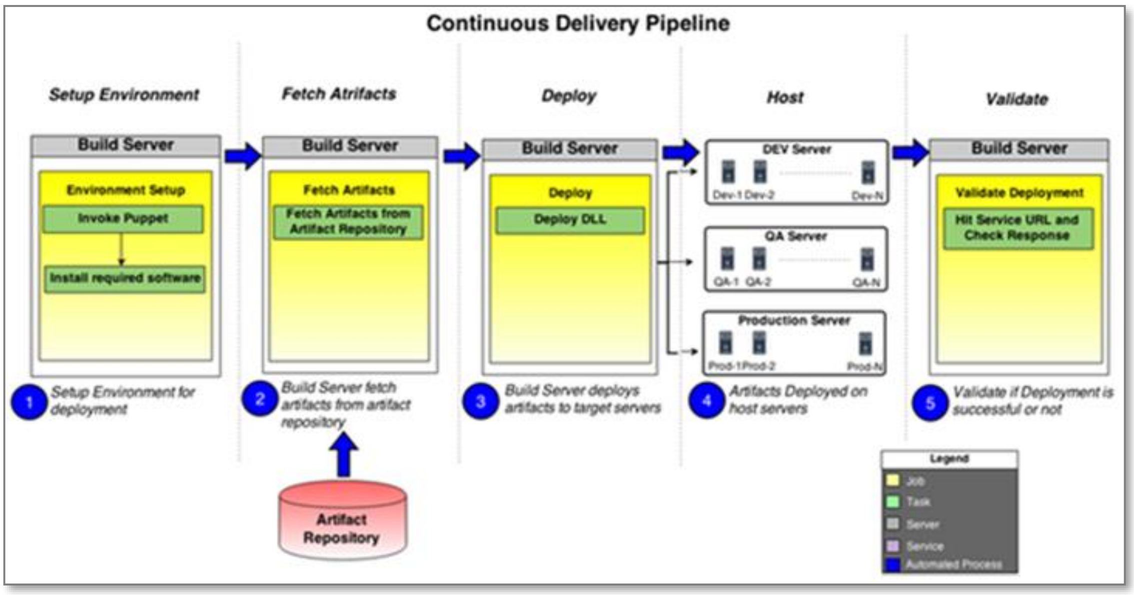

Continuous Delivery Pipeline
Setup Environment
Fetch Atrifacts
Deploy
Host
Validate
Build Server
Environment Setup
Invoke Puppet
Install required software
Fetch Artifacts
Fetch Artifacts from Artifact Repository
Deploy
Deploy DLL
DEV Server
Dev-1 Dev-2 Dev-N
QA Server
QA-1 QA-2 QA-N
Production Server
Prod-1 Prod-2 Prod-N
Validate Deployment
Hit Service URL and Check Response
1 Setup Environment for deployment
2 Build Server fetch artifacts from artifact repository
3 Build Server deploys artifacts to target servers
4 Artifacts Deployed on host servers
5 Validate if Deployment is successful or not
Artifact Repository
Legend
Job
Task
Server
Service
Automated Process

Fig. 15: Continuous Deployment pipeline

Multi-Tenancy

Multi-Tenancy helps to achieve business as well as technical benefits. It highly promotes reusability of existing developed components. The idea is to capitalize on the existing components developed and serve new clients with minimum efforts. This cuts down most of the development costs and also helps maintain standardization across clients.

Multi-Tenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client-organizations (tenants). Multi-tenancy contrasts with multi-instance architectures where separate software instances (or hardware systems) operate on behalf of different client organizations. With a multitenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance.

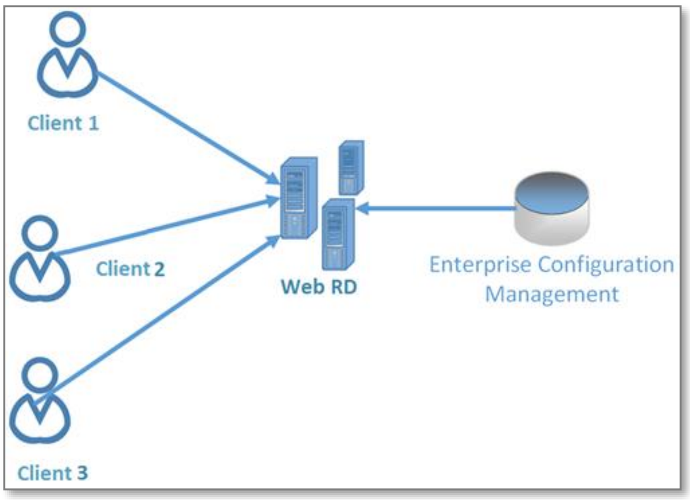


Fig. 16: Multi-Tenancy

a. Multi-Tenancy in Web RD - Conductor?

Multi-tenancy in Web RD - Conductor is achieved with the help of loose coupling between UI and Conductor, this is done in the form of Goals. As said above Conductor applications makes decisions and returning goals on the UI. In the UI framework Wizards are defined for carriers which are mapped directly to Goals. These goals are consumed by Pages / Subpages which is again defined by Wizards. The goals returned by Conductor, UI has to just perform the action defined.

Admin Console can be used for configuring UI for the existing / new carriers as per the need. This will take care for making changes on the UI side for the new clients. The base reusable framework of Web will still stand the same. From the business flow perspective, by adding flow configurations for the new client in Conductor the new business flows can be achieved. Under this whole process we end up using the same base framework for reusability and hence it results in on boarding new client in the least time possible.

The pipeline takes care of codebase, and hence, new client ends up using the same pipeline, which do not require additional efforts.

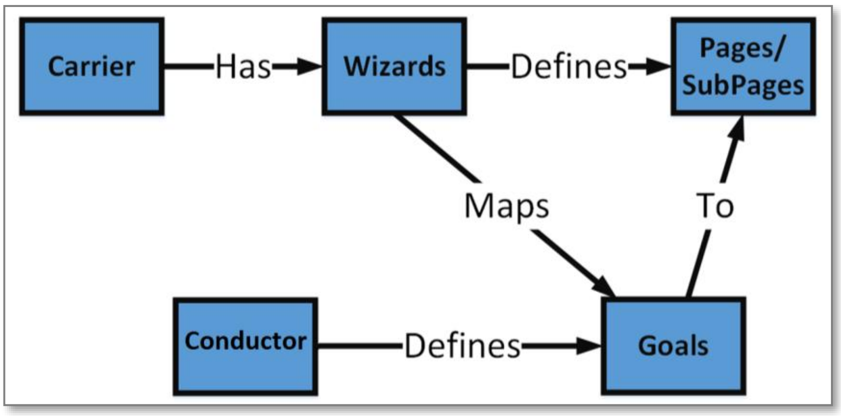


Fig. 17: Multi-Tenancy achieved in Web RD – Conductor

b. Web RD - Conductor Framework feature summary

The Web RD framework is quite agile and robust it is built upon understanding of shortcomings of legacy systems. The Web RD architecture is lean and agile and fully scalable

Salient features of Web RD framework:

- Admin Console based configuration (editor, approver) using Admin tool
- Isolation of cross cutting concerns
- Well defined interfaces between the various layers that promotes loose coupling
- Promote component based design and code reuse
- Easy to expose functionality as services to external applications and platforms will be scalable and

  E.g. - Mobile apps
- Modular design makes it easy to test each layer independently and follow a test driven development approach.
- Automatic build promotion and installation using build pipeline (zero manual effort)

Case Study

a. Context

As one of the insurance client has an application which does the claim processing on Web, the application having mixture of Web RD - Conductor and legacy application. But, Legacy applications do not support and cannot be operated upon Mobile phones or tablets because of technical constraints. Using Web RD - Conductor framework these applications can be made self sufficient to run on other mobility devices.

As the customers keep on increasing investing on other channels such as CSR has a direct financial impact on the investment and costs. Using Web RD - Conductor framework Web application can be made accessible to customers who want to access Web applications directly on Mobiles / Tablets. This would help save costs in terms of infrastructure and manpower which would otherwise be required.

b. Objectives

- It is proposed to implement framework to address issues legacy systems and also can easily pluggable and configurable as per client requirement.
- The framework should be very agile and scalable to overcome shortcomings of legacy systems and configurable to add new requirement as per clients.
- Any UI changes should be easy to incorporate on the framework.
- Also, Multi-Tenancy should be an available option in the framework to onboard any client with minimum effort.
- Deployment challenges should be overcome to save time and cost compared to Legacy systems.

c. Solution

Synechron solution included a GAP analysis of the existing architecture of the legacy system and their shortcomings, pointing the technical constraints faced by the legacy system and proposing solutions that will be able to cater to the objectives using the Web RD framework. It also included on-boarding of a couple of internal consumers onto the Web RD - Conductor framework.

d. Salient Features

- Detailed analysis of the legacy system, functional and technical
- Analyze the technical constraints in the legacy system
- Business challenges faced currently and the target state intended
- Migration and training plan for the post WEB RD Framework implementation
- Metrics comparison post implementation and continuous tracking for reaping the benefits of the WEB RD Framework