QuickRDA

RDA Modeling Overview

Erik Eidt

Version 3.0

May, 2011

***Copyright (c) 2011, 2012 Hewlett-Packard Development Company, L.P.***

[Introduction 5](#_Toc335635323)

[Architectural Artifact Alignment 6](#_Toc335635324)

[QuickRDA: RDA Contextual Modeling 6](#_Toc335635325)

[QuickRDA: RDA Conceptual Modeling 6](#_Toc335635326)

[Approaches & Techniques 7](#_Toc335635327)

[Composition & Decomposition 7](#_Toc335635328)

[Abstracting Up 7](#_Toc335635329)

[Refactoring 8](#_Toc335635330)

[Specific Layer Modeling Documents 9](#_Toc335635331)

[Contextual Modeling Guide 9](#_Toc335635332)

[Conceptual Modeling Guide 9](#_Toc335635333)

[References 10](#_Toc335635334)

[QuickRDA: Introduction & Overview 10](#_Toc335635335)

[QuickRDA: RDA Domain Language 10](#_Toc335635336)

[QuickRDA: RDA Contextual Modeling 10](#_Toc335635337)

[QuickRDA: RDA Conceptual Modeling 10](#_Toc335635338)

[What Makes Good Architecture 10](#_Toc335635339)

***Table of Tables***

*No table of figures entries found.*

***Table of Figures***

***No table of figures entries found.***

# Introduction

This guide describes the modeling methodology of Role-based Domain Architecture, RDA. It offers insights into the methodology free of reference to specific tooling, so is applicable when RDA is a mind and paper exercise as well as when it is supported with other tooling.

Role-based Domain Architecture is named after the top layer, which captures architecture using roles and responsibilities for the abstractions. As the name also implies, RDA is for capturing domain design, not platform design.

We can consider the concepts in RDA both in the small and in the large. In the small, we are concentrating on the individual elements of the graph itself, where each concept is represented by a node or edge.

By contrast, in the large we are concentrating less on the individual graph elements and more on the meaning of the connected items taken together as a whole; when doing this, we must consider not just a single node but several nodes and some of their connecting relationships.

For example, one role is a single named node in the graph, so is one responsibility. However, when we talk about role in the large, it is more than a single node in the graph: we also think of the responsibilities assigned to that role. Similarly, when we think about a responsibility in the large, we consider not just that one node, but also the artifacts it provides and consumes in interacting with the other roles.

While the QuickRDA: RDA Domain document concentrates on the view in the small, this document looks more at the meaning of the information being captured via the network of nodes and edges.

## Architectural Artifact Alignment

Ordering is necessary for any serialization, such as this document. However, the ordering of topics presented herein should not be construed as ordered steps that the practitioner must follow. What is important — instead of following a prescribed order — is an outcome of aligned architectural artifacts.

*(Note that for these purposes, we consider the notion of aligned artifacts to include code & data, as well as architectural and design artifacts.)*

This notion of alignment is critical; alignment is concept that supports iterative development as well as the application of resources in parallel. Our notion of alignment is a formal and architecturally-based concept, which serves to replaces the waterfall notions of gives & gets, of inputs & outputs, of prerequisites, of ordering of the generation and development of artifacts.

Alignment is based on the presence of necessary overlap between artifacts; such overlap is present in good architecture, serving to connect the dots. For more information on this subject, please see the white paper QuickRDA: RDA Domain Language

##### QuickRDA: RDA Contextual Modeling

##### QuickRDA: RDA Conceptual Modeling

What Makes Good Architecture.

## Approaches & Techniques

Some of following approaches are written with roles, responsibilities, and artifacts of contextual architecture in mind; however as general approaches, they can be applied to the other layers as well.

### Composition & Decomposition

RDA’s models support several notions of composition and decomposition. Between each layer, refinement supports composition and decomposition to other layers of abstraction.

Within a layer, the modeling supports other kinds of composition and decomposition, namely like and unlike composition and decomposition.

As an example of like composition and decomposition, the contextual layer allows roles composing and decomposing into other roles; it allows responsibilities composing and decomposing into other responsibilities, and, it allows artifacts composing and decomposing into other artifacts. With like composition and decomposition, RDA provides more than just hierarchy as there are explicitly named and multiple are possible (not just parent-child).

As an example of unlike composition and decomposition, the contextual layer allows decomposing a role into responsibilities, and a responsibility into the providing and consuming of artifacts exchanged with others — and vice versa for composition.

For good architectural clarity, and capture of how the roles relate to responsibilities and how those relate to artifacts, we need to capture role to role relationships, artifact to artifact relationships, and responsibility to responsibility relationships in addition.

For instance, differentiating between (rather than conflating) several similar, related artifacts gives us a better selection from which to choose for describing what artifact a responsibility provides or consumes; this in turn increases the clarity of the capture of the responsibility.

### Abstracting Up

Abstracting up is the term we use for getting to higher levels of meaning.

In establishing layering we say that an upper layer relative to a lower layer can be thought of as the *what* relative to the *how*; they are linked by *why*. Yet these are relative labels. Any given layer is the *what* to the layer below and *how* to the layer above. In order to abstract *down* — to give more detail — we perform an exercise of thinking of something as the *what* then considering the *how* (according to *why*). We can repeatedly apply this pattern to expose more detail: changing focus and abstracting down, we switch the though label of a given layer from *how* to *what* — making room for another *how* and more detail.

Abstracting *up* is the reverse: consider something as the *how* that might have previously been thought of as the *what*, and reason as to *what* relative to this *how*. In doing this we get to a truer meaning of intentions and semantics.

Because of the abstract nature of contextual architecture, *we will want to keep this notion of abstracting up constantly in mind*. RDA supports the notion of abstracting up within contextual architecture in part via the composition and decomposition features available for roles, responsibilities, and artifacts. The various layers of RDA also cater to abstracting down and up.

### Refactoring

Refactoring is the technique by which developers make changes to their architecture and code that allow further feature expansion in a desired direction — without actually introducing any new functionality, just changes that increase flexibility. This goes to separating concerns of expansion enablement from expansion itself.

In other terminology, the increased flexibility improves architectural quality and reduces technical debt — without adding new features. In theory, when refactoring is performed within a domain model it can be done without breaking any tests. The value of this is that the effort to add a new feature can be decomposed into two smaller portions: first, a design refactoring with attendant changes to implementation — followed second by feature expansion with attendant changes to implementation. This separation of concerns reduces overall complexity by allowing designers to focus on these concerns independently; this in turn improves testability, scheduling, and confidence. Any decomposition of work into smaller, individually complete-able and individually testable units pays off.

RDA supports the notion of refactoring in part via the composition and decomposition features found for roles, responsibilities, and artifacts; in general refactoring is also encouraged between the layers: for example, working on conceptual service architecture may reveal otherwise unaddressed articulated contextual roles, responsibilities, or artifacts.

# Specific Layer Modeling Documents

## Contextual Modeling Guide

See QuickRDA: RDA Contextual Modeling.

## Conceptual Modeling Guide

See QuickRDA: RDA Conceptual Modeling.

# References

##### QuickRDA: Introduction & Overview

##### QuickRDA: RDA Domain Language

##### QuickRDA: RDA Contextual Modeling

##### QuickRDA: RDA Conceptual Modeling

##### What Makes Good Architecture