QuickRDA

Introduction & Overview

Erik Eidt

Version 3.2

August, 2011

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

[Introduction 5](#_Toc336066963)

[Overview of Documentation for QuickRDA 6](#_Toc336066964)

[RDA Domain Languages 6](#_Toc336066965)

[RDA Modeling Guide 6](#_Toc336066966)

[Modeling System Installation Guide 6](#_Toc336066967)

[Modeling System Diagram Generation Guide 6](#_Toc336066968)

[Domain Fact Entry Form User Guide 7](#_Toc336066969)

[Metamodel & Domain Fact Entry Form Developer Guide 7](#_Toc336066970)

[Domain Model Interchange 7](#_Toc336066971)

[Overview of Role-based Domain Architecture 8](#_Toc336066972)

[Architectural Foundation 9](#_Toc336066973)

[RDA Notion of Architectural Description 10](#_Toc336066974)

[RDA Notion of Layering 11](#_Toc336066975)

[Mapping between Adjacent Layers 12](#_Toc336066976)

[Layering as a Tree 12](#_Toc336066977)

[RDA Layering 14](#_Toc336066978)

[Simulation & Execution of Contextual Models 16](#_Toc336066979)

[Overview of QuickRDA Tooling 17](#_Toc336066980)

[General Concepts 17](#_Toc336066981)

[Fact 17](#_Toc336066982)

[Source Unit 18](#_Toc336066983)

[Domain Language 19](#_Toc336066984)

[Domain Model 19](#_Toc336066985)

[View 20](#_Toc336066986)

[View Specification 20](#_Toc336066987)

[Diagram 21](#_Toc336066988)

[Report 21](#_Toc336066989)

[Modeling Foundation 22](#_Toc336066990)

[Tools & Status 23](#_Toc336066991)

[Potential Next Steps 23](#_Toc336066992)

[Other tools & Research Opportunities 24](#_Toc336066993)

[References 25](#_Toc336066994)

[TOGAF 25](#_Toc336066995)

[RDA Introduction 0.71, PowerPoint, 25](#_Toc336066996)

***Table of Tables***

[Table 1. Current Tooling 23](#_Toc336066997)

[Table 2. Future Toolset 23](#_Toc336066998)

***Table of Figures***

[Figure 1. Architectural Description 10](#_Toc336066999)

[Figure 2. Architectural Layering 11](#_Toc336067000)

[Figure 3. RDA Domain Model 14](#_Toc336067001)

[Figure 4. The Domain Model 17](#_Toc336067002)

[Figure 5. Our Formal Modeling Foundation 22](#_Toc336067003)

# Introduction

Role-based Domain Architecture, RDA, is an emerging methodology for capturing architectural description, for decomposing problem spaces, and for creating service oriented architectures.

Just like other methodologies, such as TOGAF, RDA’s purpose is to capture, communicate and share architecture, and facilitate its development. However, unlike TOGAF, RDA’s architectural capture begins not with business processes within an enterprise, but instead with parties that interoperate in an ecosystem playing roles that interact via artifacts in carrying out responsibilities.

The methodology of modeling according to Role-based Domain Architecture is supported by tooling that is light weight yet formally grounded; this tooling is called QuickRDA, and the primary tool in the set is the *QuickRDA Modeling System*.

This document provides an introduction and overview of each of the following:

* the set of documentation on QuickRDA,
* the methodology of Role-based Domain Architecture, and,
* the current toolset for QuickRDA.

At present, the QuickRDA Modeling System is the presently the primary tool in the chain (though others are in the works). The modeling system is used to capture a domain model, from which it can generate a variety of diagrams from varying viewpoints, perform analysis and generate reports, and export model content. While graphing is supported by the modeling system, graphical editing of models is not, at present.

*A web-based tool for graphic editing of models and dynamic collaboration is also being developed; though not documented as yet.*

# Overview of Documentation for QuickRDA

The documentation set for QuickRDA and its tooling consists of the following:

1. Introduction & Overview — *this document*
2. RDA Domain Model Definition
3. RDA Modeling Guide
4. Modeling System Installation Guide
5. Modeling System Diagram Generation Guide
6. Modeling System Domain Fact Entry Form User Guide
7. Modeling System Metamodel & Domain Fact Entry Form Developer Guide
8. Domain Model Interchange

## RDA Domain Languages

A domain language is a set of concepts and relationships for formally modeling a domain. RDA’s domain language is layered, with each layer addressing a closely related set of concerns.

The RDA Domain Languages lists and briefly describes the concepts and relationships used in Role-based Domain Architecture. (Internally, domain languages are also captured as models, making them metamodels.) This document describes the specific concepts and relationships; it is rather dry and doesn’t cover the approach and best usage, which is the subject of the next document.

## RDA Modeling Guide

The Modeling Guide discusses how to use the domain model definitions to capture domain models and includes discussion on carrying out the methodology. It describes the approach to modeling using the concepts of RDA, free from discussion of tooling usage.

## Modeling System Installation Guide

The Modeling System Installation Guide describes the operational environment of the QuickRDA modeling system tool, followed by its installation. The documentation covers installation of the QuickRDA tool itself, along with necessary open-source software for generating and viewing RDA diagrams.

## Modeling System Diagram Generation Guide

The Modeling System Diagram Generation Guide describes the basic operation of this QuickRDA tool, in particular, documenting the Build Table and its options for building diagrams and producing reports.

## Domain Fact Entry Form User Guide

The Domain Fact Entry Form User Guide describes the provided Domain Fact Entry Form for capturing RDA models using declarative tables found in the standard Domain Fact Entry Form. The document describes the meaning and use of the columns in those tables, and routine customization of the Domain Fact Entry Form by hiding, deleting or rearranging the provided columns.

## Metamodel & Domain Fact Entry Form Developer Guide

The Metamodel & Domain Fact Entry Form Developer Guide describes the formal foundation of the QuickRDA toolset and one of its mechanisms for extensibility.

The QuickRDA toolset comes with domain metamodels for the 4 layers of RDA, along with 3 metamodels that map between adjacent layers.

Defining additional metamodels that build on or extend RDA is possible, as is defining other metamodels wholly unrelated to RDA. Domain metamodels are expressed declaratively using tables and in terms of the underlying metamodel.

In order to make use of new metamodels, Domain Fact Entry Form work is necessary, which can take the form of augmentation of the standard RDA Domain Fact Entry Form as well as creation of new Domain Fact Entry Forms.

The Metamodel & Domain Fact Entry Form Developer Guide describes the specification of custom metamodels and Domain Fact Entry Forms.

The underlying metamodel itself is captured both in a declarative table and also within the code of the QuickRDA tool; customization of the underlying metamodel is beyond the scope of what the toolset supports.

## Domain Model Interchange

The Domain Model Interchange document specifies the format for sharing models and changes to models called the Unit of Interchange, which is a data container that wraps model facts providing a context for their interpretation. Domain Model Interchange (DMI) is based on the QuickRDA underlying metamodel. DMI can be used to support manual import & export, automated exchange, and collaboration. Additional tools can be added to the QuickRDA toolset by consuming or producing this format.

# Overview of Role-based Domain Architecture

RDA’s goal is to capture domain architecture for modular service-oriented architecture in a multi-party ecosystem. This results in specifications that support the distributed development necessary for modern, loosely-coupled enterprise services as well as larger ecosystems involving multi-tenant Software as a Service delivery models.

These specifications are capture in models. An RDA model is light weight yet formal.

This document provides an *informal* overview of the RDA layering. Rigor is left to the subsequent document RDA Domain Languages.

This document continues with informal discussion and background of the layers.

## Architectural Foundation

RDA defines specific *abstractions* as the building blocks of information capture though modeling. To abstract means the hiding of detail; we use it to mean the hiding of *irrelevant* detail to a particular set of concerns so that we can focus on those concerns. A specific set of such concerns, isolated from the specific detail of a particular case and called out with formal definitions, are RDA’s abstractions.

RDA’s abstractions are comprised of *concepts* and *relationships*. In general terms, an RDA model is a graph, which is composed of *nodes* connected by *edges*, or in alternate popular terminology, *boxes* or *shapes* connected by *arrows* or *lines*, wherein, the nodes represent *concepts* and the edges represent *relationships*. Both concepts and relationships in RDA are always explicitly identified, in other words, named, titled, or labeled — in particular, this means that RDA does not use unlabeled or implicitly defined relationships.

For example, RDA can describe some role, say *General Contractor*, and some responsibility, say *Compete for Bids* that is *assigned to* the role; this would constitute two concepts and as a relationship connecting them.

Since in RDA, relationships are also concepts (i.e. a relationship is a form of concept) for brevity we can collectively refer to concepts and relationships as just concepts. In the rare case that we need to refer to concepts that are not relationships, this literature will use the term entities and parenthetically refer to them as concepts that are not relationships.

RDA uses a layered architecture; each layer focuses on a different set of concerns that, taken top to bottom, successively provide increasing design detail. Each layer does this using a different, well-defined set of abstractions at its foundation.

RDA consists of four layers of architectural specification: Contextual, Conceptual, Logical, and Physical, in top to bottom order. These layers support concepts ranging from business parties interacting in an ecosystem at the top to standards and technology choices at the bottom.

RDA uses *refinement*, which is the ability to decompose a concept into further detail — or in reverse, concepts can compose to higher level of detail.

### RDA Notion of Architectural Description

In RDA, the kinds of concepts and relationships available for modeling are organized into closely related groups called *architectural descriptions*. An architectural description is a single *layer* in RDA’s multi-layer capture.

RDA architectural descriptions describe *entities* that interact with each other by sharing some *content* according to some connecting *relationships*. RDA architectural descriptions are used in layers, described next.

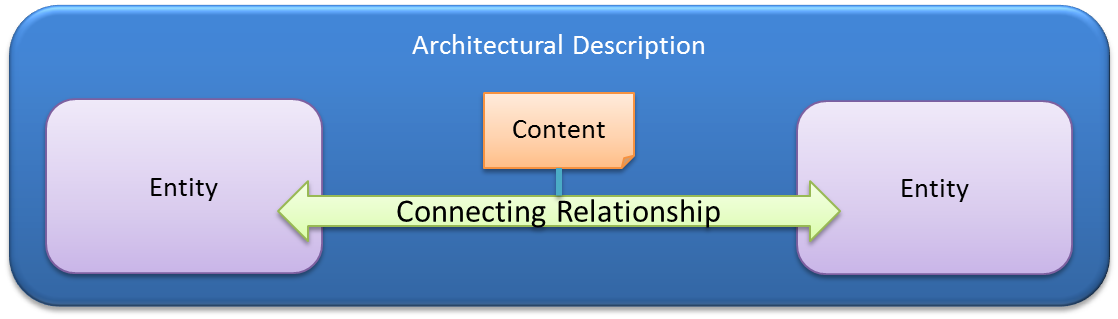


Figure 1. Architectural Description

Refinement occurs within each layer: concepts compose and decompose in terms of both like and related concepts. As examples of like composition & decomposition, roles can compose & decompose into more roles, and, artifacts can compose & decompose into other artifacts. *(This is done using explicitly identified relationships in contrast with hierarchical composition that uses implicit relationships).* As examples of related composition & decomposition, a role can decompose into its responsibilities; a responsibility can decompose into the artifacts it provides and consumes.

### RDA Notion of Layering

In the abstract, the relationship between adjacent layers in RDA is shown here:

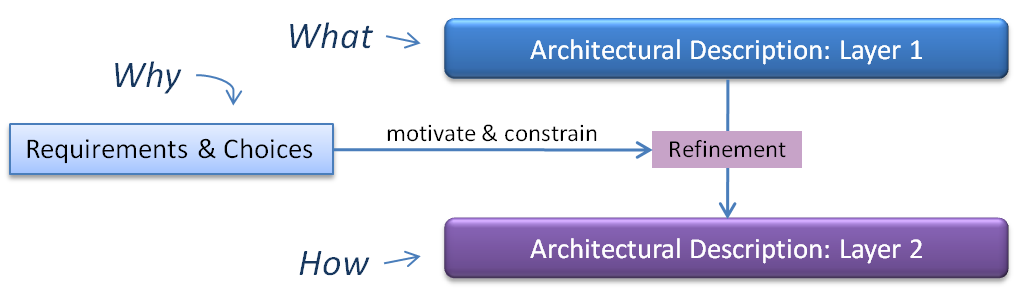


Figure 2. Architectural Layering

Each layer has a particular set of closely related concepts and relationships, which are abstractions that allow focus on a particular set of concerns ranging from:

* business context free of automation and systems; to
* conceptual services in a modular service-oriented architecture; to
* logical service designs, interfaces and interaction patterns; to
* physical technology choices

In general, refinement describes the decomposition of design choices into further detail. Refinement can be hierarchical, by which we mean that relative to the concept being refined, the refinement decomposes a concept into smaller concepts of the same type (e.g. a role Plays one or more other roles; a responsibility has a parent responsibility), or, refinement can decompose a concept into one or more dissimilar types (e.g. a role has responsibilities, a responsibility provides or consumes artifacts).

In addition, RDA explicitly connects the layers. Since the concepts of one layer are different from the concepts of another layer, inter layer refinement will be of the dissimilar type: refinement will relate concepts in the upper layer to concepts in the lower layer.

QuickRDA tooling today uses light-weight yet formal modeling to capture:

* the “What” shown here as layer 1,
* the “How” shown here as layer 2, and,
* the refinement map between adjacent layers.

The reader may notice that the QuickRDA tooling does not capture everything in the above diagram, namely Requirements & Choices. Today, these are captured outside of QuickRDA tooling (sometimes in textual documentation). This includes *Requirements* that motivate *Choices* in refinement between layers (as well as the set of Requirements & Choices that motivate the top layer itself).

### Mapping between Adjacent Layers

Refinement also occurs between layers: concepts can decompose into further detail in lower layers (and in reverse, compose into higher level of detail in upper layers). A mapping layer captures this refinement between adjacent layers that describes how a modeled domain’s concepts in the upper layer are formally connected to the modeled concepts in the lower layer, ensuring consistency in the transition as the focus of concerns (the abstractions) shift between of the layers, and connecting the dots to reveal successive design detail from upper to lower layers.

Entities (concepts that are not relationships) of one layer can be refined by entities in another layer; relationships in one layer can be refined by relationships in another layer. In addition, however, it is also possible and useful, to have a relationship in one layer refined by an entity in the next layer. It is usual to have one-to-one (though reasonable to have many-to-many) relationships between concepts of the lower layer as refinements of the upper layer, and vice versa.

There is a notion of completeness that requires every element of the upper layer to be provided for in the mapping to the lower layer — and vice versa. The intention is that there are no concepts in the upper layer going unrefined in the lower — and, vice versa, that there are no added concepts in the lower layer that are not derived from refinement of upper layer concepts. In practice, a full assessment must take into account both concept refinements between the layers as well as the concept refinements (composition & decomposition) that occur within a layer.

For example, let’s say there is a concept X in an upper layer that is refined by to a concept Y in a lower layer. Y may participate in (intra layer) decomposition into Z, in such case, for the assessment of completeness, Z is implicitly refined by being a decomposition of Y, and, no explicit refinement relationship of Z to (concepts in) the upper layer is appropriate.

The requirement for completeness of mapping between layers (i.e. with no additions or extras unaccounted for in either layer) can also be thought of as constraining: an upper layer constrains a lower layer. Non-adjacent upper layers indirectly constrain lower layers; all layers constrain actual implementation outside of RDA. *(This is not to suggest a particular development order — indeed a lower layer can be developed first or in parallel as long as it aligns appropriately.)*

### *Layering as a Tree*

We often think of a domain model as comprising one model at each of RDA’s layers, and this is indeed the common case. However, given another set of requirements and choices another version of a given layer can be created, with both versions being consistent with adjacent upper layer, though not necessarily consistent with each other. The structure of the layers is really that of a tree growing from the root of a common top layer, and, the common case of a single “stack” is a degenerate version of this tree. This tree provides valuable leverage points for alternate, interchangeable “How”s for a given “What”. This mechanism provides opportunities to revision lower layers and actual implementation while remaining true to the upper layers.

As a simple example, a particular conceptual services model refining a given business contextual model may choose focus only on the consumer to business role interactions (for necessary completeness, declaring the other roles as not refined by conceptual services), while another conceptual services model may focus on supply chain interactions. With some refactoring, these two conceptual models could be merged into a larger conceptual model for the domain.

As yet another example, a physical technologies model may choose HTTP-REST-style messages for a given logical architecture, while another physical technologies model may choose SOAP-style messages for the same logical architecture. Implementations following one physical technologies model would not be compatible with those following the other. (Though again, through some refactoring, these physical technology models could be merged, implying that implementations need to support both REST & SOAP messaging.)

## RDA Layering

The following diagram shows an overview of the layered architectural descriptions in RDA. RDA consists of four layers of architectural description.

Below RDA’s four layers is implementation (code, data, etc…): RDA provides specifications for and constraints on implementation, the actual implementation is performed outside of RDA.

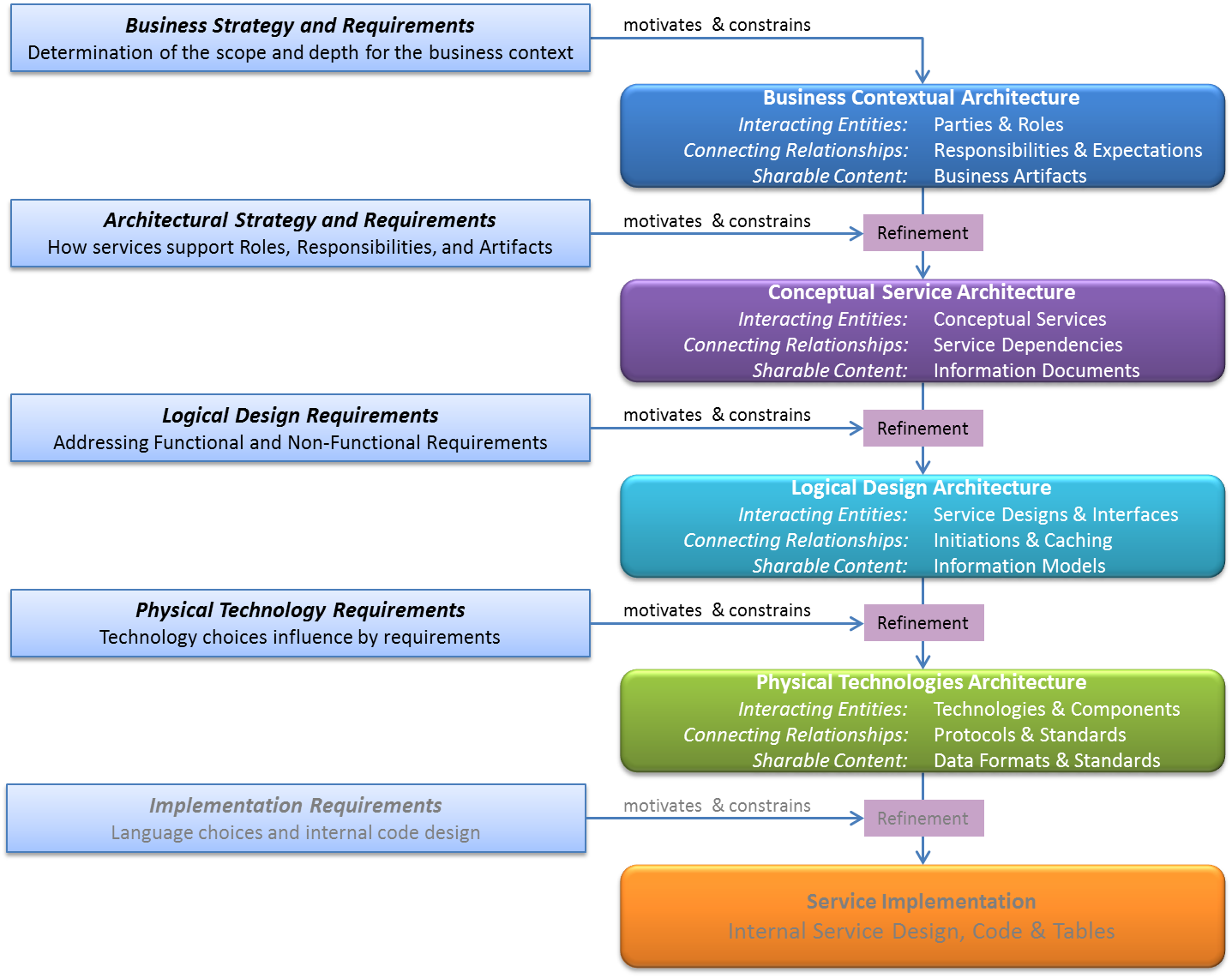


Figure 3. RDA Domain Model

As shown above, RDA domain modeling has business context at the top. The business context is for capturing a detailed, thorough, and formal understanding of the business *completely free of automation and IT systems*. This capture can be reasoned over for analysis or execution.

The subsequent lower layers focus on various levels of elaboration of services that will provide of automation support to the business context through modular service-oriented architecture.

Further description and usage of the layers is found in RDA Domain Languages and RDA Modeling Guide.

### Simulation & Execution of Contextual Models

RDA contextual models can be used for two purposes; one is in development of modular service-oriented architecture using the rest of the layers shown above here.

However, another potential use of contextual models is in communicating roles & responsibilities, such as in a project, an organization, group, or business. It is also theoretically possible to simulate execution of a business context model (though the QuickRDA modeling system does not itself support this, additional tooling could). Automation assistance could also be developed to support direct execution of contextual models; thus, facilitating the day to day communication and collaboration that occurs in carrying out roles & responsibilities, measuring results and identifying opportunities for organizational improvement. Using model by such automation, it should be possible to roll out changing roles & responsibilities more quickly within an organization.

# Overview of QuickRDA Tooling

Modeling in QuickRDA is done in terms of source units used to capture facts. These source units all logically combine to produce what we refer to as a single domain model. A domain model can be filtered to show one or more different views.

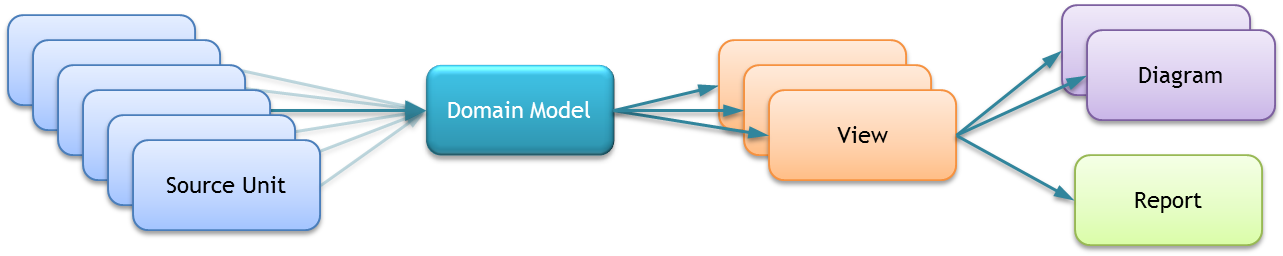


Figure 4. The Domain Model

To further understand the operation of the tooling, we defined some vocabulary.

## General Concepts

We define and differentiate the concepts of a ***Fact***, a ***Source Unit***, a ***Domain Mode Definition***, a ***Domain Model***, a ***View***, a ***View Specification***, a ***Diagram***, and a ***Report***.

### Fact

A fact, here a domain fact, is either the declaration of a domain concept or a domain relationship. These domain concepts and relationships are expressed in terms of a Domain Model Definition, e.g. a language, such as RDA’s Contextual, Conceptual, Logical, or Physical architectures (or the mappings between these layers). For example, a fact might establish the existence of a particular role, say Customer. Or, a fact may establish a relationship between two roles, such as that one role is a member of another role.

The available concepts and relationships for the modeler to use are defined in the RDA Domain Languages document. The RDA Modeling Guide document discusses usage of these concepts and relationships.

### Source Unit

Facts are grouped into source units. Source units are used for organizing facts. Any number of source units may be used to describe a domain model.

Source Units can come in a variety of kinds. One such kind is the **Domain Fact Entry Form**, which is a customizable table in an Excel spreadsheet.

In the QuickRDA modeling system, a source unit is a single table, tab or worksheet in Excel or .csv .

Source units can be in multiple sheets or tabs within a single spreadsheet file and also spread across multiple spreadsheet workbook files. There are several strategies for choosing what facts belong in what source unit, though these are beyond the scope of this introduction.

For more information about using and working with source units, see the Modeling System Diagram Generation Guide document, which discusses how to work with source units to create views for producing diagrams and reports.

### Domain Language

The domain language is also a model (it is a metamodel), one that defines the RDA languages themselves, describing the kinds of concepts and their possible relationships as the available abstractions for domain modeling. For example, the domain model for RDA’s Contextual layer defines the concepts of Role, Responsibilities, Artifacts, and their possible relationships, such as when a responsibility is Is Assigned To a role, or a responsibility Consumes an artifact.

The metamodel for RDA is defined in the RDA Domain Languages document.

### Domain Model

Modelers and architects decide what constitutes the ecosystem they are modeling, the domain of interest: that is, what the scope of the domain is: what’s considered in the domain, what’s not. (This scoping might be understood up front, or may be iteratively refined during the modeling of the domain.)

A Domain Model is a model of a given ecosystem. A domain model consists of all of the facts we have chosen to capture, modeled using metamodel concepts and relationships.

Ideally, all of the facts that describe an ecosystem interconnect; in [graph theory](http://en.wikipedia.org/wiki/Graph_theory) we’d say that the domain model is a single connected graph, that logically there is one single domain model for the given ecosystem, and, that by design, a domain model for a given ecosystem can be analyzed and reasoned over as a single model. During iterative development of a domain model this ideal condition of a single connected graph doesn’t always hold true. For example, when a bottoms-up development approach is involved in modeling the domain, then various pieces may initially not all fully connect. Such a condition of disconnected sub graphs simply indicates the domain model is incomplete rather than that the given ecosystem is composed of multiple domain models.

*(Note that having a single connected graph is not the only measure of completeness or well-formed-ness of an RDA domain model — other criteria range from subjective to objective, for example: a domain model must cover the intended ecosystem (fairly subjective); in the contextual layer, all artifacts that are provided by responsibilities must be consumed by some other responsibility, and vice versa (fairly objective).)*

We may also speak independently of models at RDA layers, e.g. a contextual model or a conceptual model. Though these layers may be spoken of and even initially developed as independent models, they are technically subsets of the larger, complete domain model, which has subsets for the layers along with mappings connecting them.

Capture of domain models is discussed in RDA Modeling Guide; also see the Domain Fact Entry Form User Guide for the more mechanical discussion of how to use spreadsheet tabs to capture facts.

### View

A view is a subset of the domain model. The modeler-chosen subset most likely focuses on a particular aspect of a domain model, such as a single layer of RDA, or a sub-portion of a single layer of RDA. Of course, a slice cutting across several layers is also reasonable.

For more information about using and working with views, see the Modeling System Diagram Generation Guide document, which discusses how to work with source units to produce diagrams and reports.

### View Specification

A view specification tells the QuickRDA modeling system how to subset the domain model. There are two approaches to the subset specification.

The first approach is to explicitly include (or omit) source units in defining the view This approach has the merit of simplicity, though doesn’t scale well for modelers, since using this technique, the larger the domain model the more source units needed; and the decision of which source unit to use to hold which facts becomes burdensome. *(Note: for modelers familiar with original versions of the QuickRDA tool, this was the only approach available; however additional capabilities have been developed enabling another approach.)*

The second approach is the use of *advanced filtering expressions* and/or *abstraction expressions*, which allow the view to focus on a particular subset of interest, essentially removing facts that are not part of that focus.

Advanced filtering expressions are a query language (having some analogies with SQL) that allows filtering a view into a smaller view by keeping only chosen facts, for example, focusing only on the interactions between two chosen roles.

Abstraction Expressions provide a mechanism to make the graph simpler, by altering and removing. For example, in the business context layer, roles have responsibilities that exchange artifacts. In a view, the artifact exchanges can be shifted from the responsibilities to the roles, and the responsibilities eliminated. Thus, the view has roles exchanging artifacts (without responsibilities). *(Currently there are a number of build switches for abstraction of the business context layer. In the future, we hope to provide a general purpose abstraction language instead of adding more switches, e.g. for the other layers.)*

The approaches of source unit inclusion, advanced filtering, and abstraction may be combined: multiple chosen source units selected to form a (larger) view, and then filtered or abstracted to another (smaller) view.

*(Some view specifications are analogous to a viewpoint in other modeling systems, in that the view specification can be applicable to or across more than one domain model. Other view specifications contain model-specific references (e.g. for filtering) and are thus not generally applicable across other domain models.)*

For more information about using and working with views, see the Modeling System Diagram Generation Guide document, which discusses how to work with source units to produce diagrams and reports.

### Diagram

A diagram is the graphic rendering of a view (which is of a subset of the domain model).

The QuickRDA tooling supports a number of different rendering options. as an example, the diagram can be flowed top-to-bottom or left-to-right.

As another example, a number of concepts in the RDA layers have a notion of containment; those can be rendered visually showing the containment or not. When so visually rendered, contained concepts are drawn within the boundaries of containing concepts — and the containment relationship itself, the labeled edge (i.e. arrow or line), is then omitted. When not so visually rendered, then the containment relationships are shown the same as others, using labeled edges (i.e. arrows or lines).

(There is also an option that renders containment visually while keeping the labeled edges; this and other options can be helpful in debugging domain models.)

For more information about using and working with views, see the Modeling System Diagram Generation Guide document, which discusses how to work with source units to produce diagrams and reports.

### Report

A report is an assessment or evaluation of a particular view. Since a report is generated from a view, the report may be generated for a subset (as specified by the view) of the domain model. Generally speaking, however, since much of the analysis in the report is based on notions of completeness of the domain model as a whole and of the individual RDA layers. When given a subset of the model to analyze, there is likelihood that it will complain about incompleteness of various aspects of the model. Thus, results from the report are more meaningful when analysis is supplied with the whole domain model (or at least a whole layer) by including all the source units a model.

For more information about using and working with views, see the Modeling System Diagram Generation Guide document, which discusses how to work with source units to produce diagrams and reports.

Analysis of the domain model is an area that has much potential for research and development. Though currently there are no switches for tuning the report analysis, additional reporting capabilities are likely to emerge.

### Modeling Foundation

The following diagram shows the formal modeling foundation. Each successively higher layer introduces a set of closely related concepts and relationships that are defined in terms the concepts and relationships of the layer below.

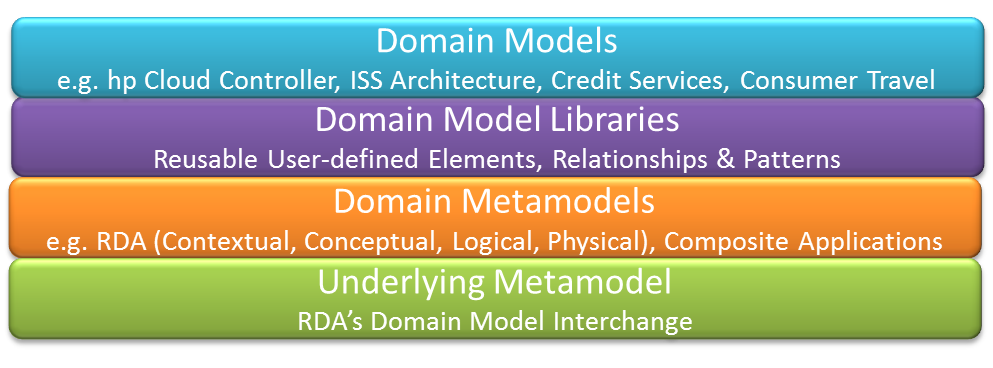


Figure 5. Our Formal Modeling Foundation

## Tools & Status

The primary tool in the toolset for QuickRDA is the QuickRDA modeling system. We’re also working on a web-based tool more suited for collaboration & versioning of the model.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tool** | **Version** | **Input** | **Source Inclusion** | **Filtering** | **Abstraction** | **Diagramming Options** | **Output** | **Report** |
| QuickRDA  Modeling  System | 2.4 | .xlsx .csv | Build Table | | | | .dmi  .svg | .xlsx |
| Web Tool |  | Graphical User Interface | | | | | |  |

Table 1. Current Tooling

## Potential Next Steps

Domain Model Interchange, DMI, defines a Unit of Interchange for exchanging models and model subsets (views), along with support for versioning and change proposals. DMI has a file format using the .dmi extension. Units of Interchange captured in .dmi files can serve as another kind of source unit.

The following illustrates other potential tooling for an extended tool chain, based on .dmi as the source unit instead of spreadsheet based tables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tool** | **Input** | **Filtering** | **Abstraction** | **Diagramming Options** | **Output** | **Report** |
| Modeling &  Collaboration | .dmi | Graphical User Interface | | | .dmi |  |
| Merge &  Filter | .dmi | Command Line | | | .dmi |  |
| Report | .dmi | Command Line | | |  | .xlsx |
| Diagramming | .dmi | Command Line | | | .svg |  |
| QuickRDA  Modeling  System | .xlsx  .csv | Build Table | | | .dmi  .svg | .xlsx |

Table 2. Future Toolset

A modeling tool might be web-based or a client application. Collaboration support involves storing or archiving of previous versions and support for in progress proposal for change.

Merging is the capability to combine views into a larger unit of interchange. Collaboration features involve manipulation of versioning information and change proposals.

Report generation analyzes domain models.

Diagramming provides publishable drawings based on the domain model.

And both the QuickRDA modeling system as well as the current web-based tool could continue to participate in this tool chain.

## Other tools & Research Opportunities

Automation support for direct execution of Contextual Models could leverage the contextual capture for (office) productivity support instead of service design.

Research in analysis and assessment of domain models is an area that is rich with opportunities.

# References

##### [TOGAF](http://en.wikipedia.org/wiki/TOGAF)

##### RDA Introduction 0.71, PowerPoint,