QuickRDA

RDA Contextual Modeling

Erik Eidt

Version 3.0

May, 2011

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

[Modeling Business Contextual Architecture 5](#_Toc335635426)

[What RDA Business Contextual Architecture isn’t About 6](#_Toc335635427)

[Concerns from Lower RDA Layers 6](#_Toc335635428)

[Detail that doesn’t change the Patterns of Interaction 6](#_Toc335635429)

[Offering Architecture 6](#_Toc335635430)

[Top-Level Characterization 7](#_Toc335635431)

[External Characterization 8](#_Toc335635432)

[Outsourcing 9](#_Toc335635433)

[Concept Overview 10](#_Toc335635434)

[Roles 10](#_Toc335635435)

[Artifacts 10](#_Toc335635436)

[Responsibilities 11](#_Toc335635437)

[Layering of Roles 12](#_Toc335635438)

[Generic Roles in Opposition — Generic Pairs 13](#_Toc335635439)

[Domain-Specific Roles in Opposition — Domain Pairs 14](#_Toc335635440)

[Composite Roles — Domain Role Players 15](#_Toc335635441)

[Middlemen 16](#_Toc335635442)

[Abstract Domain Parties 17](#_Toc335635443)

[Domain Actors — Concrete Parties 17](#_Toc335635444)

[Plan Linkage 17](#_Toc335635445)

[Artifacts 18](#_Toc335635446)

[Promises & Persistence 18](#_Toc335635447)

[Components & Views 19](#_Toc335635448)

[Kinds 19](#_Toc335635449)

[Level of detail 19](#_Toc335635450)

[References 20](#_Toc335635451)

[QuickRDA: Introduction & Overview 20](#_Toc335635452)

[QuickRDA: RDA Domain Model Definition 20](#_Toc335635453)

[What Makes Good Architecture 20](#_Toc335635454)

***Table of Tables***

*No table of figures entries found.*

***Table of Figures***

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

# Modeling Business Contextual Architecture

The business contextual architecture is one the most significant parts of Role-based Domain Architecture. Because its focus is purely business, free of the information systems that so many of us are used to dealing with, it is also arguably the most abstract and difficult.

A RDA Business Contextual Architecture will articulate:

* Roles having responsibilities interacting via business artifacts
* A top-level, external characterization of the domain: an ecosystem of peers
* Artifacts — and artifact to artifact relationships independent of roles & responsibilities
* Layered roles — roles composed using role to role relationships
* Responsibilities connecting the roles & artifacts — using decomposition as well

## What RDA Business Contextual Architecture isn’t About

### Concerns from Lower RDA Layers

We’ve said before that to understand this given layer, we also need to understand what’s not in the layer. This is important because each layer is intended to focus on its closely related concepts without conflating concepts from the other layers, since they are separate concerns that are at different layers of abstraction. The business contextual layer should not incorporate concepts belonging to the conceptual service, logical design, or physical technologies layer. The contextual layer describes a business context free of any automation systems. The contextual model is applicable whether the domain uses paper and abacus or information systems!

Working at the contextual layer, when tempted to think about automated services or information systems, capture or jot down the concerns for later use, and then transform these thoughts into business-context capture by considering the business roles that might own the automation, by considering the responsibilities that are being assisted or carried out by these information systems, and by considering the business artifacts involved in the exchanges.

### Detail that doesn’t change the Patterns of Interaction

Business context is about capturing patterns of interaction between roles. It is meant to be specific and precise while remaining abstract. However, it is not for capturing all possible domain detail.

For example, if we can describe that two parties interact, one to purchase and the other to sell widgets, it isn’t important how much money and what kind of widget, provided the pattern of interaction is the same for all dollar values and all widgets. If the dollar amount (free or very large) or kind of widget (big or small) influences the pattern of interaction then we ought to capture this in the business context. Yet however, if for one of the parties an internal approval is necessary for a large dollar amount, this does not necessarily influence the pattern or shape of the interaction and thus most likely has no reason to be captured above and beyond the normal pattern. An external approval or confirmation may, however, change the interaction pattern or may be considered detail we don’t need to capture. One reason perhaps to capture such a confirmation could be that the confirmation is very important, even a legal requirement in the domain. Another reason is that the pattern of interaction will require additional information services to be designed (though their existence and design is for the other layers of RDA below contextual).

### Offering Architecture

*At present, the capturing of offering architecture is not integrated into RDA, though this a work in progress being addressed.*

## Top-Level Characterization

The domain of interest to the modeler can cross corporate boundaries (many do) or organizational boundaries within a single enterprise or institution. Either way, a domain involves a number of interacting parties, various kinds of consumers and suppliers, etc... Some of the interacting parties are of greater interest because eventually we’re going to design information system for them.

None the less, the contextual domain model should capture all the roles in the ecosystem, both central and fringe — it’s purpose is the capture of context, which requires breadth. We’re interested in the way the central interests interact with all the other parties or roles *as peers* in the ecosystem whether or not we are creating information systems for them.

According to the RDA methodology, it is preferable to capture fringe roles whose responsibilities are not fully captured, rather than capture responsibilities of the central interests providing or consuming artifacts with no corresponding consumer or provider, respectively. *This is a restatement of the principle that roles do not exist in isolation.* This implies at least minimally capturing roles that we aren’t necessarily directly focusing on in order to be able to fully capture the domain model involving the roles that are of central interest.

For example, some supplier role may not be fully articulated; however, without the role parts would be consumed by the business of interest without being provided. A consumer role may not be fully articulated, however, without the role, the business of interest would provide artifacts that are not consumed.

This is the notion of capturing roles that act as peers forms what we call top-level capture and is one of the main objectives an RDA Business Contextual model.

In the refinement from contextual to conceptual, for completeness, we explicitly capture the fringe roles that are not participating in further refinement; this allows readers and reasoners to differentiate between roles consciously decided as such and incompleteness.

The notion of top-level capture goes hand-in-hand with the notion of external characterization described next.

## External Characterization

A goal of a contextual domain model is to remain neutral with respect to all possible concrete parties, individuals, or corporations that examples of the top-level roles in the ecosystem. This means we need to identify and capture abstractions rather than concrete entities, since they may change over time. While the approach taken can vary (e.g. starting from concrete and refactoring toward abstract or starting from the bottom and abstracting up to the top), the important thing is that the contextual domain model captures the ecosystem’s *external characterization*:

Roles are relative to each other; they assemble into ecosystems using responsibilities to exchange artifacts. Responsibilities (manifest in both description and in artifacts exchanged) reflect the expectations between the roles —the expectations that ecosystem has of the roles, that the roles have of each other. As such, we call this capture *external characterization*, in that what is being captured as responsibilities — even decompositions and sub decompositions of a responsibility —are part of and visible to the whole ecosystem; the contextual domain model captures external characterization.

By contrast, the opposite of external characterization is focus on the internal operation of specific, concrete party, which we call internal implementation. Internal implementation is sometimes very useful, though neutrality across multiple possible players of the role is sacrificed. *This is in part by definition of the approach of decomposing the implementation of a specific, concrete party, and, also in part that it is hard to identify truly party-neutral considerations that aren’t part of external characterization.* Internal implementation is an *optional* perspective of an RDA Business Contextual model: it is best done independently from the external characterization to keep these concerns separate.

## Outsourcing

In many domains, we will find various ways of contracting out business functions — outsourcing. Architects should consider whether outsourcing of roles and responsibilities should be captured, or not: this is a choice that depends on the domain and its scope and the intended use of the capture. During initial development descriptions of outsourcing are generally better be left out; they can be added later as necessary.

Also, sometimes we’ll discover that we change our minds about whether something is outsourcing or is really another top-level business role operating as a peer with the others in the ecosystem. Outsourcing vs. being a vendor may likely be a continuum rather than an absolute.

On one end of that continuum, outsourcing can be characterized as a choice that can be independently made per every party in the ecosystem that would or could play a given role in the ecosystem. For instance, if, practically speaking, each party can easily choose insourcing vs. outsourcing independently, then there is likelihood that the role that could be insourced vs. outsourced is true outsourcing — with the attendant recommendation of initially leaving this out of the domain capture.

As an example in the trucking domain, each trucking company has a practical choice of doing routine engine maintenance either in house or at a variety of service providers. Thus, the role of engine maintenance might not be captured in the trucking domain as a top-level peer role, at least initially, even though it is required for the organization to deliver value to their market. In this example, maintenance is more of an internal business function for the trucking company rather than a vendor or supplier functioning as a peer in the ecosystem.

By contrast, if the outsourcing choice is not available to be made independently broadly across the industry, then there is likelihood that such a role in the ecosystem (which may initially appear to some as outsourcing) is really one of the top-level, peer roles of the ecosystem. For instance, if most in the industry have a choice of outside vendors, yet find it impractical to choose an in house alternative, then these vendors probably don’t represent outsourcing, but rather suppliers in the larger ecosystem; this would imply considering these suppliers as peers in the top-level domain model.

As an example in the trucking domain, each trucking company has a choice in fuel vendors, but has no practical in house choice for the supply of fuel. Thus, in this example, fuel vendors are peers in the trucking domain.

## Concept Overview

### Roles

Roles are entities that can assume responsibilities; that can represent the assignable patterns of interactions between parties. Some roles will be concrete and some abstract. Roles don’t exist in isolation; they are a relative concept. We build on the relative notion of the roles as the underpinning of good capture of roles using layering. This is given further discussion in the section below, Layering of Roles.

### Artifacts

Artifacts are business items that can be:

* Real world items:
  + Tangible items such as widgets or money
* Information content items:
  + Documents, forms, or records, such as reports, submissions, tickets, trip and record locators
* Messages, which usually contains some kind of
  + request for change, or
  + indications of status,
  + and often containing some content that references real world items or other artifacts
* They can also describe outcomes:
  + Results, or business conditions, or decisions.

Generally speaking, we want to consider the most abstract form of business artifact that allows sufficient capture of the responsibilities and roles in the domain. For example, in many cases, the notion of money is sufficient: we don’t need to differentiate between forms of cash and credit card purchases for ordinary business domain capture. Of course, we will likely want this differentiation in a financial services domain, or, if somehow it changes the patterns of interaction between the parties.

Artifacts also benefit from layer or composition and decomposition. This is discussed further, below.

### Responsibilities

The purpose of responsibilities is to identify and capture what the role is really trying to accomplish; again, we reiterate that this is relative to the position of the role in its ecosystem.

However, since an RDA business context capture already includes the notion of exchanging artifacts we don’t need responsibilities that re-articulate this exchange. When tempted to describe a responsibility as merely consuming an artifact, or providing an artifact, we should ask ourselves the question: what is that role really trying to accomplish that results in it needing an artifact or generating an artifact. This extends to the sending or receiving of communication, as well. Our experiences show that we may need to iterate on this question several times regarding the capture of a single responsibility, as it can often takes a few tries before come to something resembling a real business responsibility. For example, capture of an HR department responsibility is better as “Promote Individuals” or “Exercise Corporate Talent Development Plan” than “Provide Promotion”, or “Send Email about Promotion”.

Responsibilities are not categories: they should read as expectations that the other roles have of each other rather than as category buckets.

## Layering of Roles

At its deepest conceptual foundation, the notion of a role is a relative concept: certain roles are strongly linked together forming a cooperating set. Within such a set, roles are decidedly asymmetric: we can easily differentiate between roles in the set because each role does different things — has different responsibilities. The roles work opposite each other; hence, they are roles in opposition. For example, a consumer role plays opposite a provider role forming such a set; a patient plays opposite a doctor forming.

The strong linkage between the roles in such a set is formed by the expectation that the roles have of each other, which are ultimately manifest in the exchange of artifacts between the roles. In RDA, these expectations are captured as *roles assigned responsibilities that provide artifacts to, and consume artifacts from, other responsibilities assigned to other roles*. Thus, each such set is composed of roles *connected* by responsibilities and artifacts.

For simplicity in the rest of this documentation, we’ll often loosely use the term *pair* or *role pair* as shorthand when we really mean two *or more* roles forming a set connected by working in opposition.

We build up from the deep foundation described above to create a framework that positions roles in layers, articulating how RDA supports composition and decomposition among roles. We will be describe the framework from the bottom up (most generic and abstract) to the top (most domain specific and concrete), because that order lends itself to defining each group in terms of the assembly of roles from the previous (lower) group.

*However, this presentation order is not necessarily to be taken as prescribing ordered steps that domain architects need to take in articulating the composition of roles for their domain — to the contrary, we often work both bottoms up and top down, iterating and refactoring until they meet.*

### Generic Roles in Opposition — Generic Pairs

This layer is very abstract; it introduces concepts that are applicable across all domains. Despite being so abstract, it brings common notions of roles familiar from daily life to bear for use at subsequent layers. However, also due to its abstractness, we treat this group as a mental or paper exercise: we don’t necessarily attempt formal capture.

Many of these generic roles in opposition can be seen as related; a number of them are forms of the general consumer-provider pattern. Each of the following bullets represents one set of generic roles in opposition:

* Consumer–Provider
* Contractor–Contractee
* Contractor–Subcontractor
* Mentor–Mentee
* Employee–Employer
* Employee–Manager
* Shopper/Browser–Offerer
* Buyer–Seller
* Publisher–Subscriber
* Coach–Team Member

While we have a preference for binary roles because our experiences show that they work well to capture our intentions, there is no limitation to binary roles; we can have three or more roles opposing. For example, we can split the Provider of Consumer–Provider into a Provider and a Shipper making Consumer–Provider–Shipper. We can split the Consumer of Consumer–Provider into Consumer and Payer, making Consumer–Payer–Provider. We can combine the two, as in Consumer–Payer–Provider–Shipper. However, we should keep this layer fairly focused, as it is the purpose of the subsequent layers to compose these simpler relationships into larger ones.

We keep this simple notion of asymmetric roles working opposite each other (pairs, if you will) in mind as we move to the description of the next layer.

### Domain-Specific Roles in Opposition — Domain Pairs

This group takes the concepts of generic pairs and casts them into the domain of interest to create domain-specific pairs. For example, from the air industry we have: travel marketer & air carrier operator; and from the credit industry: borrower and lender. These pairs of roles are given domain responsibilities that provide and consume domain artifacts that we will capture in the contextual domain model as the lowest in the layering of roles.

Once again, we encourage focus of this layer; composition is delayed for another layer. We don’t expect ecosystems to emerge from this layer (they will in start to emerge in the next layer). Instead, here, we’re creating the domain-specific building block roles for the next layer to compose with.

We can also create domain-specific versions of other opposing roles: construction contractor–homeowner; dealership–customer; travel shopper–travel seller.

In the framework of layering roles, *this is layer where we primarily capture responsibilities*, their assignments to roles, and their interaction via providing and consuming artifacts.

In capturing responsibilities, one thing to ask is what are the key decisions that a role is the authority for making? What are the long term persistent artifacts that this role is the authority or owner of record for? We ask these to better understand why other roles turn to this role; it is because they turn to the role for the artifact it consumes or produces.

### Composite Roles — Domain Role Players

This layer articulates chunkier roles that compose roles from the domain pairs as follows: a chunkier role composes one — *but not all* — of roles from some pair. A composite role does this *across several domain pairs*: a composite role is assembled by choosing one of the roles from each of *several* domain pairs.

This is the key to forming ecosystems via composite roles: these chunkier roles each compose more than one domain roles — each from a different domain pairs.

For example, a general contractor may compose:

* the manager role of employee–manager, plus
* the contractor role of contractor–contractee, plus
* the contractor role of contractor–subcontractor

It is composing one half of three different domain pairs. The other half of each of the pairs might be played oppositely by:

* the employee role of employee–manager by an employee of the general contractor,
* the contractee role of contractor–contractee by a homeowner, and
* the subcontractor role of contractor–subcontractor by another contractor

As a result, we have four composite roles: the general contractor, their employee, the homeowner, and another contractor. These four roles are more than just roles in opposition; they are bound together in a more complex ecosystem of interactions and dependencies.

Here is another example of composite roles from the concert domain, starting with two domain role pairs:

* Concert Shopper–Ticket Seller
* Concert Attendee–Event Provider

These can be assembled by three composite roles as follows:

* Ticket Middleman plays
  + Concert Shopper (to Event Host), and
  + Ticket Seller (to Concert Customer )
* Concert Customer plays
  + Concert Shopper (to Ticket Middleman), and
  + Concert Attendee (to Event Host)
* Event Host plays
  + Ticket Seller (to Ticket Middleman), and
  + Event Provider (to Concert Customer)

#### Middlemen

Middlemen are also captured as composite roles, as they require the machinery of this level for capture. We can describe this middleman role as playing a providing role opposite some consumer and playing the consuming role opposite some provider. To improve clarity of this situation, we can describe that certain consumers are limited in interaction to the middleman as provider, or that certain suppliers are limited in interaction to the middleman as consumer. These networks are elaborated in the capture using the notion that the middleman plays a role “In Plan” P, where P describes that network:

* The middleman plays a consumer role in plan P1
* The supplier plays a provider role in plan P1, linking the middleman and supplier,
* The consumer plays a consumer role in plan P2
* The middleman plays a provider role in plan P2, linking the consumer and middleman

### Abstract Domain Parties

This group has abstract domain business entities, like Airline and Bank.  Here, abstract corporate entities usually play more than one composite role: the Airline plays Travel Agent and Travel Operator; we can also describe independent travel agents, etc…  Sometime we also describe that the abstract Airline role plays its Travel Agent role in a specific plan, which allows us to limit/target the interactions to specific other roles in the same plan.

### Domain Actors — Concrete Parties

The top group is the set of actors, representing actual domain corporations (and individuals), like American Airlines, or Wells Fargo, which then play abstract domain entities.  These concrete actors may also play their abstract domain entities in a plan that links several of these roles together into a specific ecosystem.  (In this group, actual individuals (and sometimes corporations, too) can be represented by classes (e.g. the class of (this kind of) customer); classes allows us to get to the same concreteness i.e. down to individuals (and sometimes even corporations) without listing everyone.

### Plan Linkage

Several of the layer support linkage in a network or plan. This is useful to differentiate or limit the interactions of some parties or composite roles to specific other roles.

It also serves to call out different possible interaction patterns, which can serve as the basis for differentiated refinement to RDA’s conceptual service architecture. For example, one set of roles linked by a plan may be refined to a user interface supported by a service, whereas another set of the same underlying roles linked differently may be supported only by services with no user interface. Differently linked roles may be supported by different services altogether even though the underlying roles being composed are the same.

## Artifacts

Many of the artifacts relate to each other as kinds or types, or as components or views. Clarity around artifacts helps quality of domain capture, not just with respect to the artifacts themselves, but also the responsibilities which provide and consume artifacts: conversely, when artifacts are not well understood, responsibilities cannot be captured well.

Several concepts go hand-in-hand in understanding business artifacts. Usually several closely related artifacts can be identified and differentiated from each other. Clarity comes be identify separate artifact and capturing the nature of the relationship between such closely related artifacts.

### Promises & Persistence

We need to identify business artifacts that represent promises. Promises are business artifacts that have an implicit or explicit reference to another artifact that satisfies the promise, as well as carrying implicit expectation that the promise holder has of the promise maker for eventual satisfaction (of the promise). Tickets and record locators are great examples of promises. It is implicit that they were purchased with the idea of redeeming a specific promise, for example, by attending a concert or by passenger air travel. Promise artifacts *Refer To* other artifacts.

We need to identify the authorities of record and for what; the authorities of record and what they store are often closely related to promises. Being an authority involves responsibilities of the keeping of some long-term records (longer term than a message, at least), and the responsibilities, for instance, to respond to inquiries with respect to the status of the promises. For example, an online marketplace is responsible not just for taking orders, but for maintaining orders and responding to inquires about those orders. Responsibilities can *Manage* artifacts.

We need to identify input artifacts as part of a request (and output artifacts as response). For example, the information you supply in ordering – an order request – is not the same as the order itself — in part because the order persisted whereas the order request is not, and in part because the order contains a record locator that represents promise for satisfaction of the order (as well as agreement (promise) to pay for the order), whereas the order request itself does not. A response artifact is a *Reply To* a request artifact.

An update request is a form of request that also *Refers To* what is being updated.

*(Technically, the content of the order request is also different from the order request itself which is a message wrapping or form containing the order request content, though this distinction is not always significant to the shape of the roles and responsibilities, and thus the contextual capture.)*

### Components & Views

Artifacts can be more directly composed and decomposed. One artifact may be a *Component Of* another artifact, which implies that the former must exist in order for the latter to exist. One artifact may be a *View On* another artifact, which implies that the latter must exist in order for the former to exist.

### Kinds

One artifact can be a more general type; another artifact can be a *Kind Of* that other artifact. This can be helpful in expressing likeness among artifacts; sometimes we’ll say that a responsibility consumes a type, meaning any artifact that is of this type is acceptable.

### Level of detail

Capturing some relationships between artifacts is part of the proper understanding of the domain. However, one goal is to have a balance of detail between roles, responsibilities, and artifacts. Many comfortable with logical and physical design will have temptation to decompose an artifact into all its constituent parts right away in the capture of business context. This temptation should be resisted as follows: we should defer to lower layers any artifact decomposition that turns out as not having effect number of the roles and responsibilities — what artifact decomposition doesn’t affect the patterns of interactions between the roles should be omitted, i.e. that decomposition can be deferred to another lower layer, e.g. to the conceptual layer.

A simple example can be the notion of a mailing address: in some domains there is no need to understand the fields of an address in the capture of the business context because the individual fields (such as city, zip) are not involved in roles & responsibilities except as the address concept as a whole. Understanding these fields may very likely be necessary in lower layers of the domain architecture, and should be articulated at the level of the domain capture that requires it.

# References

##### QuickRDA: Introduction & Overview

##### QuickRDA: RDA Domain Model Definition

##### What Makes Good Architecture