Beyond Enterprise Architecture

Erik Eidt

Version 3.2

August, 2011

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

[Introduction 5](#_Toc322337256)

***Table of Tables***

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

***Table of Figures***

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

# Introduction

As time progresses, we see customers predictably continue to enlarge the scope of the problem domains they want to address using Information Technology (IT). Over the years, this scope has expanded from the rather narrow domains involving single applications, to domains of the enterprise involving numerous interacting applications, to domains of rich ecosystems involving many enterprises.

Years ago, any given problem domain a customer was facing could be addressed with a single application. Rarely is that the case anymore today, as customers have long since moved to the problem domain of the whole enterprise, whose complexity necessitates coverage with numerous applications. Ideally, these applications are non-overlapping and well integrated; when this is the case, this would imply an effectively designed service oriented architecture. All too often, however, the applications have each been designed and evolved independently, meaning they lack common service interfaces, have significant functional overlaps, and each assume blanket authority despite their various overlaps — all of which make it more difficult to force, tease, and fool them into cooperation.

Meanwhile, the advent of cloud offers customers new areas of promise and opportunity. One such area of opportunity is that of dynamically scaling in for cost and dynamically scaling out for dynamic load. Another area of opportunity is that of rich collaboration across multiple enterprises and user communities, which is of primary interest for this paper.

Ecosystems involve a community of users playing varied roles. Some of these roles are consumer roles having interests & desires. Other roles are supplier roles, whose interests are generally those of business. In interacting, these various roles exchange funds, goods, services, and information. The exchange of information is of particular interest, as information sometimes represents knowledge, collaboration, or promises, such as for future goods or services.

As an ecosystem example, let’s take the domain of concerts. We have many parties with varied interests. We have consumers whose interests are around attending and enjoy concerts. We have ticket agents whose interests are in distributing and selling concert tickets and possibly in advertising, notification of changes, refunds, etc... We have venue hosts, whose interests are in booking their venues with interesting concerts. We have performers whose interests are in performing, and, agents for the performers whose interests are in matching performers with venues. We have a variety of vendors, such as for food and beverages services. We have hospitality and transportation businesses (e.g. those running hotels and airlines). All of these roles have need for collaboration and interaction with one or more of the others.

As another adjacent example, let’s take the domain of air travel. We have airlines, whose interests are in operating air carrier services. We have passengers whose objectives involve travelling. We have airports, baggage handlers, federal regulators and authorities, state, and local agencies and authorities, etc…

While cloud offers the opportunity to address such ecosystems more comprehensively by bringing them all together, it also dramatically exaggerates existing problems. Increasingly complex problem domains requires correspondingly increasing rigor. This means we need rigor in our software designs. However, the very expression, capture, and understanding of the problem domain are quite challenging, and we have seen that done incorrectly leads to building software that doesn’t address market needs, which represents substantial failure at the scale necessary to build software addressing ecosystems. It is insufficient to model a problem domain merely as a list of requirements and use cases — instead we need rich models that capture interactions between varied parties involved in the ecosystem.

Our approach to designing ecosystem scale software is to increase design methodology, and starts not with software architecture, but first with understanding the problem domain, which we call the context. We formally identify and model roles and their interactions. We categorized these interactions with responsibilities, which we define as the expectations the various parties have of each other to fulfill their obligations. Individual interactions exchange business artifacts, which are funds, information, goods, or services. As mentioned above, the exchanges of information are of particular interest, since they can represent knowledge such as product reviews, shopping information, such as good travel destinations, or where to get the best prices; information can represent promises for future goods and services, such as a concert ticket, or an airline seat reservation. And of course, information is also of particular interest to IT systems.

Our formal modeling approaches identify roles by the responsibilities they have and connect them through identified exchanges of business artifacts, such as funds, goods, services, and in particular, information. Further, roles compose with other roles, responsibilities compose with other responsibilities, and business artifacts compose with other business artifacts. We label models captured using these concepts and relationships with the term contextual domain models. Having a contextual domain model is the first step in designing quality software services that addresses the complexities of ecosystem-scale problem domains.

Next: how contextual domain models lead to service oriented architectures.