Designing a system of microservices

<http://www.apiacademy.co/designing-a-system-of-microservices/>

**Written by: Matt McLarty | May 1, 2017 - 1:58 PM**

This is part two of a three-part blog series on visualizing microservices. See[*part one here*](http://www.apiacademy.co/visualizing-microservice-architecture/).

I’ve been working with a number of large organizations recently, helping them understand and realize the value of Web APIs and microservice architecture. In this work, I have seen these organizations struggle consistently with determining how to define optimal boundaries between their microservices. In the industry, this is a known problem but not a solved one. Although much has been written about the value of cohesive services and the power of bounded contexts, there appears to be a void in guidance on how to practically identify these.

One reason for this issue is that the people trying to determine service boundaries are technologists who are inherently looking for a technological solution but defining cohesive, capability-aligned service boundaries instead requires domain expertise. Fundamentally, this should be a modelling exercise that is independent of whatever technology is implemented under the covers. However, even those organizations that have consciously taken a technology-agnostic modelling approach have struggled with the process of defining service boundaries.

This has happened, in part, because the most popular software modelling methodologies often have implementation bias. For example, [domain-driven design](https://techbeacon.com/get-your-feet-wet-domain-driven-design-3-guiding-principles) (DDD) has a bent toward object-oriented programming and UML has a data modelling perspective. On top of all this, there is no industry consensus on how a microservice system model should be depicted visually—what components should be depicted, how relationships should be represented, what terminology should be used—let alone any tooling in the space.

Still, all hope is not lost. Some DDD concepts—most notably the idea of “bounded contexts”—have gained popularity and started the industry conversation on service boundary definition. Importantly, the higher-level abstractions in the DDD methodology—domains, subdomains, bounded contexts, aggregates, context maps—are technology-agnostic and model-centric.

Since the accepted approach to understanding a system is to focus on the [relationship between its components](http://watersfoundation.org/systems-thinking/definitions/), we need not go deeper than the context map if we want a basic representation of a microservice system. So, perhaps we can use the DDD context map as a starting point for visually representing a system of microservices.

Even at the context-mapping level, a large organization’s complete system of services would be incomprehensible. So, in order to make the context map useful, it is important to have a context for the context map itself. This context could be the decomposition of a particular monolithic application or the service interactions of a particular initiative. The only way to coherently build a large organization’s system of microservices is to do so piece by piece, context by context. Along the way, these contexts can be combined to project the complete picture, if such a picture is even needed.

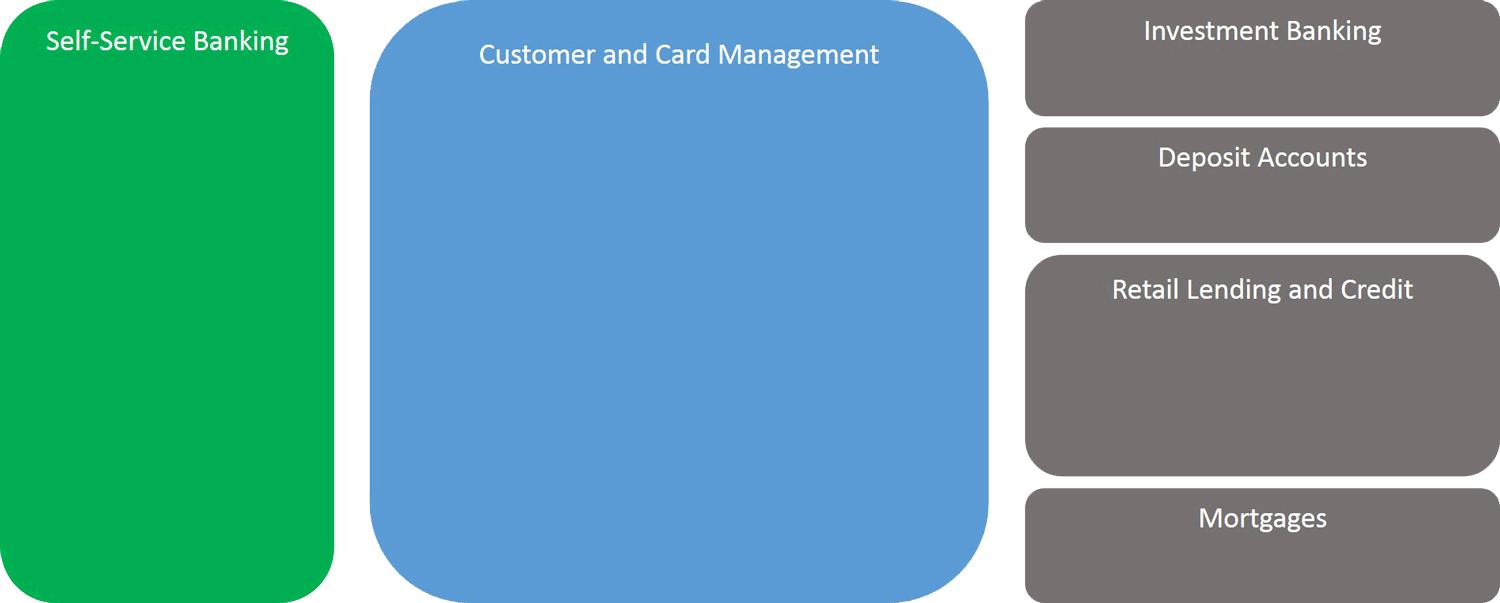
**Microservice Context Mapping: An Example**Let’s look at an example of this approach in action. Consider a large retail bank that is looking to introduce customer-centric services in order to meet its strategic goals for customer retention. It would like to offer a new customer-centric payment solution that will allow its customers to make purchases based on their total relationship with the bank—their accounts, investments, assets, interaction patterns—rather than basing the authorization decision on the balance of one specific account.

Retail banking is an inherently complex business domain. There are interrelationships between entities like customers, products and servicing channels (branches, online banking, point of sale), as well as mandatory regulations related to financial transparency and data privacy. Therefore, although the idea behind this new payments service is intuitive, it presents a complex problem space. Where do you start?

Following the DDD approach loosely, we can first start by breaking the retail banking domain down into subdomains. As a starting point—and following the inevitability of Conway’s Law—we can start with the organizational units that make up retail banking and are relevant to the customer-centric payments solution. These are:

|  |  |
| --- | --- |
| **Organizational Unit** | **Description** |
| Self-Service Banking | Applications that support self-service banking channels (such as mobile, Web, ATM and point of sale) will be needed to both set up and execute customer-centric payments |
| Customer & Card Management | Applications for customer information, customer authentication, and card-based authorizations are most closely aligned with the new payments service |
| Deposit Accounts | Systems of record and origination for deposit accounts (checking, savings) house information needed to authorize and fulfill customer-centric payments |
| Retail Lending and Credit | Systems of record and origination for lending products (e.g. personal line of credit) and credit cards also house information needed to authorize and fulfill customer-centric payments |
| Mortgages | Systems of record and origination for mortgages include information that can help the customer-centric payments solution make authorization decisions |
| Investment Banking | Although not part of retail banking, customers’ investments with the bank will be considered in the future for the customer-centric payments solution |

These subdomains can be depicted visually as follows:

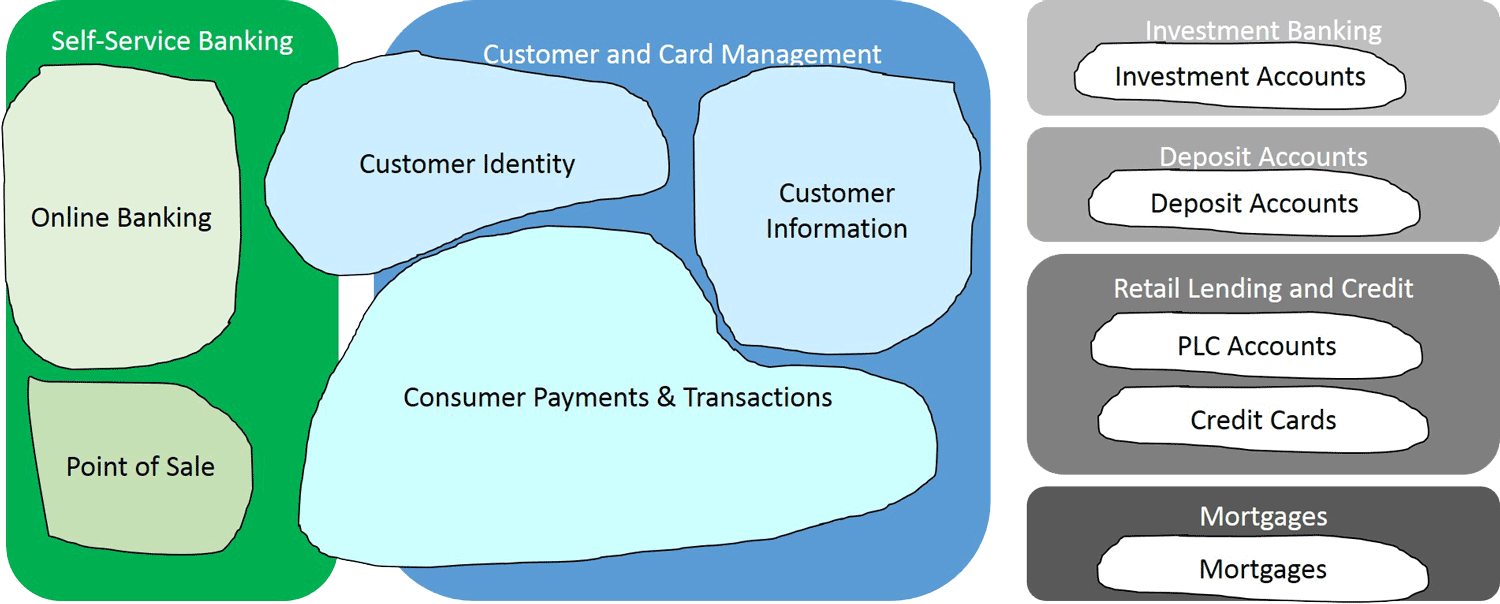
[](http://www.apiacademy.co/wp-content/uploads/2017/05/designing-a-system-of-microservices-1-750-v3.png)

This gives us a starting point to think about how to classify the services needed to create the customer-centric payments solution. The Self-Service Banking and Customer & Card Management subdomains are highlighted, since they have already been identified as the key subdomains for the solution. Using the DDD notion of “linguistic boundaries”, there are clearly some bounded contexts that emerge from the subdomains.

Within Self-Service Banking there are two distinct bounded contexts that come into play. The Web banking channel shares a common language with the mobile banking channel, since they both offer a high-touch customer experience and the applications supporting the mobile channel were built by and share components with the Web channel. We will call this context Online Banking. Point of sale, on the other hand, has a distinct language which is centered more around device management and merchant relationships. We will call this context Point of Sale.

Customer & Card Management is even more divided. Core customer information—including the authoritative list of the bank’s customers, their contact information and their product holdings—forms one distinct vocabulary, which we will call the Customer Information bounded context. Another distinct language is used to describe card functions, which are primarily tied to payments activity. This is the Consumer Payments & Transactions context. Lastly, there is a Customer Identity context focused exclusively on customer security and access control.

The product subdomains are not as critical to this payments solution but it is notable that the Retail Lending and Credit subdomain breaks down into separate PLC Accounts and Credit Card contexts. The complete set of relevant bounded contexts looks like this:

[](http://www.apiacademy.co/wp-content/uploads/2017/05/designing-a-system-of-microservices-2-750-v3.png)

Having established the bounded contexts, we can now think about what services and service consumers will make sense within them. As mentioned earlier, Online Banking has two service consumers: the Online Banking Web App and the Mobile Banking App. The new payment solution will also be consumed by message requests received through POS networks from the Point of Sale context.

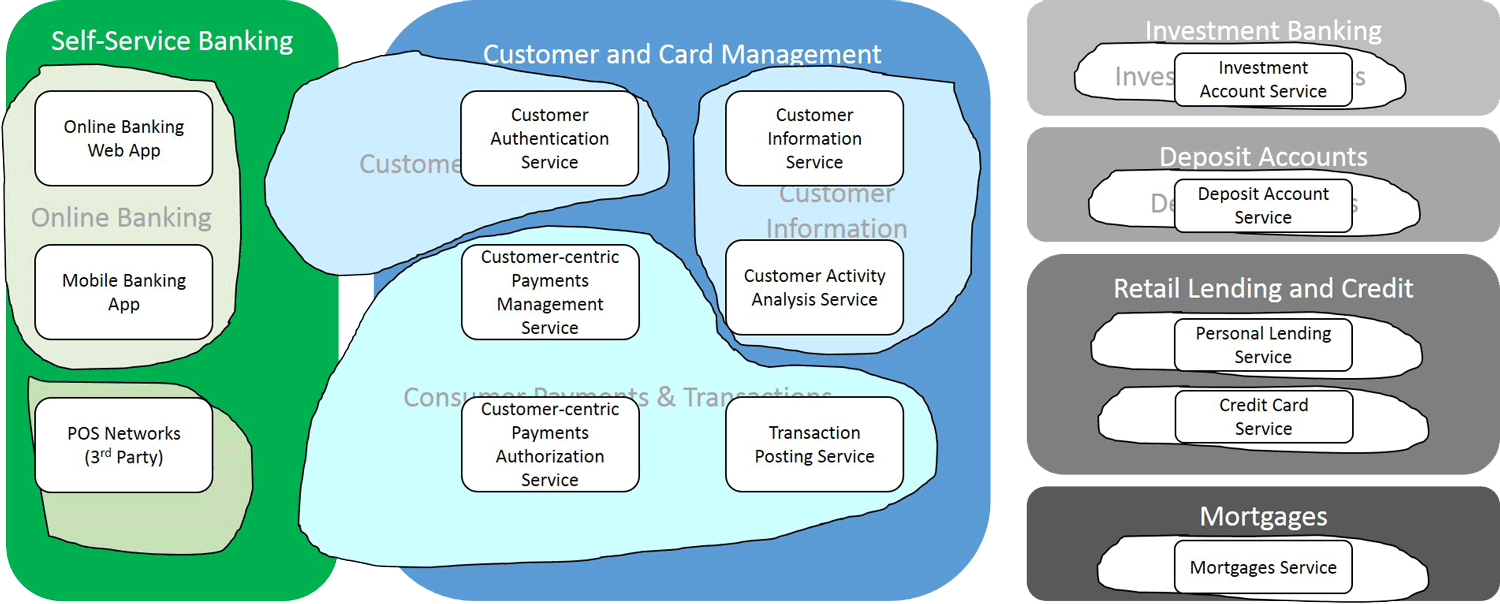
The Customer Identity context must provide a Customer Authentication service in order to ensure only appropriate requesters are able to access the new payments service. The Customer Information context contains the cross reference between customer and product holdings, so it follows that a core Customer Information Service is needed to supply this data for the new payments solution.

In addition to this, tracking customer fiscal activity—the financial events related to their product holdings—will be useful in making authorization decisions. Therefore, a Customer Activity Analysis service can be created within the Customer Information context.

The Consumer Payments & Transactions context is where the core services will be built for the new customer-centric payments solution. To start, customers need a way of signing up for the new offering and setting their account and product preferences. For this purpose, a Customer-Centric Payments Management Service is introduced.

For non-functional reasons—such as performance, security and availability—we will create a separate service for authorizing payments, called the Customer-Centric Payments Authorization Service. Lastly, since the posting of transactions to customer accounts can be posted later than the authorization decision, we will create a Transaction Posting Service that is decoupled from the authorization service.

For the product subdomains, we will only refer to a single eponymous service per bounded context. For example, the Deposit Accounts context simply contains a single Deposit Account Service. The overall system of services now looks like this:

[](http://www.apiacademy.co/wp-content/uploads/2017/05/designing-a-system-of-microservices-3-750-v3.png)

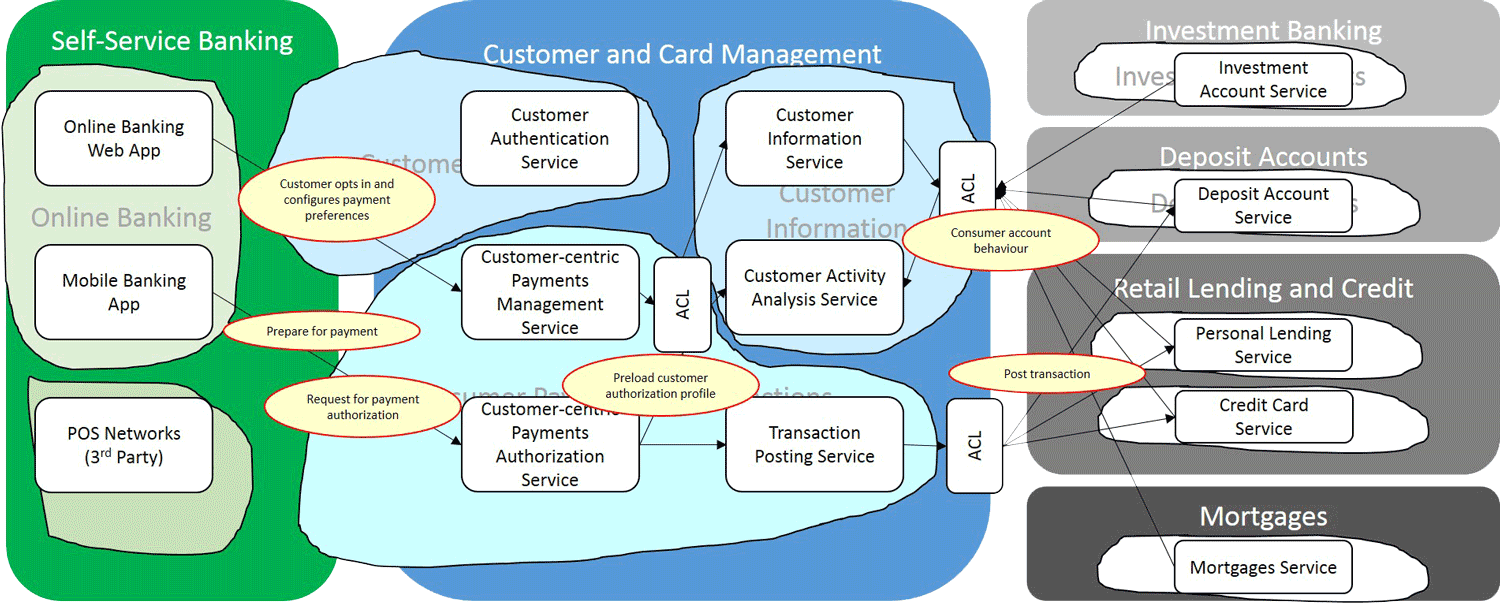
The last step in this design process is to annotate the interactions that will take place between services. In fact, one might want to model the interactions between bounded contexts in order to tease out the individual services but—for the sake of overlaying the visuals—we will depict this step last. Illustrating all possible interactions at this stage would be visually confusing, so we will only focus on some key ones that help us understand how the system will function. Here are some examples:

* A customer uses the Online Banking Web App to sign up and set their preferences for the new payments solution through the Customer-Centric Payments Management Service
* The Customer-Centric Payments Management Service retrieves the customer’s product portfolio from the Customer Information Service
* The Customer Activity Analysis Service builds a customer financial profile using information from the product systems
* The customer executes a customer-centric payment through the Mobile Banking App, which calls the Customer-Centric Payments Authorization Service
* The Customer-Centric Payments Authorization Service in turn builds an authorization profile using information from the Customer-Centric Payments Management Service and the Customer Activity Analysis Service
* Once authorized, the payment is forwarded to the Transaction Posting Service for completion, which then calls the appropriate product service

There are a few important points to note about this system of microservices. First of all, do not assume that each of these interactions will take place in real time when servicing the end user activity. For example, although the Customer-Centric Payments Authorization Service needs information from a few other services, an event-based approach combined with caching could be used to ensure its processing is self-contained in order to service the customer’s authorization request in real time.

Secondly, an anti-corruption layer (ACL) can be established between bounded contexts in order to provide loose semantic coupling. Finally, you’ll note that—although this is a fairly complex picture—there are no implementation details. In other words, this model could be implemented using many different technologies including languages, hosting environments and network protocols.

The final context map—including the interactions—is as follows:

[](http://www.apiacademy.co/wp-content/uploads/2017/05/designing-a-system-of-microservices-4-750-v3.png)

Ultimately, the intent of this system design process is to help take a best guess on how to define service boundaries for microservices within a complex solution. Hopefully, the approach described here will both establish a good initial model for the services as well as making it easier to redraw the service boundaries down the road.

The next step in the overall microservice design process will be to dive deeper on the individual services. And that will be explored in part three of this series. Stay tuned!