Understanding Service Variability for Profitable Software as a Service: Service Providers' Perspective

Eng Lieh Ouh[1] and Stan Jarzabek[2]

[1] Institute of Systems Science,
National University of Singapore
25, Heng Mui Keng Terrace, Singapore 119615
`englieh@nus.edu.sg`

[2] Department of Computer Science, School of Computing,
National University of Singapore
Computing 1, 13 Computing Link, Singapore 117417
`dcssj@nus.edu.sg`

Abstract. The number of tenants that subscribe to and pay for a service, and the cost of the SaaS computing infrastructure are the main factors that drive service profitability. In many application domains tenants' requirements for service vary. Service variability is the degree of the actual variability provided by the Service Provider over the variability required by the tenants for the service. With growing number of tenants, the likelihood of facing more diverse tenants' requirements increases. We conducted a study of how choices regarding service architecture affect service variability and the cost of supporting a service. We identified positive and negative impacts of service architectural choices on service variability and on Service Provider's costs. We illustrated how the knowledge of those impacts can help Service Providers analyze service profitability based on different service architecture models, leading to more-informed decisions regarding adoption of SaaS.

Keywords: Software as a Service, SaaS, service variability, service architecture, Profitability

1 Introduction

Service variability is the degree to which Service Provider can accommodate tenant-specific requirements into a service. The Service Provider tries to accommodate these requirement variations into the service so as to better fit the service to the tenants. As the unfit costs for the tenant decreases, the relative economic advantage of the SaaS business model increases [1]. If tenants' requirements for service vary only moderately, it is possible to engineer required variability into a service on cost-optimal (from Service Provider perspective) SaaS architectural model whereby all the tenants share the same service instance during

service execution. Dynamic binding techniques may be sufficient to address modest variations in service requirements. However, such cost-optimal SaaS solution may not be feasible if tenants' requirements differ in more drastic way. Shared service instance and dynamic binding techniques impose limits on how far we can vary service requirements. Then, a Service provider might consider SaaS architectural model with dedicated service instance for each tenant. Operational cost of such architecture is higher than that of shared instance, but dedicated instance architecture opens much more powerful options for engineering high-variability, adaptable services with static binding techniques.
To come up with a SaaS solution that maximizes profits, a Service Provider must weigh the revenue from selling a service to potentially many tenants, against the cost of SaaS computing infrastructure to support the service. Given inter-dependencies among factors that collectively determine profitability of service offering, the task is not easy.
We conducted a study of how choices regarding service architecture affect service variability and the cost of supporting a service. We identified positive and negative impacts of service architectural choices on service variability and on Service Provider's costs. We illustrated how the knowledge of those impacts can help Service Providers analyse service profitability based on different service architecture models, leading to more-informed decisions regarding adoption of SaaS. Our study is qualitative. In future work, we will extend it with quantitative analysis and models that more precisely correlate SaaS costs and benefits, giving more accurate insights into profitability of SaaS from service variability perspective.
The paper is organized as follows: We first describes the architectural choices of SaaS relevant to service variability in Section 2. In Section 3, we introduce the architectural models and further analysis the scenarios related to service variability in Section 4. Section 5 is on related work and Section 6 is our conclusion.

2 Techniques and Saas Architectural Choices Relevant to Service Variability

2.1 Service Engineering

1. *Static Binding Variability Techniques (SBVT)* - Static binding techniques instrument service code for adaptability to tenants' variant requirements at the design time. During (pre-)compilation or build time, variant requirements are bound to the variation points in service code to produce a custom service. Commonly used variation techniques include preprocessing (macros), Java conditional compilation, commenting out feature code, design patterns , templates and parametrisation, and build tools (e.g., make or Ant). XVCL [2] extends the concept of macros to provide better support for variability management in terms of generic design and separation of concerns.
2. *Dynamic Binding Variability Techniques (DBVT)* - Using dynamic binding techniques, we design a service that can adapt itself to the needs of

different tenants at runtime. Design patterns, reflection and parameter configuration files consulted during service hosting exemplify dynamic binding techniques. One common technique is using Aspect-Oriented Programming which involves specifying of aspects point cuts and advices that will describe the variability. Another common technique is Service Oriented Architecture Service Binding and Registry Lookup which involves registering of variants in the registry. Application components lookup the registry at runtime to dynamically bind variants to a service.

2.2 Service Packaging

As new tenants are on-boarded and requirements of existing tenants or service functions change, service must be adapted to accommodate evolving needs of tenants. Service adaptation has to be done without affecting existing tenants.

1. *Service Level Encapsulation (SLE)* - For Service Level Encapsulation, a service is implemented with identified shared service components. There is clear separation of components for each service, but not between tenants. The tenants who are using the service can be temporary affected during service modification but the tenants who are not using the service will not be affected.
2. *Tenant Level Encapsulation (TLE)* - For Tenant Level Encapsulation, a service is implemented with specific service components for each tenant. There is clear separation of components for each tenant. During service modification, only the specific tenants are affected.

2.3 Service Hosting

A service can be hosted on single or multiple application instances. Application instance is a software process (executable application code) running on an infrastructure platform.

1. *Shared Instance (SI)* - For Shared Instance, tenants access a service through a common application instance. The Service Provider considers this option typically to maximize resource utilization.
2. *Dedicated Instance (DI)* - For Dedicated Instance, each tenant accesses the service on its own dedicated application instance. The Service Provider considers this option due to high variation in tenants' requirements or for compliance to service level agreements.

A summary of the architectural choices is shown in a tree structure in Fig. 1.

2.4 Impact of the Architectural Choices

The architectural choices for service variability impact the degree, costs and benefits of the service variability. Table 1 summarize these relationships.

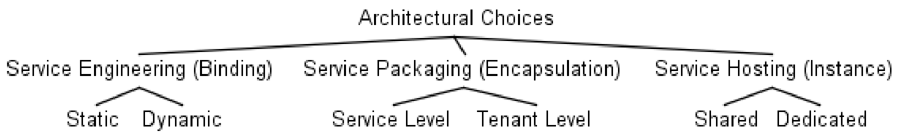
Architectural Choices
Service Engineering (Binding)
Service Packaging (Encapsulation)
Service Hosting (Instance)
Static
Dynamic
Service Level
Tenant Level
Shared
Dedicated

Fig. 1. Architectural choices

1. *Degree of Service Variability* - The degree of service variability is the extent and scope to which variations in service requirements can be handled. To understand the possible variations in service requirements, we use the entity-controller-boundary pattern to break down each service into its service components. Each service can be represented by interactions among a set of service components in terms of boundary, controller and entity components. Boundary components are used to interface externally with information elements and can varies with the elements and its representation termed as *Interface Variability.* Controller components manage the interactions among components. The composition of components including the flow of interactions and processing logic can varies among tenants result in *Composition Variability* and *Logic Variability.* Entity components represent the data of the service and can varies in the type of data elements and its structures. We termed this as *Data Variability.* The variations in service requirements for each service is the sum of all the variations of its service components for that service. A service is able to achieve high degree of service variability if it has the ability to handle high variations of service requirements.
2. *Costs and Benefit of (Profitability of investing in) Service Variability* - The cost incurred includes the cost for designing or re-designing the service binding in service engineering and service packaging. It also includes the infrastructure cost for service hosting to support service variability. High degree of service variability can be better supported by static binding (service engineering), dedicated instance (service hosting) and tenant level encapsulation (service packaging). However, these decisions requires higher costs in terms of design efforts and computing resources to run the service. The benefit of managing service variability is to increase the revenue by widening the tenants base. By being able to support higher degree of service variability, the tenant base can be increase easily. In a given situation, the Service Provider needs to make decisions to minimize the cost of service engineering/service packaging/service hosting and maximize the revenue (by widening the tenant base), ultimately affecting the profitability of a service.

3 SaaS Architectural Models

SaaS Architectural Models differ in how the service code is managed during service engineering, service execution and service hosting. For the purpose of this paper, we assume three architectural models. The Fully-Shared model is based on a shared application instance and service components being shared by tenants

Table 1. Impact of the Architectural Choices on Service Variability

	Degree of Service Variability	**Cost of Service Variability**	**Benefit of Service Variability**
Service Engineering			
Static Binding	High Degree	High Cost	Large Tenant base
Dynamic Binding	Low Degree	Low Cost	Small to Medium Tenant base
Service Packaging			
Service Level Encapsulation (SLE)	Low Degree	Low Cost	Small to Medium Tenant base
Tenant Level Encapsulation (TLE)	High Degree	High Cost	Large Tenant base
Service Hosting			
Shared Instance (SI)	Low Degree	Low Cost	Small to Medium Tenant base
Dedicated Instance (DI)	High Degree	High Cost	Large Tenant base

during service execution. The Partially-Shared model is based on shared application instance but as compared to Fully-Shared model the software components in Partially-Shared model can be tenant-specific (TLE) or service-specific (SLE) or both. For example, the boundary components can be tenant-specific while the controller components are service-specific. The No-Shared model is based on each tenant having own, dedicated application instance and the service components are tenant-specific. The Fully-Shared model can only adopt dynamic binding techniques while the Partially-Shared and No-Shared models can adopt both static and dynamic techniques. Table 2 summarize these relationships.

Table 2. SaaS Architectural Models

Models	**Service Hosting**	**Service Packaging**	**Service Engineering**
Fully-Shared	Shared Instance	Service Level Encapsulation	Dynamic Binding
Partially-Shared	Shared Instance	Service or Tenant Level Encapsulation	Static or Dynamic Binding
No-Shared	Dedicated Instance	Tenant Level Encapsulation	Static or Dynamic Binding

The approach to determine the architectural model depends on the variations of service requirements, architectural choices and the cost/benefit analysis of the Service Providers. Fig. 2 illustrates this approach.

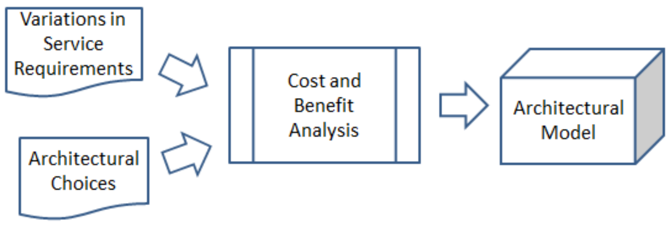

Fig. 2. Determination of the SaaS Architectural Model

4 Variability-Related Scenarios - Service Provider Perspective

Service Provider wants to employ SaaS solution that maximizes profitability of selling her application as a service. Therefore, the Service Provider needs an architectural model for a service that would lower the cost of the service offering and widen the tenants base.

1. *Lowest cost* - The Service Provider chooses the architectural model that incurs lowest cost to maximize profits. In particular for service variability, the Service Provider has to make decision to support service variability with the lowest cost. Based on Table 1 and 2, the Service Provider is likely to go for the Fully-Shared Model to minimize the cost. However, the lower degree of service variability imply that some tenants with high variations of service requirements cannot be met.
2. *Maximize Revenue* - The Service Provider needs to fulfill the tenant's expectations to widen the tenants base and increase revenue. The tenant expects their requirements to be fulfilled as if the service is single tenant. The higher degree service variability implies greater extent of the tenant's requirements that can be fulfilled. The Service Provider can go for No-Shared Model. The associated higher cost incurred by the Service Provider imply that the tenants have to be able to afford the higher fee.
3. *Tenant On-boarding (Low degree of service variability)* - The Service Provider wants to on board as many tenants as possible. However, the benefit of on boarding new tenants (increased revenue) should be weighed against the cost of adapting the service to possibly new requirements (i.e., the cost of service variability). In this example assuming there is an initially small number of tenants (e.g. 30 tenants) with low variation of requirements. In this case, the Service Provider chooses the Fully-Shared model to minimize the cost of service variability. If $\mathrm{InfraSharedCost}_{(30)}$ is the shared infrastructure cost of supporting 30 tenants and $\mathrm{CostDVTDesign}_{(30)}$ is the cost to implement the dynamic binding variability techniques, then the cost of offering an application as a service is:

$$\mathrm{InfraSharedCost}_{(30)} + \mathrm{CostDVTDesign}_{(30)}$$

4. *Tenant On-boarding (High degree of service variability)* - Assuming there are 50 more tenants (more diverse requirements) interested in the service with 30 existing tenants. The Service Provider can provide dedicated service instances for new tenants and applying both static and dynamic binding techniques to cater for variant requirements. In this case, the Service Provider needs to evaluate the overall cost of providing dedicated instances and implementing the static and dynamic binding techniques for a No-Shared model. If $\text{CostSDVTDesign}_{(50)}$ is the cost to re-design for static and dynamic binding and $\text{InfraDedicatedCost}_{(50)}$ is the dedicated infrastructure cost of supporting 50 tenants, then the cost of offering an application as a service is:

   $$\text{InfraSharedCost}_{(30)} + \text{CostDVTDesign}_{(30)} + \text{InfraDedicatedCost}_{(50)} + \text{CostSDVTDesign}_{(50)}$$

   The Service Provider can also chooses to place the 50 tenants on with existing tenants in a Partially-Shared model. In this case, the Service Provider needs to evaluate the impact due to the higher variability of requirements. If CostDVTRedesign(50) is the cost to re-design for dynamic binding, then the cost equation for service variability is:

   $$\text{InfraSharedCost}_{(30)} + \text{CostDVTDesign}_{(30)} + \text{InfraSharedCost}_{(50)} + \text{CostDVTRedesign}_{(50)}$$

   To on-board the 50 tenants, the Service Provider can make decisions based on the minimum cost of both choices.

   $$\text{Min } (\text{InfraSharedCost}_{(50)} + \text{CostDVTRedesign}_{(50)} \text{ , InfraDedicatedCost}_{(50)} + \text{CostSDVTDesign}_{(50)} \text{)}$$

   If the Service Provider is aware of the need to support up to 80 tenants(30 tenants with low variation of requirements and another 50 tenants having high degree variation of requirements), the service provider can alternatively plan to support the 80 tenants directly with No-Shared for all 80 tenants. If InfraDedicatedCost(80) is the cost for dedicated infrastructure to support 80 tenants and CostSDVTDesign(80) is the cost to implement the static and dynamic binding variability techniques, then the cost equation for service variability is:

   $$\text{InfraDedicatedCost}_{(80)} + \text{CostSDVTDesign}_{(80)}$$

5. *Service Isolation* - To many organizations, security and privacy are still the top issues in adopting SaaS. The No-Shared model would be most suitable with software components and process instance being tenant-specific. Service Providers might want to propose Partially-Shared model (e.g. only the entity components are tenant-specific) for the group of tenants who are more price-sensitive.

5 Related Work

The profitability model for SaaS is an area that attracts much interest. The author of [1] propose an analytical SaaS cost model based on user's fit and exit costs. In [3], the author analysis the pricing strategies for SaaS and COTS. The authors of [4] attempts to maximize Service Provider's profit and tenant functional commonality for tenant onboarding in terms of contracts. In comparison, we evaluate profitability from both the costs and revenue perspectives with the architectural choices.

6 Conclusion

We addressed the problem of profitability of SaaS solutions in view of the revenues from selling the service, and the cost of SaaS computing infrastructure to offer a service to tenants. The first depends on the number of tenant who pay for the service. The latter is determined by the cost of computer resources utilization, and the cost of service engineering. We identified trade offs involved in Service Provider decisions regarding the choice of service variability (i.e., the ability to satisfy the diversity of tenants' requirements) and SaaS architecture for the service. With the growing number of tenants, the likelihood of facing more diverse tenants' requirements increases. We found that high service variability may call for more costly SaaS architectures (e.g., dedicated service instance as opposed to shared instance), and more costly techniques for service variability management (e.g., static binding as opposed to dynamic binding). We summarized the results of our analysis in tables that show influences among factors that determine profitability of the SaaS solution. We believe our results can help Service Providers make more informed decisions regarding service offering.
Our current study is qualitative. In future work, we will extend it with quantitative analysis and models that more precisely correlate SaaS costs and benefits, giving more accurate insights into profitability of SaaS from service variability perspective. We plan to decompose cost and benefit of SaaS solution into more detailed factors that will include the effort of migrating an existing application into a service and on-board new tenants.

References

1. Ma, Dan. "The business model of" software-as-a-service"." Services Computing, 2007. SCC 2007. IEEE International Conference on. IEEE, 2007.
2. Jarzabek, Stan, et al. "XVCL: XML-based variant configuration language." Software Engineering, 2003. Proceedings. 25th International Conference on. IEEE, 2003.
3. Ma, Dan, and Abraham Seidmann. "The pricing strategy analysis for the Software-as-a-service business model." Grid Economics and Business Models. Springer Berlin Heidelberg, 2008. 103-112.
4. Lei Ju, Bikram Sengupta, and Abhik Roychoudhury. "Tenant Onboarding in Evolving Multi-tenant SaaS." (2011).