“Software + Services: The Convergence of SaaS, SOA and Web 2.0” Beat Schwegler

May 22, 2009
[MS Techdays 2009] Summary “Software + Services: The Convergence of SaaS, SOA and Web 2.0” Beat Schwegler

I added some personal interpretations!

Trends in IT
Consuming software as a service (SaaS) can be easy if not a lot of customization is involved, for example e-mail and CRM.

The current web has evolved from a Web 1.0 application model, i.e. 1 author and n consumers, to the Web 2.0 model that works with n authors and m consumer and by that explicitly leveraging the network effects. For example Facebook and Amazon.

What are thriving Web 2.0 applications made of?
Firstly they bring together lots of data. Why is Wikipedia doing better than Encarta? The world is feeding Wikipedia whereas Encarta is being feed only by one company’s workforce. In a slogan “Data will be the next Intel inside’

Secondly they work on several devices. Applications must be able to adapt easily to the device there run on i.e. the environment of the user. Currently users are very demanding and they switch from a desktop to a laptop to a smart phone in a wink of eye. If your application does not support all these environments you are commercially handicapped.

What new business models can be found to make Web 2.0 application a success? In other words how to make money from blogs, wiki’s, social network, enterprise search, mash-ups, Š

In their book *Place to Space: Migrating to E-business Models* Peter Weill and Michael Vitale introduce several models. Every business model is explained using three basic flows, 1. cash-flow, 2. flow of information and 3. flow of products, and relationships between parties participating in the flows 1. consumer, 2. producer or 3. intermediate party.

A typical example is Google advertising. Google provides, as intermediate party, information to a potential customer looking for a product and receives a small fee for this from a supplier of that product. You have an information flow between Google and the customer and a cash flow between the supplier of the good and Google. If the customer decides to buy there is product flow and a cash flow between supplier and consumer.

Example of Web 2.0 applications:

- Eve online

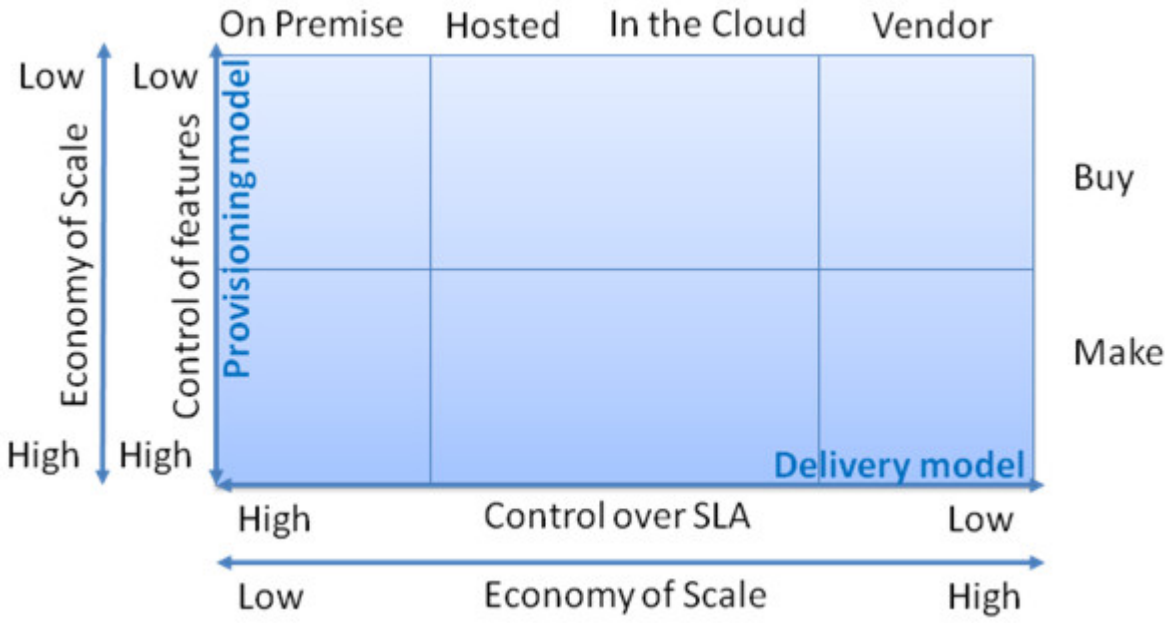
- Photosynth
- British Library
- Yosemite Extreme Panoramic Imaging Project
- Exchange server 2007

Enterprise 2.0

Enterprise 2.0 is about bringing together:

- SaaS: Service delivery model = 'How to deliver!'
- SOA: Service composition = 'How to create!'
- Web 2.0: Service Experience and business model = 'How to make money!'

Following decision matrix for service delivery and provisioning can be used in Enterprise 2.0 application models.



The provisioning axis is the decision between making it ourselves or buying a solution, balancing the control over implemented features and the economics of scale. The other axis is the delivery model going from local hosted on your own premises to completely hosted and maintained by a vendor, balancing the control over the SLA and economics of scale.

Important for Web 2.0 applications is how your company will cope if an applications becomes a success?

Technically this means that applications must be build in such way they scale-up and scale-out very easy.

Economically the question is raised how and where the applications are going to be hosted. How are you going to find the money required for the additional server capacity? Do you really want to invest in the server capacity to host it yourself? What if things go bad? In that case you?l end up with expensive unused capacity!

The economic crisis forces us to critically look at Capital Expenditure (CAPEX) and Operational Expenditure

(OPEX). Enterprise 2.0 concepts give you alternatives to investing in self hosting during the credit crunch. If you start using services in the cloud with flexible capacity schemes you?l avoid pre-financing. Operational costs are reduced as well. Think about installing a patch for an OS if you?e self hosting compared to hosting in the cloud. For the latter case it is just up to your service provider to get this organized.

The idea about moving to the cloud is basically reducing costs by folding back to core activities. For most of us hosting is not a core activity and so it can be focal point for cost reduction.

Programming Model for the Cloud

Normally applications adhere to CAP:

- Consistency
- Availability
- Tolerance to network Partitions

Applications in the cloud trade in one of these. For example ACID DB transactions are very hard, if not impossible, when you want scalable systems with thousands of simultaneous users. So CAP becomes BASE:

- Basically availably networks
- Soft-state
- Eventual consistency

Microsoft's Programming Model adhere to the Cloud

Microsoft's Windows Azure is a platform specifically designed for cloud services. Microsoft promises flexible scaling where you only pay for the hosting capacity you require. The platform brings everything, you would locally use, to the cloud. For example SQL DB has become SQL Services for Azure.

.NET Services are an extension of .NET capabilities to the cloud, trying to make cloud computing as transparent as possible to the developer in other words leveraging the developers knowledge to the cloud.

For the moment three services are available but more to follow:

- .NET Service Bus
- .NET Workflow Service
- .NET Access Control Service

Summary

- Services enrich software ‘1+1 = 3’ think about Web 2.0 new business models
- Effective provisioning is key: reuse what others already have build as a service, do not reinvent the wheel
- Think green and save a buck: cloud computing allows for power efficient software hosting in data centers
- Focus on programming models embracing cloud computing and take into account the BASE principles when developing for the cloud.
