Mashups, SaaS, and Cloud Computing: Evolutions and Revolutions in the Integration Landscape

Boualem Benatallah (University of New South Wales, Australia /University of Blaise Pascal, France)

Based on tutorial at ICDE’09 with:

Fabio Casati (University of Trento, Italy),
Florian Daniel (University of Trento, Italy),
Jin Yu (University of New South Wales, Australia)
Agenda

• Issues and Solutions in Data and Application Integration
• SOA and Service Composition
• Mashups
• Integration, Mashups, and Cloud Computing
Integration/composition is key to operations improvement and monitoring

Silos of data sources and applications (before integration, no global view)

*Integrated systems*: global view (important for cost reduction, global visibility, and increased productivity)
Example 1: Enterprise Information Integration (EII)
Example 2: Scientific processes

Data is acquired from mass spectrometer. The format is native to the instrument used.

The native binary data is converted to a standard XML-based intermediate format.

The data is converted to peak list format. Instrument-specific algorithms are used.

Remove low-quality spectra from peak list or determine the charge state of the precursor ion.

Process

Analysis of peak list by a database search engine such as Mascot.

Database search using Mascot

Analyze the peptide or protein identifications. Statistically analyze for proteomics results.

Statistical analysis using ProvaLe

Database search results

Final results

Biological information

Biological sample analysis by mass spectrometry

Convert binary data to mzXML format using RedAW

mzXML data

Convert mzXML to PeakList using mzXML to Other

PeakLists

PeakList filter

Processed PeakLists

Database search using Mascot

Database search results

Final results
Example 3: B2B Integration

Private process (Company-specific)

Public process (Standard)

Public process (Standard)

Private process (Company-specific)

(Source: e-business Architectures and Standards, Anil L. Nori, Tutorial, VLDB’2002, HongKong, China)
Example 4: Mashup (more on mashup later)
Development of Composite Applications
(In practice)

- Applications and data sources are autonomously developed and deployed.
- Proprietary technologies (communication protocols, data formats, business and presentation logic).
- Costly development and maintenance of integrated systems especially in large and dynamic environments.
Interoperability Layers

Business Partner 1
- Internal system
- External Interactions
- Policy
- User Interface
- Business Protocol
- Content of document
- Message exchange
- Programs

Business Logic Adapter
- Event Synchronization
- Data/Document Transformation
- Java Invocations

Middleware Infrastructure
- CORBA
- (D)COM

Business Partner 2
- External Interactions
- Internal System
- Policy
- User Interface
- Business Protocol
- Content of document
- Message exchange
- Programs

Business Logic Adapter
- Event Synchronization
- Data/Document Transformation
- C/C++ Invocations
Communication Layer

- Exchange of messages among partners
  - Transport binding, communication modes such as asynchronous/synchronous
  - Partners must understand messages (agree on the formats)
  - Message exchanges must be done in a secure way
  - Message exchanges must be done in a reliable manner
- Partners use different protocols (or even proprietary protocols)
  - Internet messaging (e.g., HTTP, SOAP), messaging middleware (e.g., IBM’s MQSeries), EDI VANs, remote application services (Java RMI, CORBA IIOP), ...
- Interoperability objective
  - independence from transport protocols
- Interoperability solutions
  - Translate messages between heterogeneous protocols
- Examples of solutions
  - Message broker/server, message transformer
Enterprise Application Integration

- Typically rely on distributed object frameworks such as CORBA, DCOM, EJB and other state of the art technologies such as database gateways and transaction monitors
- Separation between applications and infrastructure services (e.g., persistence management, security management, transaction management, trading, event, naming services)
- EAI suites provide pre-built data and application integration facilities (e.g., application adapters, data transformations, and messaging services)
EAI (Enterprise Application Integration)

- Typically rely on distributed object frameworks such as CORBA, DCOM, EJB and other state of the art technologies such as database gateways and transaction monitors.
- Developers focus on component specification and logic (e.g., using CORBA IDL, programs), they do not need to know where remote objects are located, in which languages they are implemented, how they communicate, etc.
- Emphasis more on platforms integration: wrapping heterogeneous systems, routing requests, remote operation invocation.
- Common API layer: business objects are wrapped with explicit interfaces, they communicate by making remote calls directly to their peers.
- Data, process, presentation level heterogeneities are worked out offline/mostly manual (some tool support exist).
Content Layer: Message structure and semantics

• Partners must understand the structure and semantics of messages

• E.g., does a document represent a purchase order? A request for quote? A production description?

• Structures (e.g., different structures for a purchase order), services may provide the same functionality but with different operation structures (e.g., different names, different signatures)

• Semantics: Does a service provide a required functionality? Does Price mean Price including tax?
Electronic Data Interchange

Buyer application

EDI System

Backend System

EDI Doc

Network connection

EDI Doc

EDI System

Backend System

VAN

Seller application
Data integration solutions

**Integrated access to:**
Multiple data sources/data flow

- Data integration approaches: EII (virtual data views), ETL/data flows (e.g., scientific processes/process data warehouse)
- Presentation logic is ad-hoc, and in hybrid applications, the application logic is ad-doc
Data Integration (state of the art)

- Wrappers (uniform access to heterogeneous sources)
- Schema matching (e.g., linguistic / structural / ontology analysis to identify elements similarity)
- Data Transformation languages (e.g., XSLT, XQuery)
- Models Management (recent work in the DB community)
- Data flow languages (ETL, scientific workflows)
- Good progress, but more work is needed on usability and consolidation
Business process Layer

• Semantics of interactions (joint business process)
• Partners must agree on the choreography of interactions and meaning of messages
• E.g., steps (send order, process order, deliver product), deals (a purchase is refundable after 2 days)
• Semantics of interactions must be well defined, such that there is no ambiguity as to:
  • What a message may mean? What actions are allowed? What responses are expected?
• For example, if a company A requires an acknowledgement of purchase orders from its partners, then partner processes must have a corresponding activity
Process/application integration

Composition/coordination

- **Integration approaches**: EAI/Workflow, SOA/BPEL
- Presentation logic is ad-hoc
Business Process Layer (Cont.)
Business Process Layer (cont.)

- Interoperability at this layer requires the understanding of the behavior of partner public processes (called external conversations, business protocols)
- Traditional EAI middleware
  - component interface describes very little semantics (e.g., message formats)
  - business process is usually agreed upon off-line.
- Automation requires rich interface description models but a balance between expression power and simplicity is important for the success of the technology (expressive: useful and usable)
Effective interface description should cater for:

- Making *implicit information* (as in closed environments) *explicit* (essential in autonomous environments)
- *Messages order* (e.g., buy after login)
- Transactional implications (e.g., can I cancel a purchase?, if yes at what cost)
- *Temporal aspects* (e.g., can I cancel a purchase any time? After a fixed time period?)
- *Security* (will the results be digitally signed?)
- *Privacy* (How do you know if partners have compatible policies?)
- *Quality of service* (e.g., performance/reliability)
- *Exception Handling* (e.g., support for transaction protocols)
Workflow Management Systems

- Information
- Flow
- Resources
- Organization

Automate business logic, information flow
Control flow

- Receive order
- Check Local Stock
  - inStock=false
    - Cancel Order
    - shippingAvail=false
  - inStock=true
    - Confirm Order
    - shippingAvail=true
    - Check with supplier
- Adapter
- Adapter
- Order goods
Data Transfer among Components

- Blackboard vs data flow
Services and Service composition
Web service

- A service available on the Web and designed to be accessible by another application
- A web service is NOT the same thing as a service on the Web
Historic standards

- WSDL (or else) interfaces

- Customer
  - SOAP-based middleware
  - SOAP (or else) messages (over http, or smth else)

- Supplier
  - SOAP-based middleware
Services as components
WS-I SOA stack

- Additional Capabilities
  - Management
  - Portals
- Business Process Orchestration
  - Composition/Orchestration
- Composable Service Elements
  - WS-Security
  - Reliable Messaging
  - Transactionality
- Messaging
  - Endpoint Identification, Publish/Subscribe
- Description
  - XML Schema, WSDL, UDDI, SOAP with Attachments
- Invocation
  - XML, SOAP
- Transports
  - HTTP, HTTPS, Others
Service composition

1. Receive order
2. Check Local Stock
   - inStock=false
     - Check with supplier
       - shippingAvail=false
         - Cancel Order
       - shippingAvail=true
         - Confirm Order
   - inStock=true
     - Order goods
Workflow system architecture

Workflow model, + possibly org model (or go to enterprise directories)

Workflow model repository

development tools

workflow engine

analytics engine

execution logs

SAP adapter

SAP

Email adapter

Email app

Custom adapter

Account mgmt
Elements of WS composition middleware

- Development tools
- Process model repository
- Composition engine
- Analytics engine
- Process execution logs

Service composition language (up to now, no org modeling)

Company A
- Web service
- Web service

Company B
- Web service
- Web service

Company C
- Web service
- Web service

Company D
- Web service
WS-BPEL 2.0

Basic Activities
- receive
- exit
- reply
- throw
- rethrow
- assign
- wait
- validate
- compensate
- empty
- compensateScope
- extensionActivity

Structured Activities
- flow
- pick
- sequence
- forEach
- while
- repeatUntil
- scope

Variables
- WSDL Message
- XML Schema Type
- XML Schema Element

Partner Links
- partner link
- Port Type 1
- Partner Link Type
- Port Type 2

Handlers
- event handler
- fault handler
- compensation handler
- termination handler

Properties Correlation Sets
- Property 1
- Property 2
BPEL and its richness

- Complex synchronization constructs
- Events
- Exceptions
- Compensation
No KISS in Web Services

• WSDL and SOAP not that easy as well, not to mention the other specs....
• Even if Web services were meant to be simple, born to be simple..
What are we talking about?

- **Mashup** – possible definitions
  - “...a mashup is a web application that combines **data** from more than one source into a single integrated tool...” [wikipedia.com – March 24, 2009]
  - “...you can integrate two or more [...] **Web APIs** to create something new and unique, known as a mashup...” [*]
- A mashup is a web application that is developed by composing **data, application logic**, and/or **user interfaces** originating from disparate web sources.
- Similar terms: service mashups, data mashups

Mashup = integration the Web 2.0 way

• Young integration practice using the Web as platform
• Highly user-driven
  • Oftentimes the actual providers of content/functionality are not even aware of being “wrapped”
  • Google Maps example: initially skilled users hacked the AJAX code of the application
• Strong evolution: from hacking to first systematic development approaches in a few years
Let’s see an example

- The HousingMaps application ([http://www.housingmaps.com](http://www.housingmaps.com)) composed of:
  - Google Maps ([http://maps.google.com](http://maps.google.com))
  - Craigslist ([http://www.craigslist.com](http://www.craigslist.com))
A mashup example

- HousingMaps (http://www.housingmaps.com)
- http://maps.google.com
- http://www.craigslist.com

Google Maps

Own application logic/UI

Craigslist
Web 2.0

- **Web 2.0?** Again, there are lots of different (and sometimes diverging) definitions:
  - “Web 2.0 is a term describing the trend in use of World Wide Web technology and web design that aims to enhance creativity, information sharing, and, most notably, collaboration among users…” [wikipedia.com]
  - “Web 2.0 is best described as a core set of patterns that are observable in applications that share the Web 2.0 label. These patterns are services, simplicity, and community...” [*]

The enabling factor of Web 2.0

- Over the last years we have been witnessing **two main trends** on the Web:
  - User participation in the **content creation** process (e.g., communities, social networks, blogs...)
  - User participation in the **development** process (e.g., mashups)

- Which are enabled or fostered by:
  - **Simplicity of usage**: intuitive, interactive applications
  - **Simplicity of development**: novel and standardized web technologies
Some figures (programmableweb.com)

- Most popular categories of **mashups**

- Most popular web **APIs**
Dynamics of the ecosystem

- Constant growth since programmableweb.com went online (over 600 days) [by Michael Weiss, Carleton University]
Developing a mashup: what does it mean?

- The mashup development scenario
Distribution of apps over C and S

Source: www.coachwei.com
Mashup component/API types

UI logic
- Visualization widgets
- Complex widgets
- Client apps

Data

Client

App

Conventional Webapp

C/S services

Client services

Server-Side services

Feeds

C/S apps

No UI
The technological landscape

Client

UI logic
(D)HTML

App
AJAX
Flash, Silverlight

Data
JSON, XML
SOAP, HTTP

Server

UI logic
HTML, templates, ...

App
PHP, Ruby, Java, C++, ...

Data
XML, RSS, Atom
Relational DBs, OODBs, ...
SOAP/WSDL web services

- Programming interfaces accessible over the Web
  - **WSDL** = Web Service Description Language
    - Abstract service description language (tech-agnostic)
  - **SOAP** = Simple Object Access Protocol
    - XML message exchange protocol
- **SOA** = Service-Oriented Architecture
  - Producer, consumer, registry (virtual marketplaces)
  - Complex advanced features: security, reliability, transactions, addressing,...
- Orchestration and choreography
RESTful web services

• A new **architectural style** of developing web services
• Principles
  • Operations based on **HTTP methods** (Get, Post, Put, Delete)
  • Services are **stateless** (no session data at the server side)
  • Access via hierarchically structured **URIs**
  • XML or JSON over **HTTP**
• Benefits
  • Simplicity and immediacy
  • No big overhead for composing and parsing messages
  • More efficient service implementations
“Protocol” usage by APIs

Protocol Usage by APIs

- REST (65%)
- SOAP (22%)
- JavaScript (6%)
- XML-RPC (3%)
- Atom (2%)

ProgrammableWeb.com 03/19/09
Mashup development manually (1/2)

- **Scenario 1** (at the beginning): No APIs available
- Development tasks
  - **Read** and interpret AJAX code of GMaps
  - **Hack** into GMaps code to implement marker support
  - **Extract data** from Craigslist with regular expressions (write a wrapper)
  - **Format** extracted data and forward data to GMaps
- **Problems**
  - No stable interfaces
  - Highly error-prone and time-consuming
Mashup development manually (2/2)

• **Scenario 2** (today): GMaps comes with AJAX API and Craigslist provides an RSS feed

• Development tasks
  - Instantiate GMaps component
  - **Layout** RSS feed
  - Set **markers** through GMaps API

• Problems
  - Manual development for skilled programmers
  - Manual parsing of RSS feed
  - No common Web API format
Partially assisted development

- There are many (online) tools for
  - Data extraction from Web pages
  - Web content clipping
  >> Aid the development of mashup components or APIs

[Images of Dapper, OpenKapow, LiXto, and RoadRunner]
Fully assisted development

• Mashup tools/platforms
  • Simplify the overall development process
  • Provide easy-to-use development instruments
  • Provide dedicated execution environments
  • Support the whole lifecycle of mashup applications
  • Enable even the less experienced user to mash up own applications

• Let’s see some representative examples
  • Yahoo Pipes, Intel Mash Maker, Microsoft Popfly, JackBe Presto (yet, there are many others)
• Powerful, hosted **data mashup** tool for the processing of
  • RSS/Atom feeds
  • XML/JSON data resources/services
• Targets skilled users and **programmers**
• **Data flow** approach (pipes)
• No support for **user interface** design
• Client-side **browser extension** for interactive mashup development
  • Data extracted from annotated web pages
  • Widgets (UI components) for data visualization
  • Copy/paste of Web contents into other Web pages
• Targets **average Web users** and programmers
• Data passing through **environment variables**
• No support for **service components**
• Highly interactive, hosted **mashup platform** for consumer mashups
  • Mashup “blocks” for data, application logic, and UIs
  • Mainly JavaScript blocks
  • Comes with own block builder
• Targets **advanced Web users** and programmers
• Data passing by **coupling** components and mapping outputs to inputs
• Still weak support for UI components
• Full-fledged **enterprise mashup platform** with desktop integration
  • Main focus on data mashups
  • Support for web services and (local) spreadsheet files
  • Separate layout support for UIs (mashlets and portals)
• Targets advanced users and programmers
• Data flow logic
• Still limited layout capabilities
Our own research on mashups

- **UI integration**
  - Stand-alone web apps as **UI components**
  - **Synchronization** among components

- **Universal integration**
  - UI, application logic, and data components
  - One **component model**: abstract components, highlight similarities
  - One **composition model**: one formalism for synchronization and orchestration
  - **Hosted** development and execution
UI integration: visual editor

- List of application components available for the mashup. Additional components may easily be loaded into the editor by referencing the respective online resource.
- Mahup logic modeling canvas.
- Graphical model of the composition logic.
- Tabs that allow the designer to switch between different views (e.g., composition logic vs. layout) on the composite application under development.
- The mashup application running in a standard web browser.
- Deployment
Hosted execution environment
Hosted execution environment

• Development challenges:
  • Seamless integration of **stateful** and **stateless** components and of **UI** and **service** components
  • **Short-living** and **long-running** process logics in the same environment
  • **Distribution** of execution tasks over client and server
  • Transparent handling of multiple **communication protocols**
Determines the **nature** of components and influences how components can be glued together.

Determines how components are **integrated** to form the mashup, assuming components are readily available.

Assists the developer in the mashup process and eases development.

Enables the **execution** of mashups and determines how mashups are delivered to their users.

---

**Analyzing mashup tools**

- **Component model**: Assists the developer in the mashup process and eases development.
- **Composition model**: Determines how components are integrated to form the mashup, assuming components are readily available.
- **Development environment**: Determines the nature of components and influences how components can be glued together.
- **Runtime environment**: Enables the execution of mashups and determines how mashups are delivered to their users.
Component model

- **Type**
  - Data (DA) vs. application logic (AL) vs. user interface (UI)

- **Location**
  - Local vs. remote

- **Direction of interaction**
  - One-way vs. two-way

- **State**
  - Stateful vs. stateless

- **Behavior**
  - Active vs. reactive
Composition model

- **Type**
  - Data (DA) vs. application logic (AL) vs. user interface (UI)

- **Orchestration style**
  - Flow-based vs. event-based vs. layout-based

- **Data passing style**
  - Data flow vs. blackboard
    (without vs. with shared memory)

- **State**
  - Stateful vs. stateless

- **Instance model**
  - Instance-based or continuous
Development environment

- **Target users**
  - Web users vs. tech-savvy users vs. programmers

- **Interface paradigm**
  - Visual drag-and-drop vs. textual editors vs. combinations

- **Type of support**
  - Composition only vs. composition + components vs. component only

- **System requirements**
  - Hosted, web-based vs. standalone
  - Additional modules, plug-ins, or browser features
Runtime environment

- **Deployment model**
  - Complied (web app based) vs. interpreted (engine-based)

- **Execution location**
  - Local vs. remote vs. hybrid

- **System requirements**
  - Browser plug-ins or extensions?

- **Scalability**
  - Number of data sources, in the number of models (compositions), or in the number of users
Applicability of mashups

• But what about the utility of mashup applications?
  • Mashups are still mostly 1-page apps...
• Only very few innovations are really breakthroughs, most innovations only create little value
• Perfectly understanding customer needs, in order to customize software and satisfy as much users as possible, is costly – if not impossible
• Mashups may leverage “user innovation”:
  • Users themselves know best what they want
  • Mashups enable them to build their own applications
The long tail of the SW market

Number of users vs Applications

New market vs Old market

Bigger and Longer
A new development paradigm?

- Characteristics of **modern** web applications
  - **Fast** development cycles (Internet time)
  - **Incremental** development (prototype-based)
  - Continuous online **evolution**
- The software life cycle of modern web applications is **no longer** captured by traditional life cycle models (e.g., the spiral or the waterfall model)
- And what about user-driven composition of web applications and **mashups**?
Crowd Programming in the Clouds
Focus of this last section

- Saas and cloud not the focus, would need a seminar on their own
  - VMs, cooling and energy mgmt, utility computing...
- Goal here is to say what they are and why they are relevant / how they are related to mashups and integration
Just like the early days of Web services

Aaron Weiss: “Cloud computing,” as it’s being called by everyone from IBM to Google to Amazon to Microsoft, is supposedly the next big thing. But like the clouds themselves, “cloud computing” can take on different shapes depending on the viewer, and often seems a little fuzzy at the edges.
Larry Ellison’s view on the cloud

Oracle Cloud Computing Center

Oracle has played a pioneering role in making Grid Computing relevant to enterprises with groundbreaking products such as Real Applications Cluster (RAC), Automatic Storage Management (ASM), and Storage Grid. More recently, Oracle has brought Grid Computing to middleware with the Application Grid approach to infrastructure. These products/technologies make the enterprise IT infrastructure elastic so that it can grow incrementally without any theoretical upper limit, as well as provide the flexibility to move resources around in order to meet dynamic business priorities.

Continuing its pioneering role in shaping enterprise computing, Oracle is pleased to introduce new offerings that allow enterprises to benefit from the developments taking place in the area of Cloud Computing. As a part of our initial offering, Oracle has partnered with Amazon Web Services (AWS) environment to offer the following products and services:

- Deploy Oracle Software in the Cloud
- Backup Oracle Database in the Cloud

These offerings may be extended to other Cloud platforms in the future.
BuzzTracker – larger scale
"Cloud-based" console takes aim at Wii, PS3, Xbox 360 (Reuters)

SAN FRANCISCO (Reuters) - A new videogame company is aiming to challenge the big three console makers by providing a "cloud-based" gaming system promising on-demand access to games and no lag time.

The fledging company, called OnLive, said its service will allow users to play games on any TV and nearly any personal computer -- even stripped-down netbooks and PCs without graphics processors.

A console slightly larger than an iPhone connects TVs and broadband connections to the OnLive service, and is operated via a wireless controller. OnLive delivers games run on servers in the "cloud," rather than locally on a PC or a console.
Cloud computing and cloud services

- IT as a service
  - Utility model
  - Hosted... managed...
  - Ideally, scalable, available, secure, efficient
  - Pay per use, no upfront cost
  - Handle peak loads
  - Share information
- Enabled by connectivity, VM technology, online/offline technology
WaaS – Whatever as a Service
Challenges for cloud providers

- Scalable/available Multi-tenant infrastructure
- Privacy/security
- Business models, SLAs (and offering different ones to different customers)
- Auditing
- Efficient resource utilization
- Usability
- Offline use
- New design patterns/models (application-driven)
Handle with care...

Tuesday, February 17, 2009, 10:50 AM PST:

Unfortunately, database file recovery has been unsuccessful and I won’t be able to recover members' bookmarks from the Ma.gnolia database. This means that the public bookmark recovery tools are the only source for recovering your bookmark collections.

If you are interested in hearing more about what happened, the history of Ma.gnolia in general, and future prospects, you can watch the latest Citizen Garden podcast below, which was recorded last week. As I mention in this podcast, I am working on relaunching Ma.gnolia as a private service on a more robust infrastructure in the coming months. I'll update this page and the twitter account with those and any other developments.
Five is enough...

• "I think there is a world market for maybe five computers..." (1943)
  • Thomas Watson (1874-1956), president and chairman of IBM
SaaS and SOA, Mashups...

- Originally meant for humans, use via browser
- Lately, saas apps provide api... distinction between saas and soa is blurring
  - Even if saas NOT born or dev with the idea of being components, not designed for this, sometimes they evolve into them
  - Examples of gmap and gdoc
- A lot more interesting services available
- Mashuppable
aaS mindset...

• Naturally leads to thinking API and thinking $aaS$
• Maybe it’s the fashion,…
• Think SME
• Everything is more “accessible”, even our own components
Ease of deployment/management

- Analogous to simplicity in mashup models
- I still have to develop my service/service composition/mashup, but
  - No need to involve our IT dept or to purchase machines
  - No need to wait 3 weeks because you found out that your blade server consumes more energy than your wiring can support
  - No need to install/manage the dev platform
  - Deploy with a click (and all the other goodies)
Share the integration logic

- PaaS can do for integration logic what SaaS / SOA do for services
  - Share, reuse
- Possible/easier to share programming knowledge, and specifically mashup and composition knowledge
Composition languages
Composition platforms
Transactional compositions
Office / enterprise automation, for professionals

SOA
Services
Standards
Middleware protocols
Intra/inter enterprise automation, for professionals

BPM
Simplified deployment/mgmt
Scalability,...
Broad svc offering, Accessibility, Sharing
Components, composition tools,
Middleware back in the platform?

Mashups
Simple compositions
Separation simple/complex
Simpler standards
Coarse components
UI integration
Targets non-professionals
Relaxed non-functional requirments
Situational applications? Rapid prototyping?

Cloud
Domain Expert Programming

• Between flexible processes and quasi-situational application
  • “Process automation” at large
• Only way out: let domain expert do the “coding” (and the prototyping, and the testing)
What do we need

• Programming languages not really for domain experts, or not for automation of enterprise processes
  • Either target problem or target users do not match or fit
• Offset complexity with knowledge reuse
  • Odds are, people (maybe experts) have done the same thing before
• Reuse
  • Insights on which components to use
  • mashup/composition knowledge
• (Not talking about semantic web, goal-driven automated composition,....)
Directions (?)

- IT becomes commodity
- Mashups for the People
- Some key challenges:
  - How to make composition models/tools that are simple enough and useful enough?
  - How to build reusable components? What are the characteristic of a “good” reusable component?
  - Can only domain-specific models succeed?
Thanks
References

- Outsourcing Business to Cloud Computing Services: Opportunities and Challenges
- Hewlett-Packard: The benefits of combining business-process outsourcing and service-oriented architecture,
- S. Murugesan. Understanding Web 2.0 IEEE IT Professional 9(4)
- Shane Robison. The next wave: Everything as a service, Executive Viewpoint,
- MapReduce: simplified data processing on large clusters
  - by: Jeffrey Dean, Sanjay Ghemawat
References (cont.)