Service Integration Architectures

An Introduction to Process Integration, Service Integration, and Mashup Integration
Before messages are exchanged:
- A: subscription to message newQuote
- B: subscription to message quoteRequest
- C: subscription to message newQuote (may be indep of subscription A)

At run time: processing of a request for quote.
- 1: publication of a quoteRequest message
- 2: delivery of message quoteRequest
- 3: synchronous invocation of the getQuote function
- 4: publication of a newQuote message
- 5: delivery of message newQuote
- 6: invocation of the createForecastEntry procedure
And, many other benefits...

- Asynchronous
- Guaranteed delivery
- Transactional
- Monitoring, logging
inventory management (subscriber)

ERP (subscriber)

dispatcher (publisher)

shipping (subscriber)

month-end closing (subscriber)

Service Bus

SOAP

JMS

new PO

SMTP

REST?
Integration at the application and UI level

- BPM/workflows
- Service composition
- Mashups
- Universal Integration
- Lifecycle management
Dimensions

- Component and state
- Granularity
- Control flow and composition model
- Data/Data passing model
- UI capability
- “Robustness” and security
- Instance-model relationship
- Tight vs light binding model
- Target developer
- Separation of simple from complex
- Interaction model
- Dev lifecycle
Business Processes, BPM, Workflows

- **Business process**: a set of steps executed to achieve a business goal
- **BPM**: refers to automation, monitoring, improvement of BP
- **Workflow**: formal description of a process, suitable for automated execution
Business Process Automation via Workflow Technology

- Business processes are at the heart of what companies do
  - Obvious goal: better/cheaper/faster processes
- How? Automation!! (1990s)
  - Rise of workflow technology

Automate business logic, information flow
Elements of Workflow Models

- Workflow modeling framework and tool
- Information modeling framework and tool
- Organization modeling framework and tool
- A workflow engine

Automate business logic, information flow
Petri nets

both states (places) and activities (transitions)

13
Data and data access model

- Blackboard vs data flow
- “traditional” vs web services-typed
  - XML, WSDL messages
Organization model - 2

- Sales
  - Consumer
    - Desktop
      - Sales exec 1
  - Enterprise
    - Laptops
      - Sales exec 15
    - Services
      - Sales exec n
  - Services
    - Sales exec z
Workflow system architecture

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

Workflow model repository

workflow engine

analytics engine

execution logs

SAP adapter

Email adapter

Custom adapter

Account mgmt
completed work items

resource broker

workflow engine

resource 1

resource 2

resource n

outbound queues

workflow definitions

workflow designer

inbound queue
Many Benefits (in theory....)

• Reduced need for human labor
  • No repeated data entry
  • cheaper

• Faster processing
  • Information flows automatically, 24x7x365

• More reliable
  • No case left behind

• Paperless and secure
The Two Sides of Business Process Management

• Execution/Automation
  • WF, megaprogramming
  • Flexibility

• Management
  • Visibility, Monitoring
  • Analysis, Optimization
  • Modeling, Re-engineering

• You can have one without the other (and in many cases, you should)
(Enterprise) Application Integration

Workflow is not the only integration technology
WS-BPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction occurs through Web Service interfaces.
Workflow system architecture

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

SAP adapter

Email adapter

Custom adapter

Account mgmt
Elements of WS composition middleware

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

- Component and state
- Granularity
- Control flow and composition model
- Data/Data passing model
- UI capability
- “Robustness” and security
- Instance-model relationship
- Tight vs light binding model
- Target developer
- Separation of simple from complex
- Interaction model
- Dev lifecycle
Mashups
Mashups: an example application

- The **HousingMaps** example
  - Application: [http://www.housingmaps.com](http://www.housingmaps.com)

- **Development concerns**
  - **Data extraction**: apartment offers are extracted from the Craigslist
  - **Data passing**: extracted data must be communicated to Google Maps
  - **New layout**: the apartment listing and the composite application need a new respectively own layout

- **Problems**
  - Development by hand: hacking of the Craigslist, programming of the Google Maps AJAX API

>> hard and time consuming task!
UI integration: research challenges

• Assist mashup development!
• Identify a good component model
  • Characterize relevant properties of UI components
  • Componentize UI + descriptors
• Derive an appropriate composition model
  • Synchronize components
  • Handle data passing
• Enabling even web users to do UI integration
  • Provide for hosted development and execution
An UI Component model

The Mixup UI component model

- **State**
  - internal state depends on visible UI

- **Events**
  - notify other components of internal state changes;
  - high-level, UI-centric semantics.

- **Operations**
  - allow the querying and modification of the internal state;
  - again: high-level, UI-centric semantics.

- **Properties**
  - may be used as design-time configuration parameters;
  - or as runtime state indicators.

- **Presentation modes**
  - Express visualization (visible, hidden, minimized,...) or life cycle information (instantiated, ready, busy,...)
The Mixup editor for UI integration

List of application components available for the mashup. Additional components may easily be loaded into the editor by referencing the respective online resource.

Graphical model of the composition logic.

Mahup logic modeling canvas.

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
Let’s build and expose our first mashup, in 2 minutes

(but…. Is it a mashup???)
Dimensions

- Component and state
- Granularity
- Control flow and composition model
- Data/Data passing model
- UI capability
- “Robustness” and security
- Instance-model relationship

- Tight vs light binding model
- Target developer
- Separation of simple from complex
- Interaction model
- Dev lifecycle
Universal Integration

- Component browser
- Service component
- Data flow connector
- UI component
- Composition canvas
- Events and operations
Goals and research challenges

• Creating a universal **component model**
  • Abstract **common features** of UI and services
  • Recognize **differences** wrt traditional service integration

• Identifying the key abstractions for a universal **composition model** that is simple but effective
  • Data flows
  • Stateful vs. stateless components
  • Components with and without UI

• Offering **integration as a service**
  • **Hosted** development and execution
Guiding principles

- **Universality**: UI, application logic, and data
- **Synchronization and orchestration**: bring together UI and application/service logic
- User and programmatic **input**: UIs + APIs
- **Lightweight** composition model: 2 pages spec, no transaction, no exceptions/compensations
- Complexity **inside** components
- **Extensibility** and openness
- **Standard** layout and graphics tools
The mashArt component model

• **State**
  - UI components: given by the “state” of the *user interface*
  - Services: internal “*program state*” (e.g., of an order service)

• **Events**
  - **Publish** state changes (data formatted as name-value pairs)

• **Operations**
  - **Change** state of *perform* operation invocations

• **Configuration** properties
  - E.g., layout settings or authentication options
  - **Constructor** parameters
Dimensions

- Component and state
- Granularity
- Control flow and composition model
- Data/Data passing model
- UI capability
- “Robustness” and security
- Instance-model relationship
- Tight vs light binding model
- Target developer
- Separation of simple from complex
- Interaction model
  Dev lifecycle
Lifecycle management
The motivating scenario

- Projects involve managing people working on “artifacts”
  - Deliverables, Papers, Proposals, complex artifacts
  - Possibly disseminated: SVN, Mediawiki, Gdocs, etc.

- How do we manage the evolution of these artifacts?

<table>
<thead>
<tr>
<th>Nome attività</th>
<th>Durata</th>
<th>Inizio</th>
<th>Fine</th>
<th>Pre</th>
<th>Nomi risorse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaboration</td>
<td>11 g</td>
<td>lun 2/9/09</td>
<td>lun 2/23/09</td>
<td>Fabio, Marcos, Florian, Maurizio</td>
<td></td>
</tr>
<tr>
<td>Final Assembly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EU Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Microsoft Project! = only for planning
Another alternative:
- Workflow systems (automation)

Drawbacks:
- Rigid, complex,
- For programmers
- Flow of actions

Adaptive workflows
- Flexible
- Still complex

Thus, people don’t usually model the evolution of these artifacts
Proposal: URLM

- A system for universal resource lifecycle management
- Designed to manage URI-identifiable artifacts
- Decoupled from the artifacts it manages
Example: Lifecycle of our SotA

<table>
<thead>
<tr>
<th>Phase</th>
<th>Action</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaboration</td>
<td>+ Change access rights + Notify reviewers</td>
<td></td>
</tr>
<tr>
<td>Internal Review</td>
<td>+ Change access rights + Notify reviewers</td>
<td></td>
</tr>
<tr>
<td>Final Assembly</td>
<td>+ Generate PDF + Change access rights</td>
<td></td>
</tr>
<tr>
<td>EU Review</td>
<td>+ Change access rights + Notify reviewers</td>
<td></td>
</tr>
<tr>
<td>Publication</td>
<td>+ Post on web site + Change access rights</td>
<td></td>
</tr>
</tbody>
</table>

Notation

- Terminal nodes
- Phase
- + label
- Action
- Transition

Gelee meeting, Feb 17th, 2009
Gelee Architecture

Web UI

- Lifecycle Designer
- Lifecycle execution widgets
- Monitoring cockpit
- Config

Auth

REST/SOAP API

Resource Abstraction

- Resource Manager
- Adapters

- Resources managers registry
- Resource instances

Lifecycle Management

- Design time Manager
- Runtime Manager
- Monitor
- Config. Manager

- Lifecycle models/templates
- Lifecycle instances
- Users

GoogleDocs
Wikipedia
SVN
Blogger

Gelee meeting, Feb 17th, 2009

Work Plan
**Trento Meeting, January 28, 2009**

**Universal Resource Lifecycle Management**

**Mockup**

**Instances | Templates | Updates**

**New Instance**

- **Share**
- **Export**
- **View trace**

**LiquidPub SotA deliverable**

**Lifecycle**

- **Current phase:** Internal Review
- **Actions in execution:** None
- **Next phase(s):** Elaboration
- **Owner:** Ronald
- **Shared:** None
- **Watches:** 0

**SKO**

- **Name:** SotA deliverable
- **Author(s):** Ronald, Hao, Fausto
- **Access rules:** View

**Next phase:** Elaboration

* Click on the phase to perform the transition

**Configure actions**

- **Notify collaborators**
  - Collaborators

**Management**

**Go!**

**Back**
Dimensions

- Component and state
- Granularity
- Control flow and composition model
- Data/Data passing model
- UI capability
- “Robustness” and security
- Instance-model relationship
- Tight vs light binding model
- Target developer
- Separation of simple from complex
- Interaction model
- Dev lifecycle
SaaS and PaaS

- **Software as a service** refers to traditional, packaged SW applications offered... as a service
  - Now we talk about both web services and services on the web
  - No deployment or mgmt cost for customers. Pay per use – or varied business models.
- **Platform as a service**: infrastructural or development offered as a service
- How related? Enables a new approach to programming.
And from here...

- Programming becomes simplified (in theory)
- Domain experts become programmers. Mass programming. Flexibility and ease of change.
- Deployment must also be simplified. Deploy on the cloud.
- Programming as a service. Your code is not buried into your PC or our local SVN.
- Collaborative programming – and learning from others. Exploit the power of the crowd.
Evolution or Revolution?

- Technology is *evolutionary*
  - revolutionary aspects (dynamic service selection, Semantic Web services) are only on paper for now
- Standardization, adoption pace, impact are *revolutionary*
- Real impact: the Web and *composition awareness.* Expose via API.
References - composition

• Curbera et al - Web Services Platform Architecture. Prentice Hall
• Leymann and roller: Production workflows. Prentice Hall
• Papazoglou: Web services – Principles and Technologies. Prentice-Hall