

A Rule-based Approach of Creating and Executing Mashups

Emilian Pascalau



Adrian Giurca



Outline

- Short introduction to the environment
- Perspectives on service aggregation
- JSON Rules
- Use case
- Conclusions

Future Internet

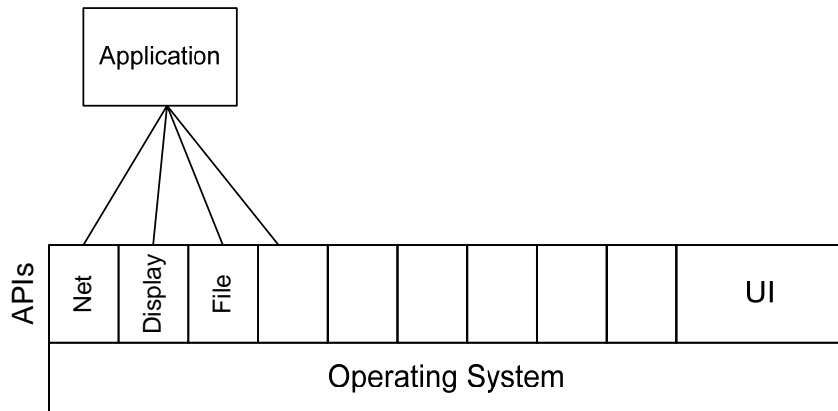
- Future Internet is that of *Internet of Services* and *Internet of Things*
- *“The Internet of Services is largely based on a service-oriented architecture (SOA), which is a flexible, standardized architecture that facilitates the combination of various applications into inter-operable services. The Internet of Services also uses semantic tools technologies that understand the meaning of information and facilitate the accessibility of content (video, audio, print). Thus, data from various sources and different formats can easily be combined and processed toward a wealth of innovative Web-based services.”*

SAP co-CEO Henning Kagermann

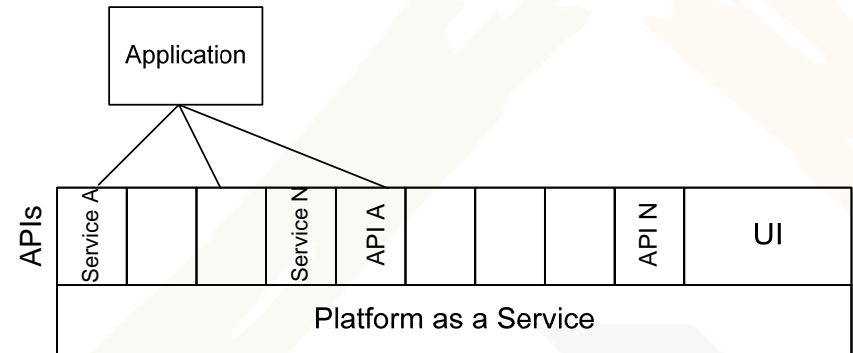
Perspectives on service aggregation 1

- Software as a Service (SaaS): *"service-based model of software is one in which services are configured to meet a specific set of requirements at a point in time"*. (Bennett et. al)
- Mashups – initially seen more like a tool approach

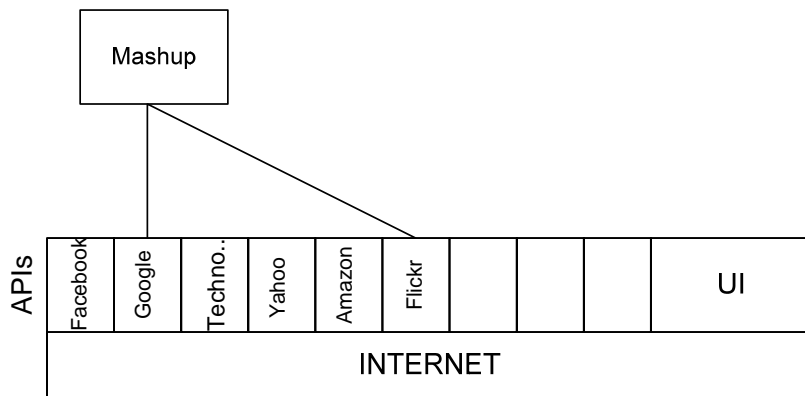
Perspectives on service aggregation 2



Computer model (David Berlind, ZDNet)



Software as a Service model



Mashups model (David Berlind, ZDNet)

What is a Web 2.0 application?

- Several applications pieced together
- Small
- Data is in the cloud
- Run on any device
- Customizable
- User defined
- Fast
- Can be easily shared (sent virally i.e. social networks)

A Rule-based perspective on service aggregation

- Players such as Google, Adobe are interested in using rules in the browser
- Ability to dynamically change the Web experience
- Rules are easy to be captured
- Offer greater flexibility
- Easy way to define behavior/interaction between services

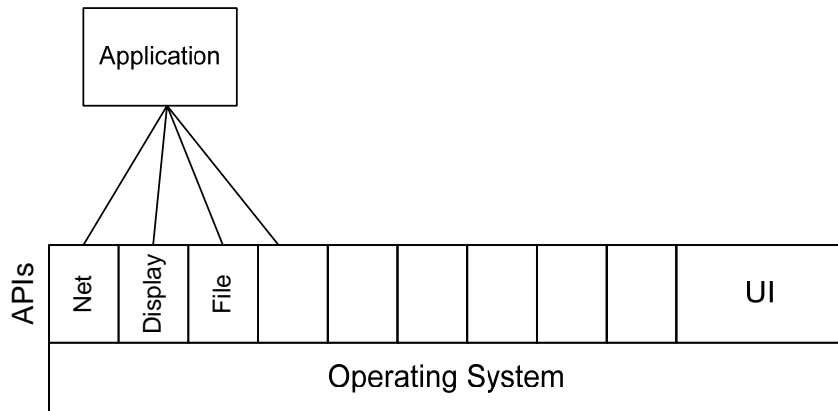
JSON Rules - Goals

- Move the reasoning process to the client-side
- Offer support for intelligent user interfaces
- Handle business workflows
- allow rule-based reasoning with semantic data inside HTML pages (reasoning with RDFa)
- create intelligent mashups – rule-based mashups
- enable users to collaborate and share information on the WWW (rule sharing and interchange)

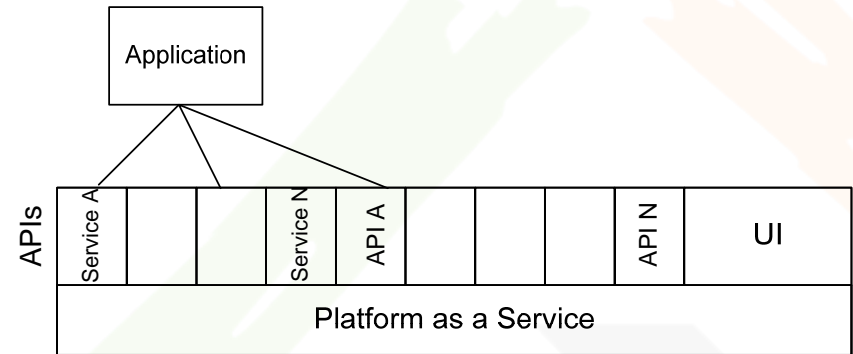
JSON Rules - Requirements

- Rules run in the browser
- Event-Condition-Action rules
- Rules defined on top of DOM structure
- DOM Events + user defined events
- Conditions address the content of the pages
- Actions defined by users (any JavaScript function calls)

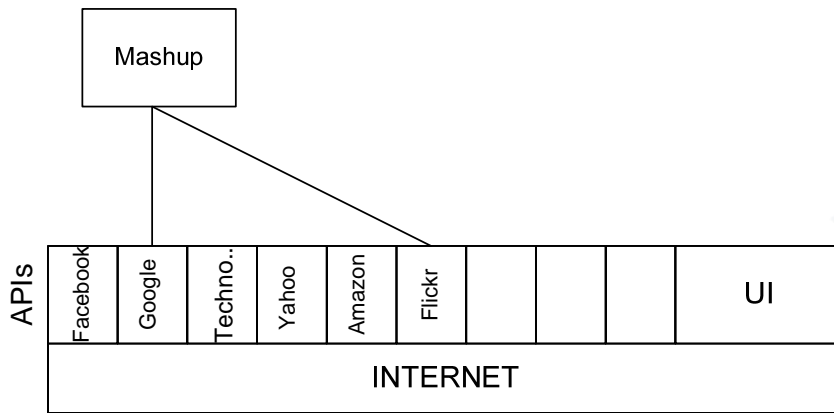
Perspectives on service aggregation 3



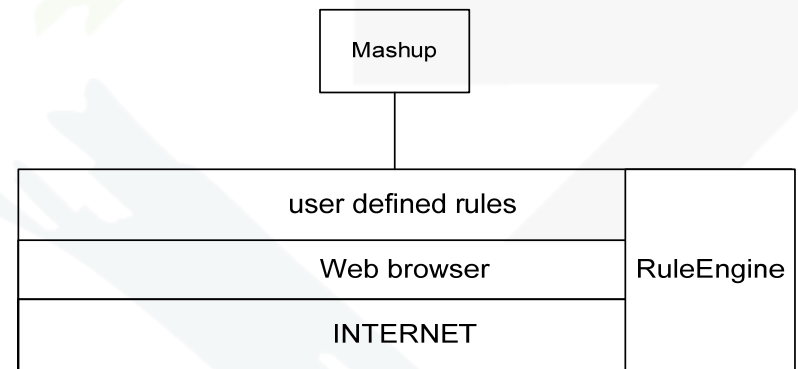
Computer model (David Berlind, ZDNet)



Software as a Service model



Mashups model (David Berlind, ZDNet)



JSON Rules – based model

Creating Mashups with JSON Rules – The problem

- Usually people working in IT are required to travel a lot, to relocate or to change jobs. When ever looking for a job there are at least several aspects that must be taken into account, especially for those who have families: the location of a job displayed on a map would be really appreciated. Being able to find out some facts (i.e. schools/universities, photos, shops, real estate agencies etc.) about the neighborhood is also important.

How could you do that?

- ...using:
 - Monster Job Search Engine (<http://jobsearch.monster.ca>)
 - Google Maps (<http://maps.gogle.com>)
 - Wikipedia (<http://www.wikipedia.org>)
 - Flickr (<http://www.flicker.com>)
- Inconveniences? A couple.... i.e.
 - Several open tabs
 - The need to copy and paste info into each tab
 - Time consuming
 - The need to remember info from different tabs which is not in sync with the current search

Refining the problem

- We are looking for a job using the Monster Job Search Service. Once the job is obtained the location is shown on a Google Maps and if it is possible, some supplementary information is provided.

Services involved

- Monster Job Search Engine
- Google Maps

Requirements

- All these services must interact in one page. This page is called *choreographer*.
- The search term must be inserted manually in a form invoking the Monster Job Search Service. The other services must react on the returned data.
- When the mouse is over a job, the job should be visually indicated on Google Maps.
- When the mouse is over a job, the information regarding the job, if any, should be retrieved from another service (such as Wikipedia).

Rule: Load services so that the Choreographer can use them

```
{ "id": "loadServices",  
  "appliesTo":  
    [ "http://www.jsonrules.org/examples/i3e/" ],  
  "eventExpression": { "type": "load" },  
  "condition": true,  
  "actions": [ "load('http://jobsearch.monster.ca')",  
               "load('http://maps.google.com')" ]  
}
```

Rule: Locate a job on Google Maps 1 - event

```
{ "id": "findJobLocation",  
  "appliesTo": [ "http://www.jsonrules.org/examples/i3e/",  
                 "http://jobsearch.monster.ca/" ],  
  "eventExpression" : { "type": "mouseover",  
                        "target": "$X" },
```

Rule: Locate a job on Google Maps 2 - conditions

```
"condition":  
["$X:HTMLDivElement(  
  className=='jobInfo' )",  
"$Y in 'child:$X'",  
"$Y:HTMLDivElement(className ==  
  'stackedViewCompany' )",  
"$Z in 'child:$X'",  
"$Z:HTMLDivElement(className ==  
  'jobPlace' )",  
"not ($Y==$Z)",  
"$T:HTMLInputElement(id=='q_d' )"  
  ,  
"$companyName ==  
  $Y.textContent",  
"$jobLocation ==  
  $Z.textContent",  
"$form:HTMLFormElement(  
  id=='q_form' )"],
```

```
<div class="jobInfo"  
style="width:673px;">  
...  
<div class="stackedViewCompany">  
  Lockheed Martin Canada  
</div>  
...  
<div class="jobPlace">  
  Ottawa , ON  
</div>  
  
<form id="q_form" action="/maps"  
...>  
<div class="srchcol controls">  
<input type="text" value=""  
  autocomplete="off"  
  maxlength="2048" tabindex="1"  
  title="Search the map"  
  name="q" id="q_d"  
  style="width : 33em;" />...  
</div></form>
```

Rule: Locate a job on Google Maps 3 - actions

```
"actions":[ "update($T, 'value',  
             '$companyName+$jobLocation')",  
            "autoSubmitForm($form)"]}
```

Conclusions – features of the rules based approach

- Mashups can be executed on any browser allowing JavaScript
- Flows can be defined on any component that is available in the service answer
- Behavior is defined declaratively
- Data can be accessed as usual (ATOM, RSS, RDF) or raw format
- Data mapping easily implemented
- UI



<http://jsonrules.googlecode.com>