Mashups

A Journey from Concepts and Models to the Quality of Applications

Cinzia Cappiello – Politecnico di Milano

Florian Daniel – Università di Trento

Maristella Matera – Politecnico di Milano

Learning Objectives

1. **Mashup definition and characterization**
   – Classifying dimensions, contexts of use, target users, benefits
2. **Mashup models**
   – Conceptual underpinning of mashups for different mashup types
3. **Mashup tools and composition paradigms**
   – How mashup models can materialize into platforms for assisted mashup development
4. **Mashup quality**
   – Quality issues for components and mashups, going beyond traditional quality models and practices

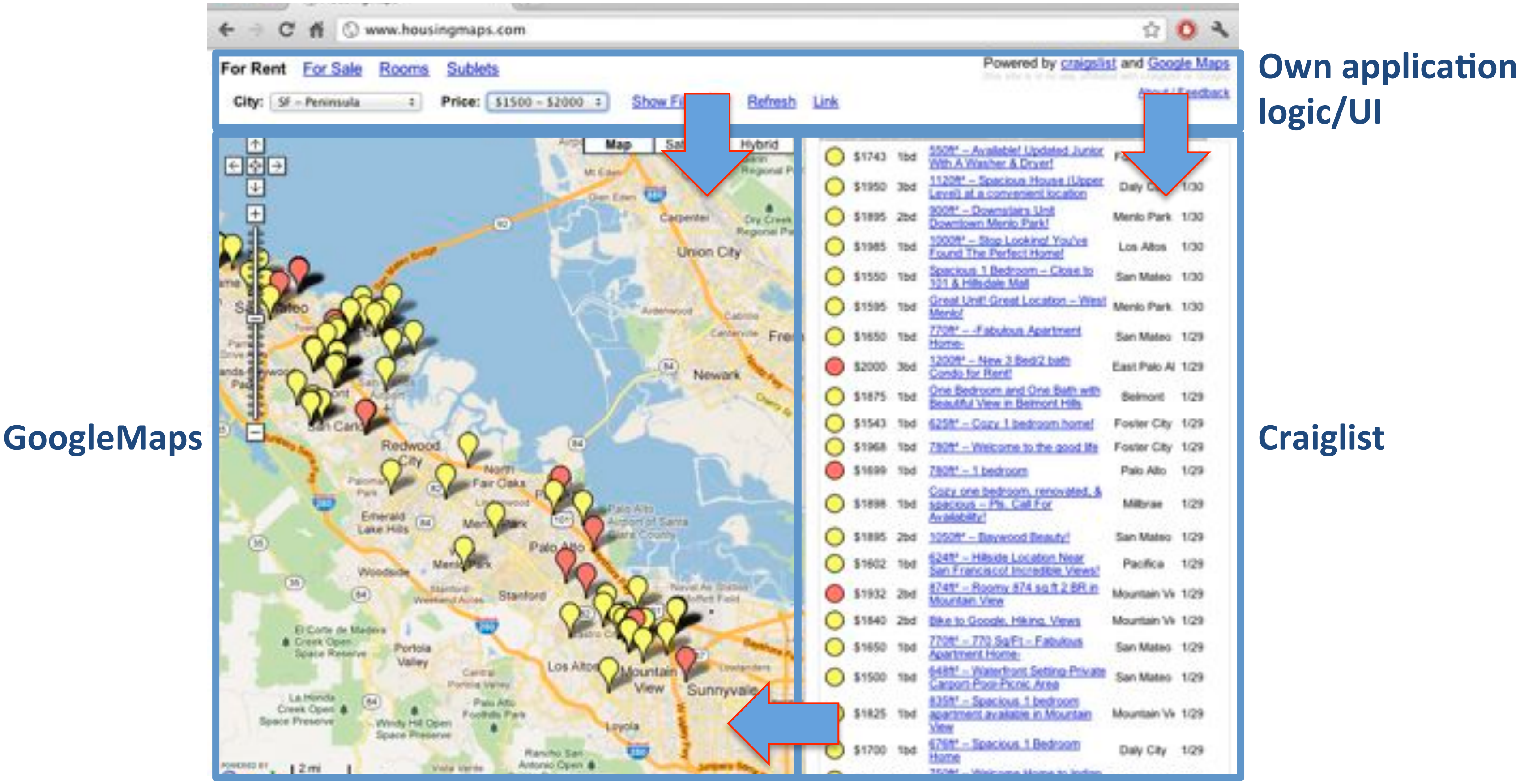
CORE ASPECTS AND DEFINITIONS

Technological and societal context

- From one-way communication medium (Web 1.0) to a distributed and democratic communication platform (Web 2.0)
- User-driven innovation
- SOA, SaaS, HTML5, sophisticated devices

Web mashups as innovative software **to reinterpret existing building blocks** by composing them in an value-adding manner

- The term mashup is widely used today
- Typical discussion points:
  - UI or not?
  - Web accessible resources or not?
  - Client-side technologies or also server-side languages?



The housingmaps.com mashup

Provides for the synchronized exploration of housing offers from *craigslist.com* and maps by *Google Maps*

Integration is the added value provided by the mashup

Mashup definition

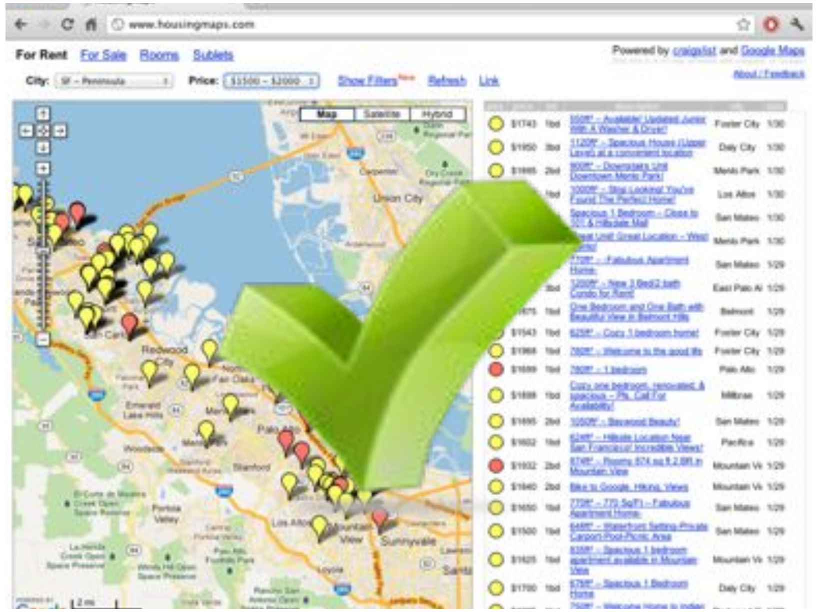
A **mashup** is an application that integrates two or more **mashup components** at any of the **application layers** (data, application logic, presentation layer) possibly **putting them into communication** among each other

→**Mashup component**: any piece of data, application logic and/or user interface that can be reused and that is accessible either locally or remotely

→ **Mashup logic**: is the internal logic of operation of a mashup; it specifies the invocation of components, the control flow, the data flow, the data transformations, and the UI of the mashup

The added value...

No added value



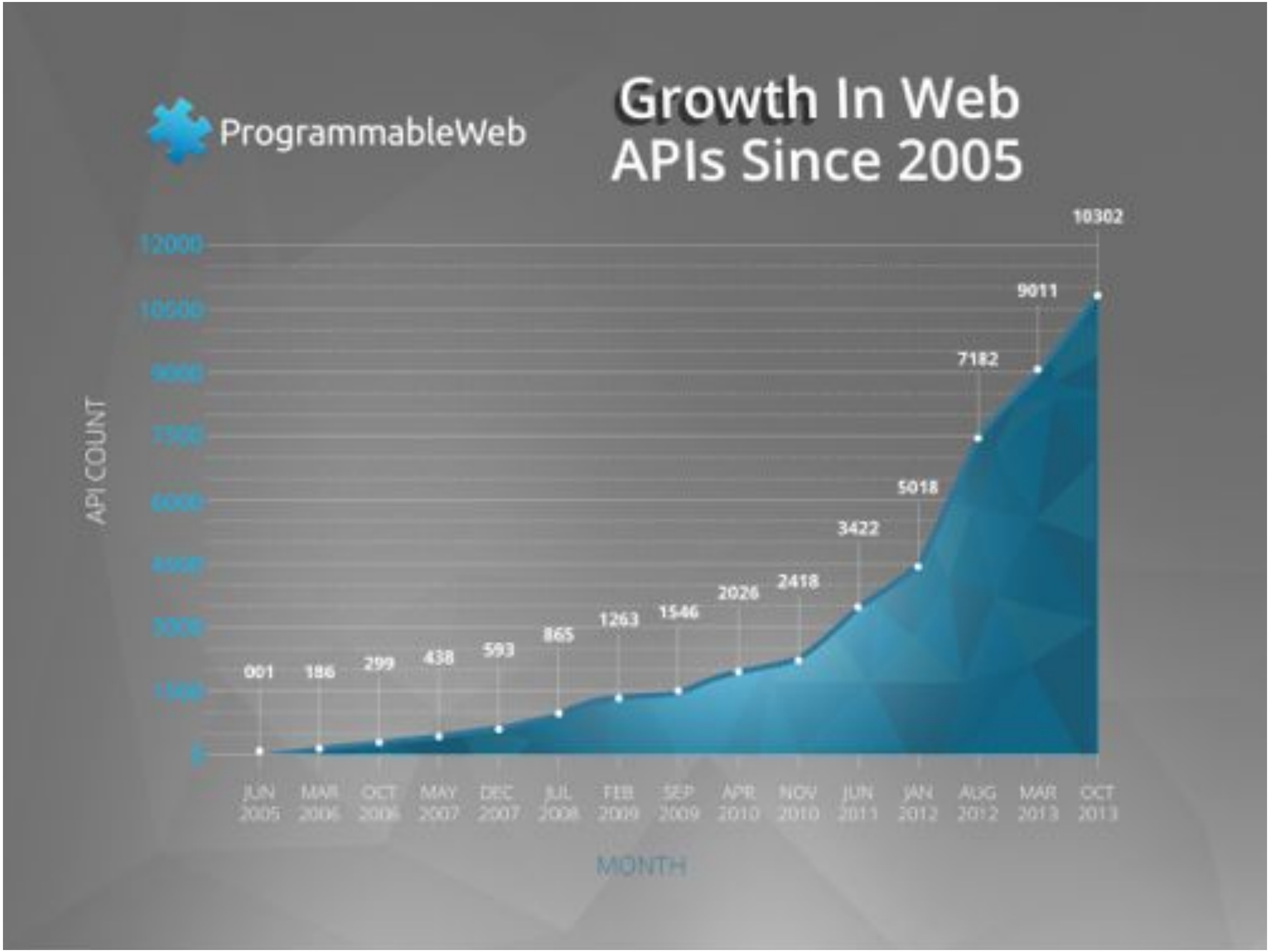
Additional information, functions, visualizations!

Other definitions

“**Web-based resources** consisting of dynamic networks of interacting components” (Abiteboul et Al., 2008)

“**API enablers**” (Ogrinz, 2009), to create an own API where there is none

“**Combination of content** from more than one source into an integrated experience” (Yee, 2008)



ProgrammableWeb
Growth In Web APIs Since 2005
API COUNT
12000
001
186
299
438
593
865
1263
1546
2026
2418
3422
5018
7182
9011
10302
JUN 2005
MAR 2006
OCT 2006
MAY 2007
DEC 2007
JUL 2008
FEB 2009
SEP 2009
APR 2010
NOV 2010
JUN 2011
JAN 2012
AUG 2012
MAR 2013
OCT 2013
MONTH

Mashup Ecosystem

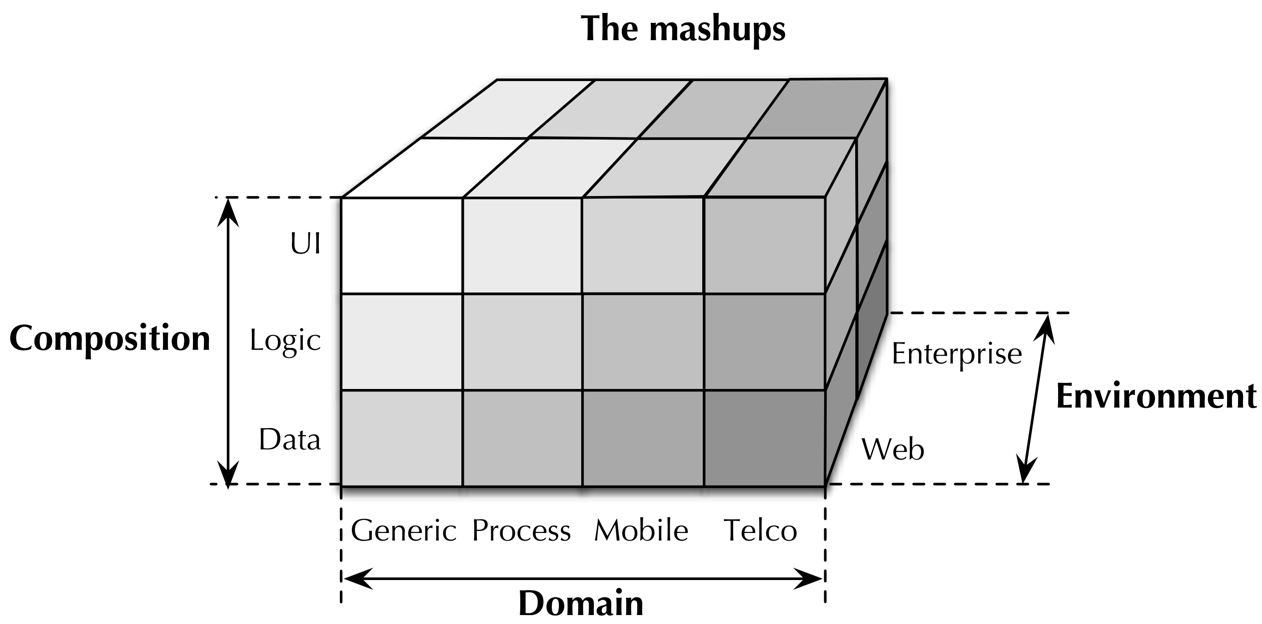
(b) All time most used tags to describe mashups

Snapshot from programmableweb.com (October 2013)

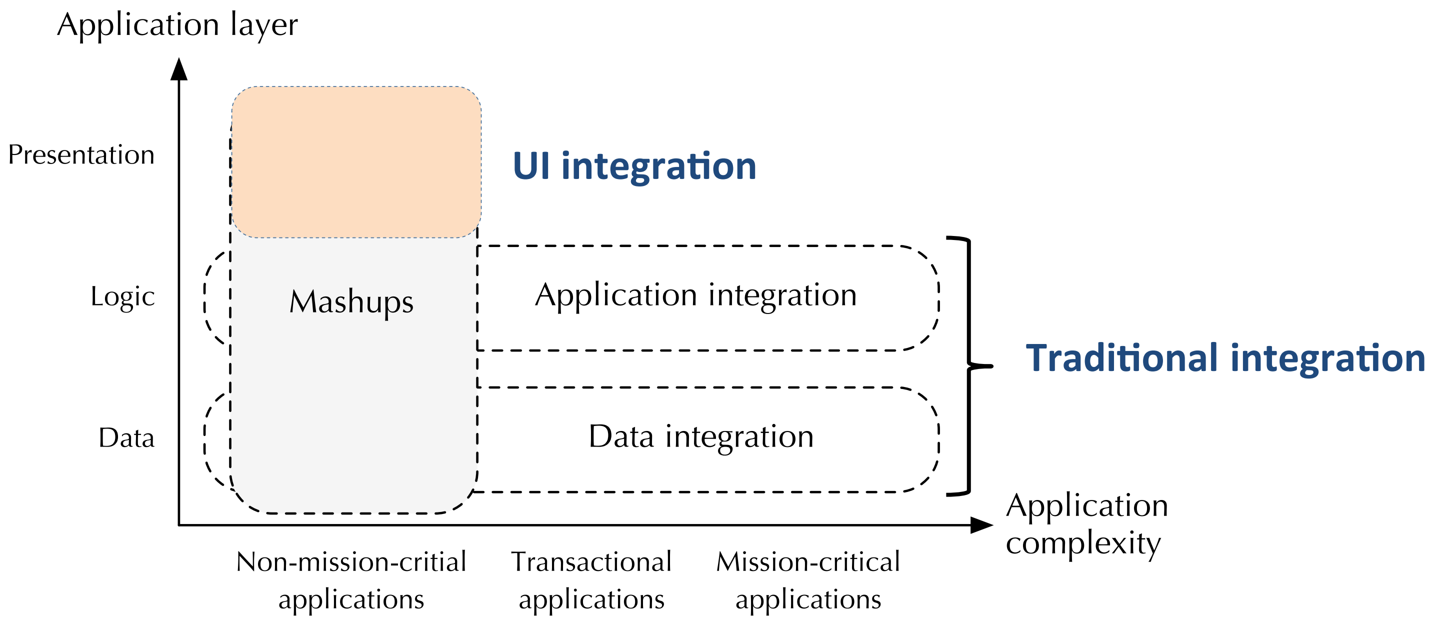
However…

- There are many applications that would not qualify as mashups
- The classification does not help characterize the mashup ecosystems from an engineering perspective

Mashup cube



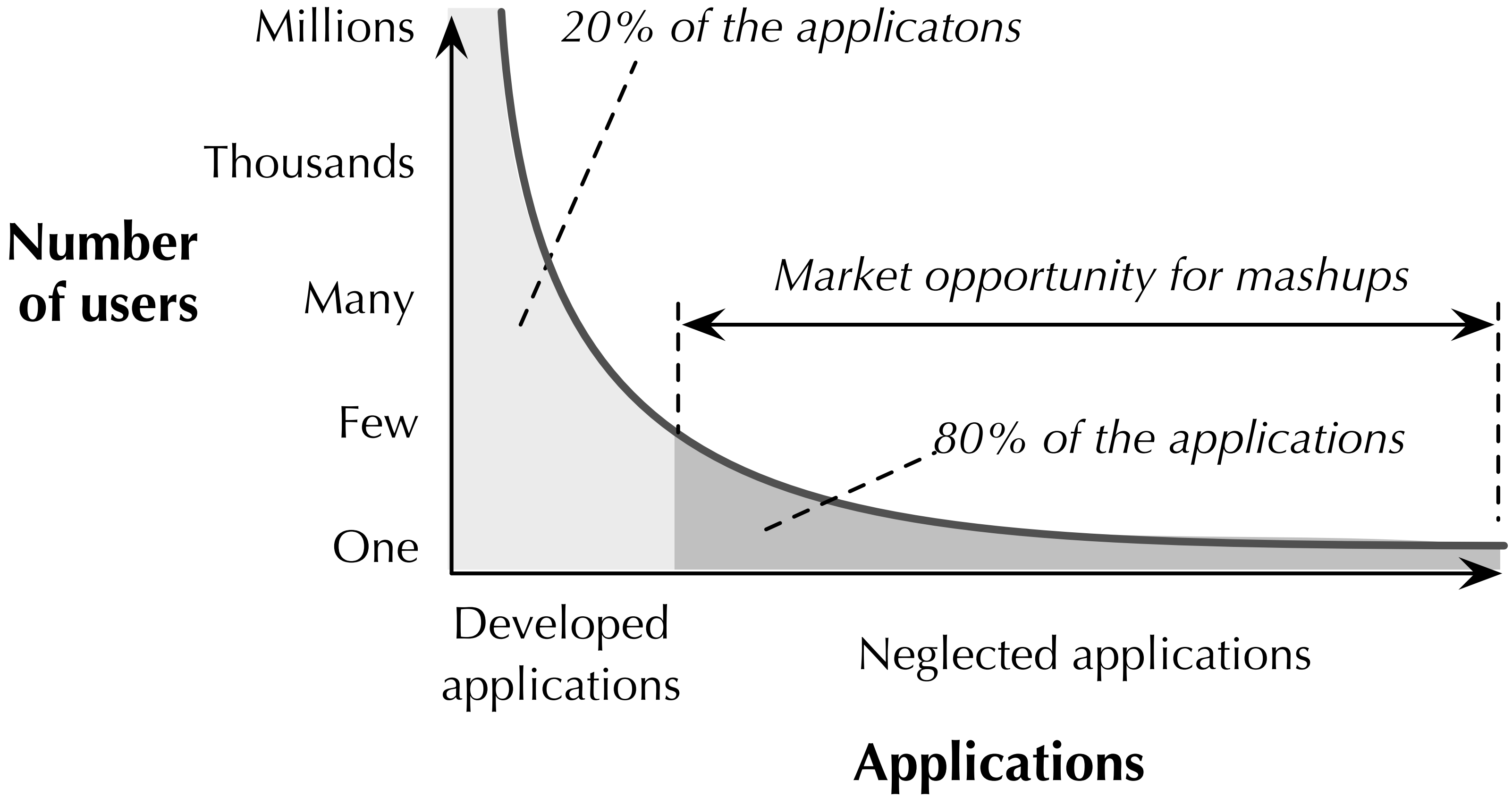
Three different perspectives on the mashup ecosystem



Mashup positioning in relation to other integration practices

Mashups introduce integration at the presentation layer and typically focus on non-mission-critical applications

The long tail model



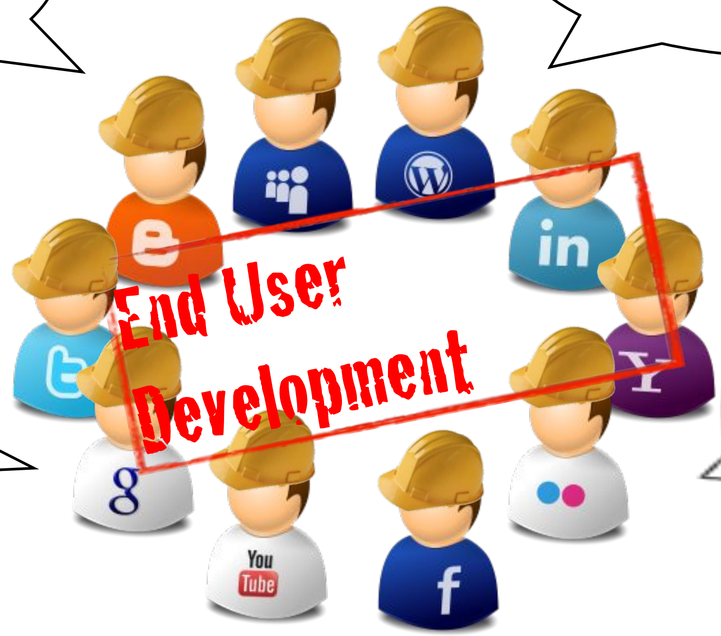
The long tail of the software market and its opportunities for mashups

Computer-assisted composition

Benefits

Knowledge transfer from end-users to developers

Higher satisfaction for end-users



Reduced costs for product evaluation

User involvement in the creation of applications

Other benefits

- Easy development of situational applications for **power users**
- Fast prototyping for **developers**
- Increased ROI for **SOA investments**
- Increased visibility by **content/component providers**

The research perspective

- Mashup development is non-trivial
  - A very large set of **(heterogeneous) technologies** and integration techniques
  - New technologies and interaction modalities emerge at **fast pace**

- Luckily, mashups typically **work on the "surface"**
  - **Reuse** of existing components - neglecting the complexity hidden behind the service's external interface
  - **Composition of the outputs** of (much more complex) software systems

- The work of developers can be facilitated by **suitable abstractions**, **component technologies**, **development paradigms** and **enabling tools**

- **Mashup development practices** are increasingly becoming the very object of scientific investigations

Part I

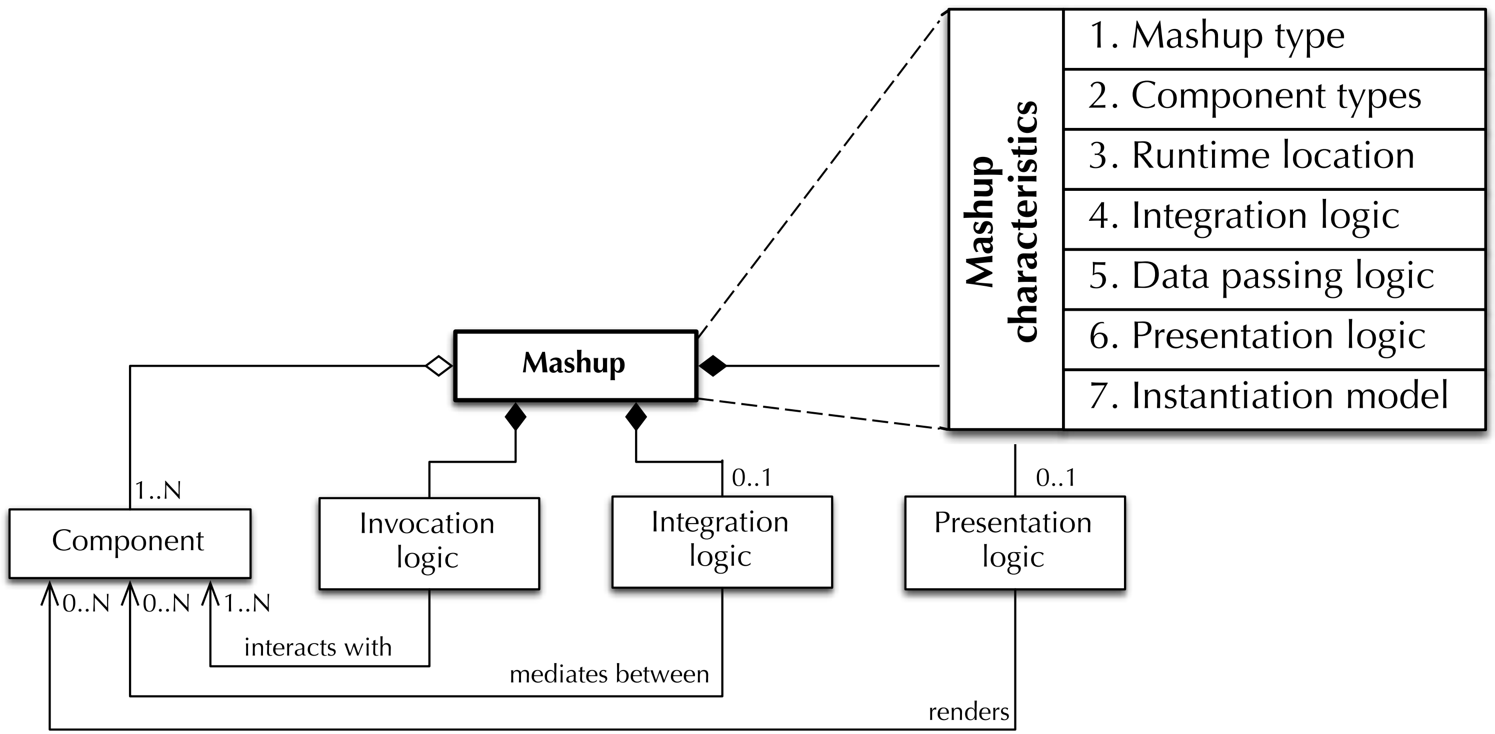
MASHUP MODELS

Learning Objectives

1. **Introducing models for different mashup types**

2. **Introducing typical architectural patterns**

3. **Identifying the peculiarity of UI integration**



Basic mashup model

A mashup integrates a set of components, possibly puts them into communication, and optionally renders results or components

Mashup characteristics	
	1. Mashup type
	2. Component types
	3. Runtime location
	4. Integration logic
	5. Data passing logic
	6. Presentation logic
	7. Instantiation model

→ Positioning the mashup at one or more of the three layers of the application stack depending on the output of the mashup

- **Data mashups**
  - Fetch data from different resources, process them, and return an integrated result set
- **Logic mashups**
  - Integrate functionality published by logic or data components
- **User Interface (UI) mashups**
  - Combine the component's native UIs into an integrated UI; the components' UIs are possibly synchronized among each other
- **Hybrid mashups**
  - Span multiple layers of the application stack, bringing together different types of components inside one and a same application; integration happens at more than one layer

Mashup characteristics	
	1. Mashup type
	2. Component types
	3. Runtime location
	4. Integration logic
	5. Data passing logic
	6. Presentation logic
	7. Instantiation model

→ Determining what kind of invocation, integration and presentation logic can be adopted for building the mashup

- **Data components**
  - RSS and Atom feeds, XML JSON, CSV and similar data resources, web data extractions, micro-formats, SOAP or RESTful services that are **used as data services only**

- **Logic components**
  - SOAP and RESTful web services, JavaScript APIs and libraries, device APIs, and API extractions

- **UI components**
  - Code snippets and JavaScript UI libraries, Java portlets, widgets and gadgets, web clips and extracted UI components

Mashup characteristics	
	1. Mashup type
	2. Component types
	3. Runtime location
	4. Integration logic
	5. Data passing logic
	6. Presentation logic
	7. Instantiation model

→ Possible architectural configurations, compatible with the requirements of the chosen components

- **Client-side mashups**
  - e.g., UI mashups
- **Server-side mashups**
  - e.g., data and logic mashups
- **Client-server mashups**
  - e.g., hybrid mashups with user interfaces

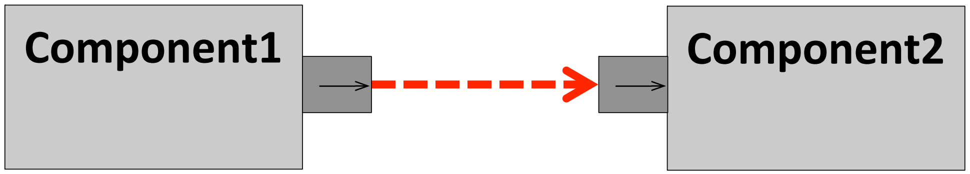
Mashup characteristics	
	1. Mashup type
	2. Component types
	3. Runtime location
	4. Integration logic
	5. Data passing logic
	6. Presentation logic
	7. Instantiation model

→ How components communicate with each other

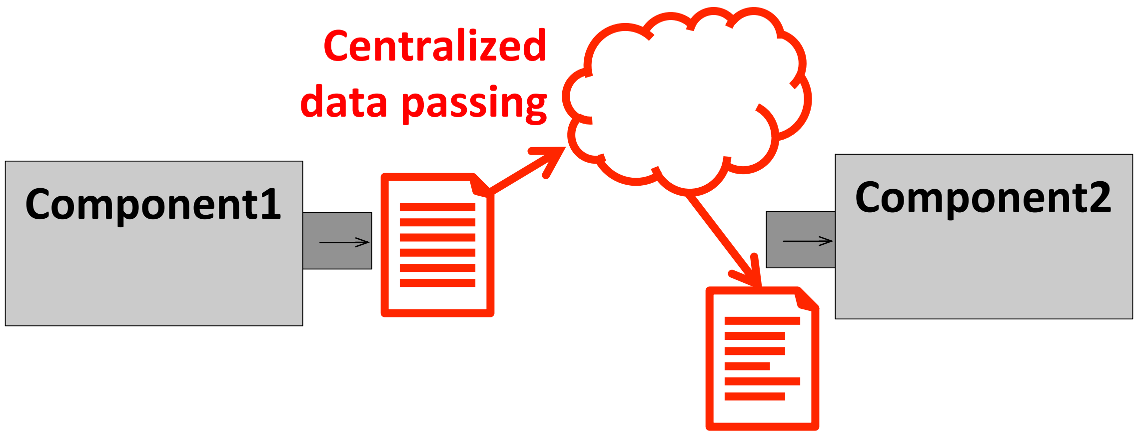
- **UI-based integration**
  - The UI of the mashup acts as a container
  - Components run in a completely isolated fashion
- **Orchestrated integration**
  - Centralized composition logic, orchestrating component execution
- **Choreographed integration**
  - Each component participating in a choreography is individually able to send and receive messages
  - The mashup puts into place only the communication infrastructure

Mashup characteristics	
	1. Mashup type
	2. Component types
	3. Runtime location
	4. Integration logic
	5. Data passing logic
	6. Presentation logic
	7. Instantiation model

→ How components exchange data



– **Direct data passing**



– **Blackboard vs. Shared memory**

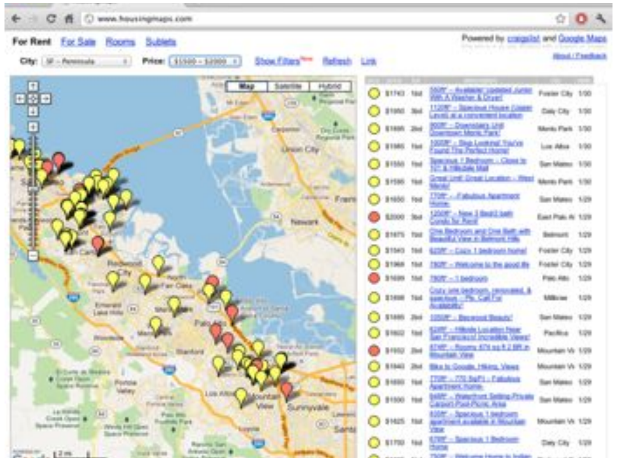
– **Mediated data passing**

Mashup characteristics

1. Mashup type
2. Component types
3. Runtime location
4. Integration logic
5. Data passing logic
6. Presentation logic
7. Instantiation model

6. Presentation logic → How components and their outputs are visualized in the mashup's UI

Reuse of components' UIs



Ad-hoc UIs

Mashup characteristics	
	1. Mashup type
	2. Component types
	3. Runtime location
	4. Integration logic
	5. Data passing logic
	6. Presentation logic
	7. Instantiation model

→ How long an instantiated mashup is running

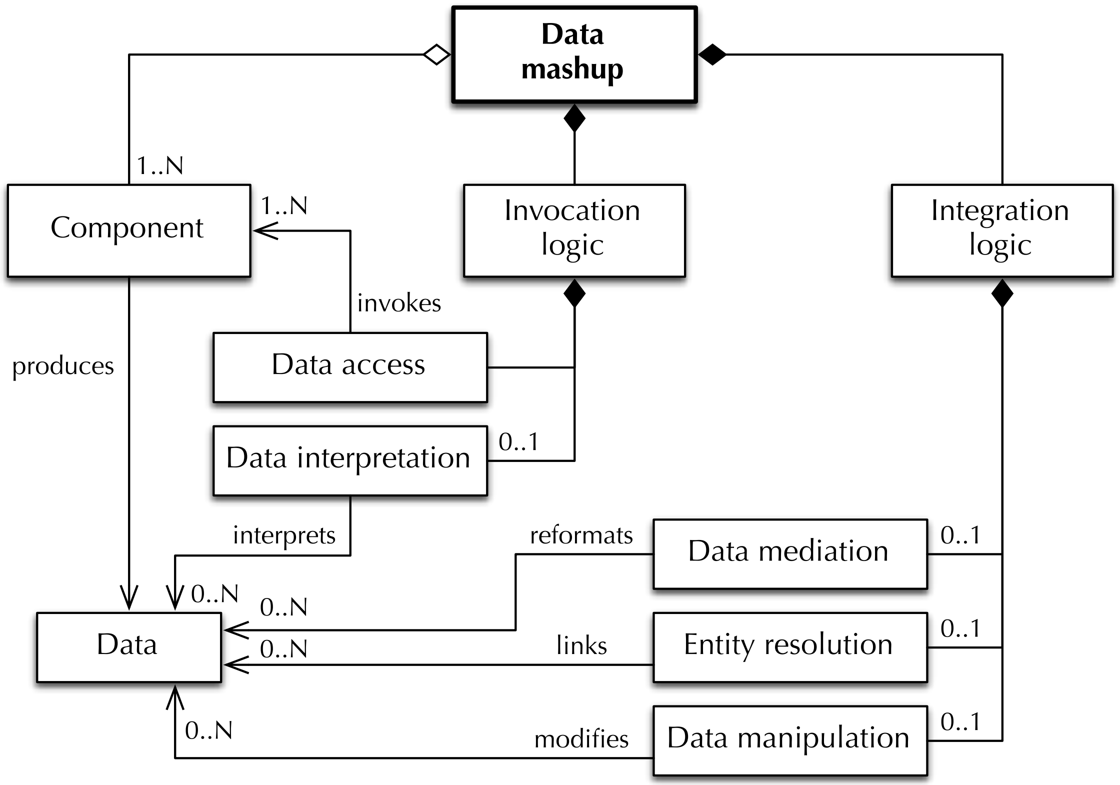
- **Stateless**
  - No internal state for their execution, ex.: data mashups
- **Short-living**
  - Last the time of a user session, ex.: UI mashups
- **Long-living**
  - Survive across different user sessions, ex.: process mashups

1. **Data Mashups**

2. **UI Mashups**

Data Mashups

- **Integrate components at the data layer** of the application stack by fetching data from different data services or Web resources, processing them, and returning an integrated result set
- **No presentation layer**
- Output: typically **published as a data source**
- Core integration practice: **data integration**

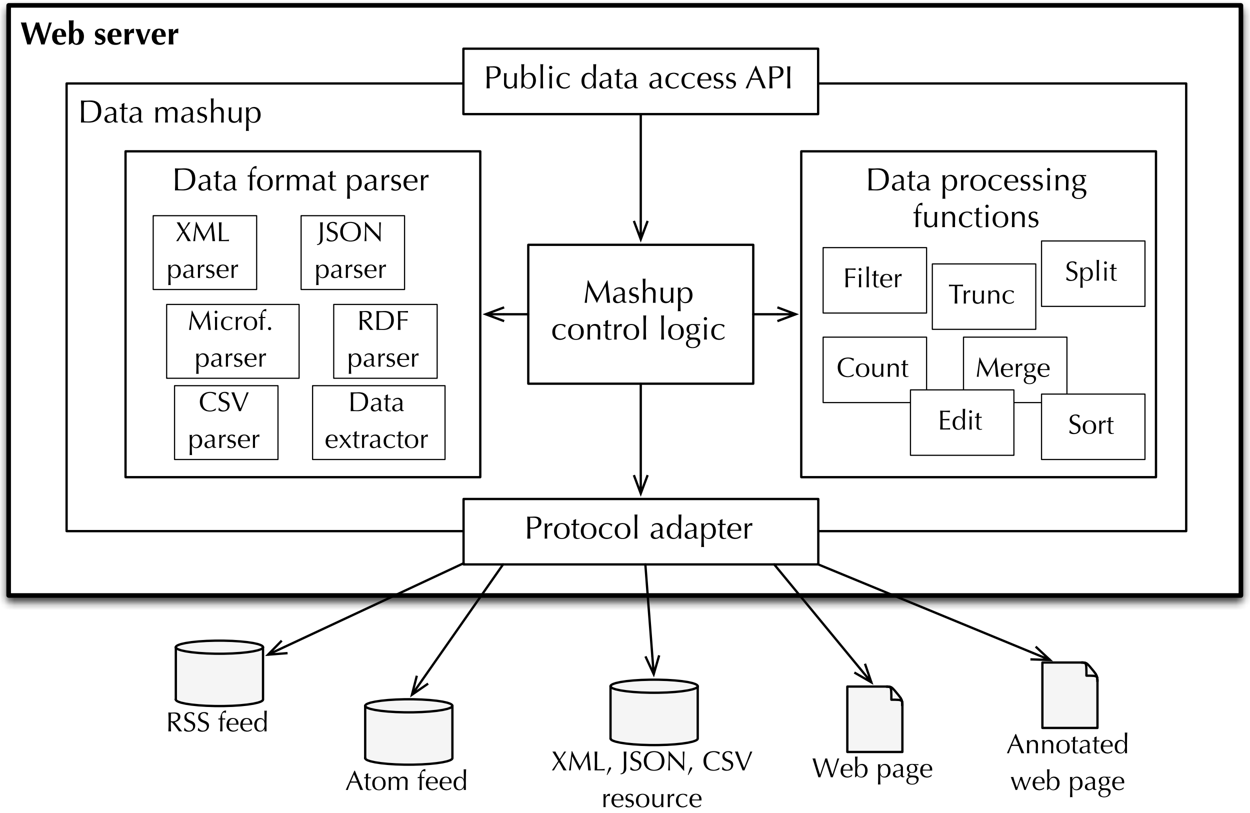


A conceptual model for data mashups

Data mashups fetch data from different sources and integrate them, mix them, filter them, process them, etc., so as to produce a unified data set as output

Compared with data integration…

- Data mashups are a Web-based form of data integration, intended to solve different problems
- Covering the “long tail” of data integration requirements
  - Very specific reports or ad-hoc data analyses
  - Simple, ad-hoc data integrations providing “situational data” that meet short term needs
  - Non-mission-critical integration requests



Different types of data components

Orchestrated integration

Direct data passing or blackboard approach

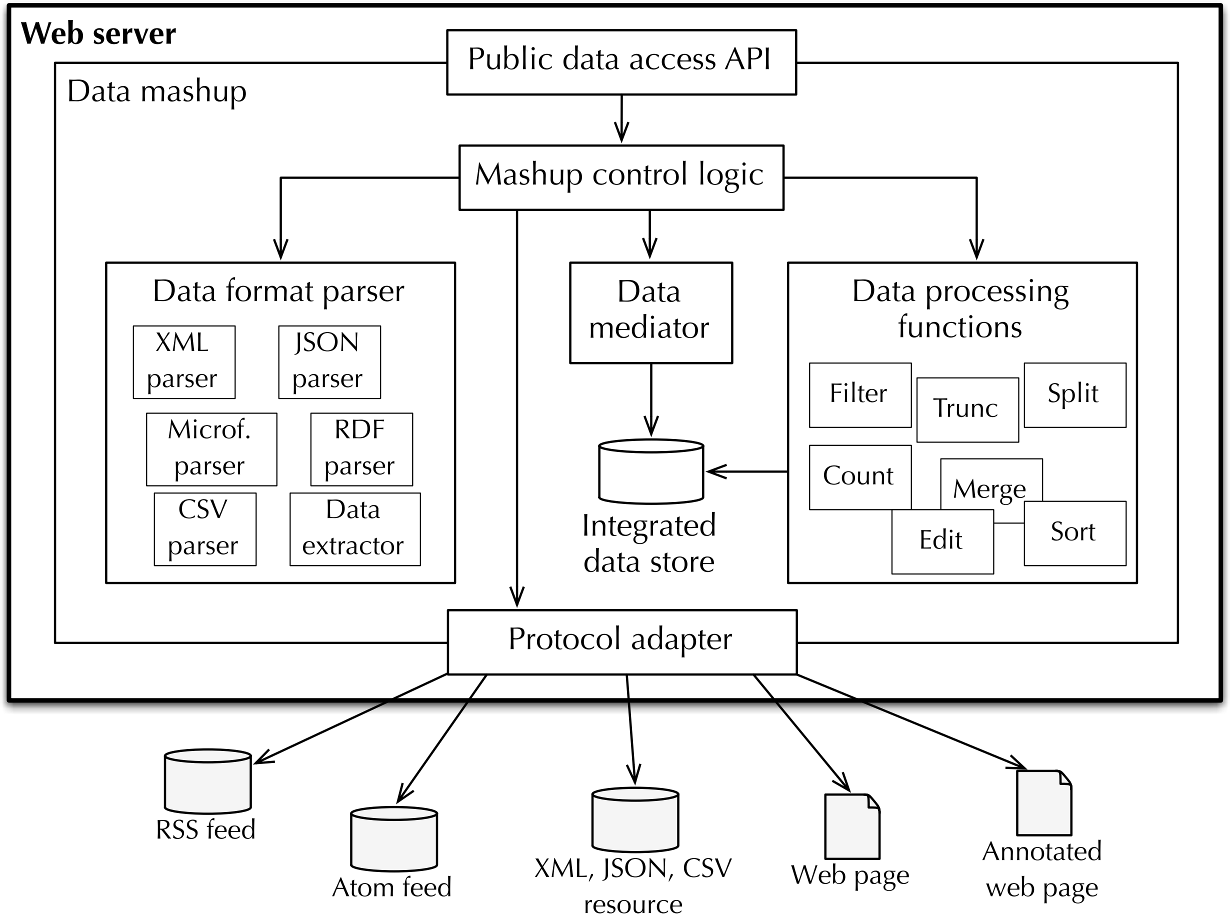
Server-side runtime location

No UI

Stateless instantiation

Point-to-point data mashups

Basic architecture with direct data passing among components and data processing functions. The mashup control logic establishes the necessary direct point-to-point communications



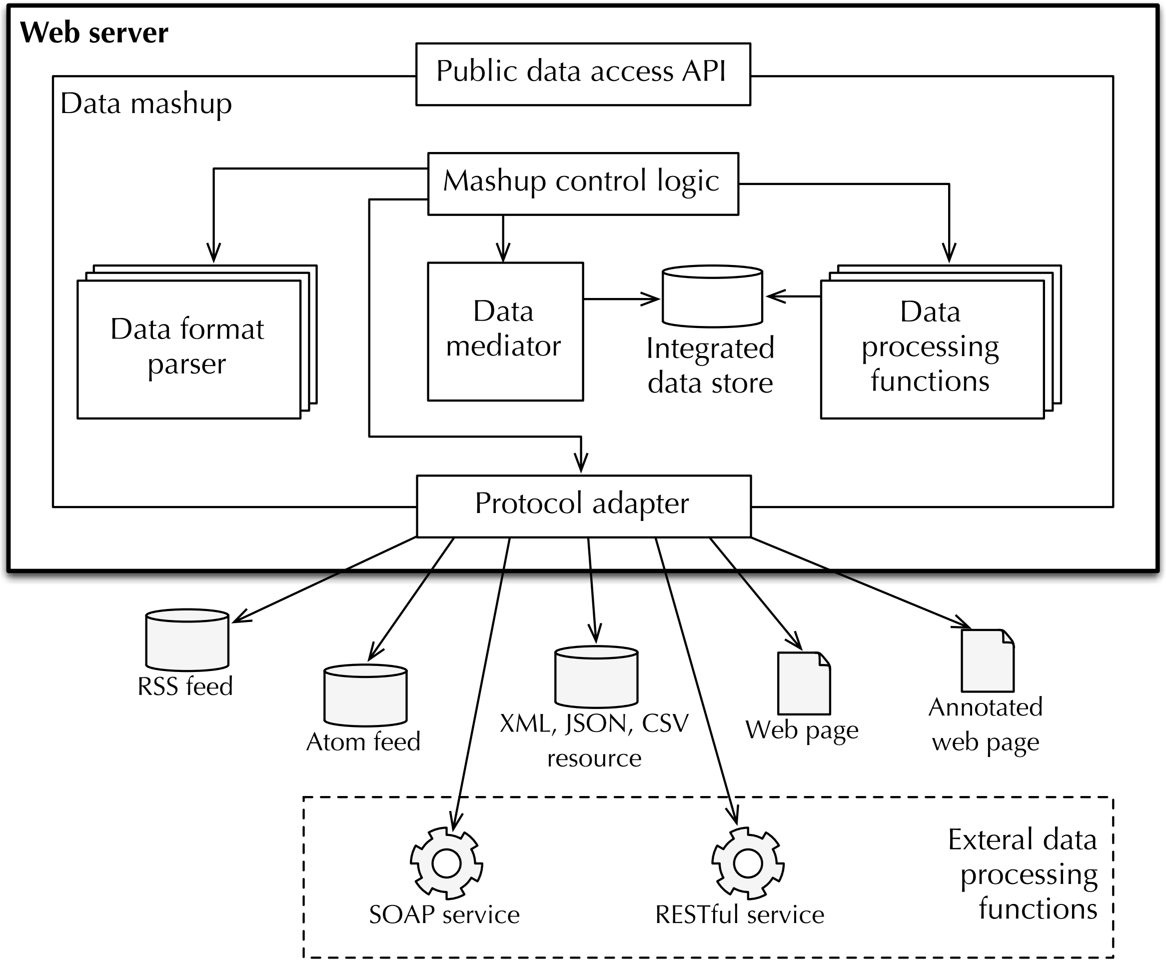
Data mediations between the source data models and the integrated data store

The schema of the integrated data store acts as a global schema

All data processing functions operate on this integrated data store

Centrally-mediated data mashups

Data are transformed and stored in an integrated data store, and all data processing functions operate on this integrated data store only

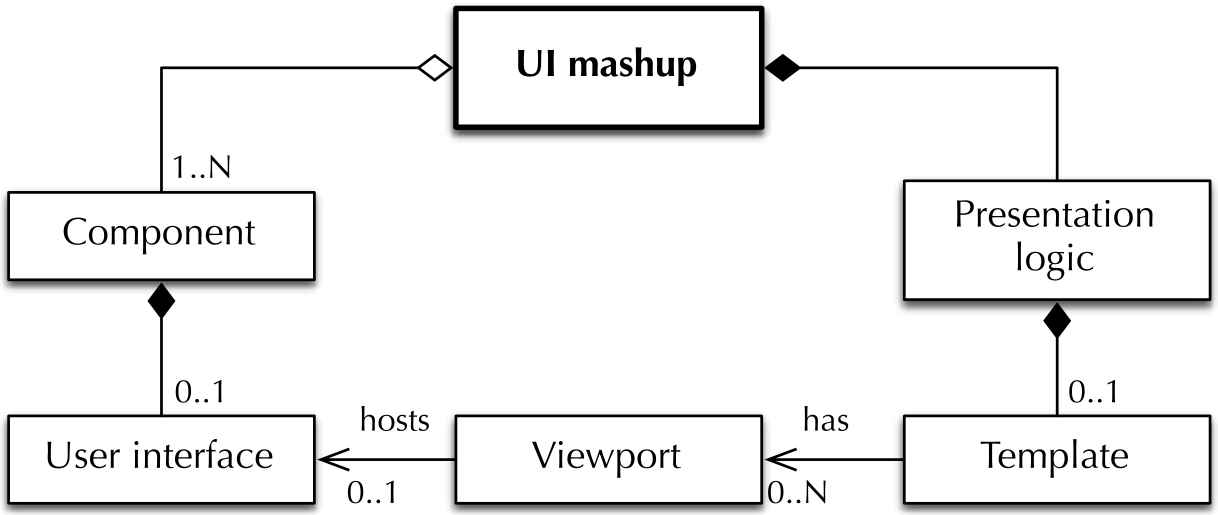


Data mashups with external data processing logic

Besides internal data processing functions, web services or similar are exploited to reuse third-party data processing capabilities and power

User Interface Mashups

- Component integration at the presentation layer **(UI integration)**
  - Reusing and possibly **synchronizing the UIs** of the involved components and mediating possible data mismatches
- Output: **a Web application** the users can interact with
- Particularly appropriate when components have natively a UI and developing a new UI from scratch is simply too costly
- Mostly **client-side**, generally **short-living**
- Different level of complexity: from sharing of a same page layout to complex synchronization/communication patterns



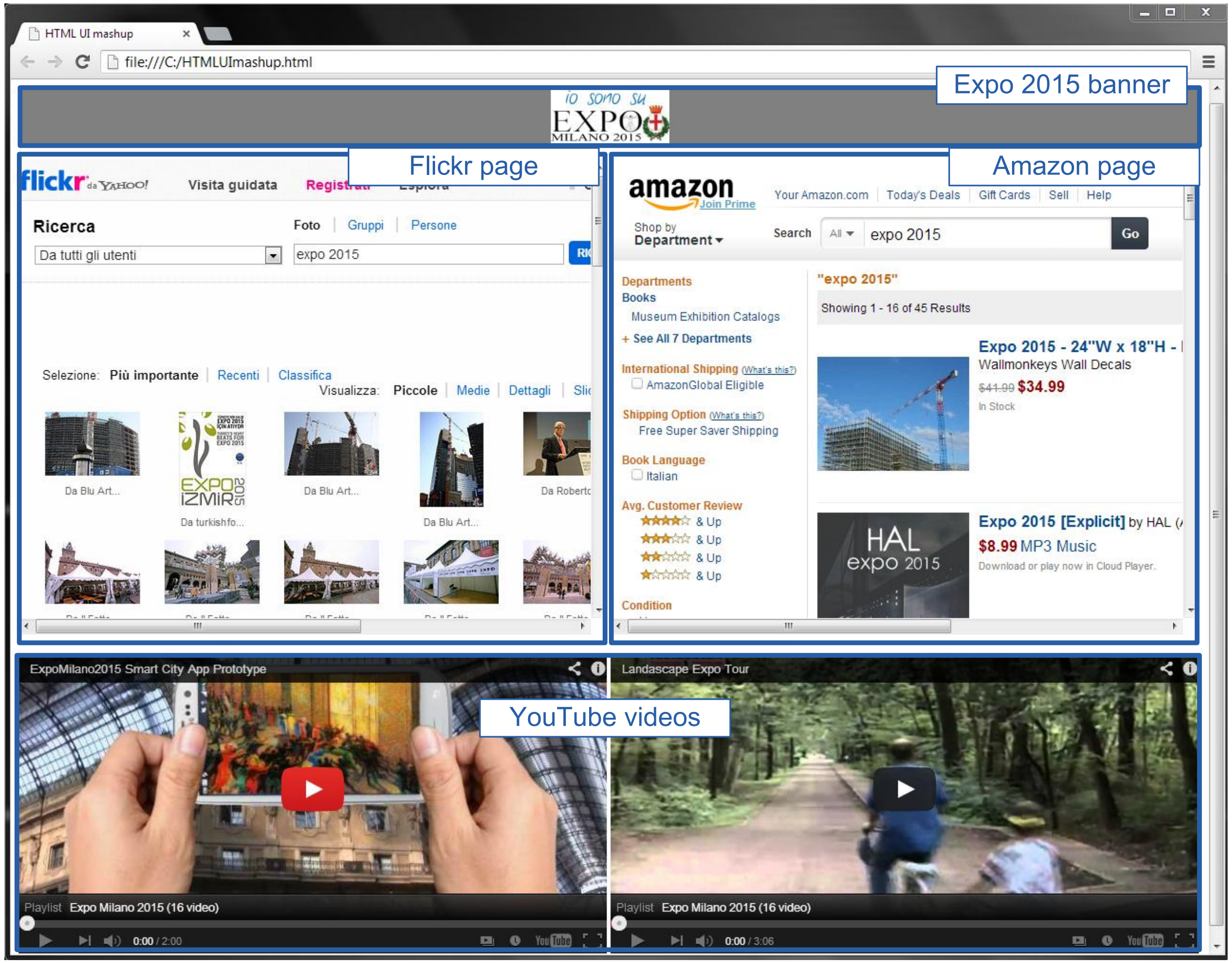
No component invocation, just embedding

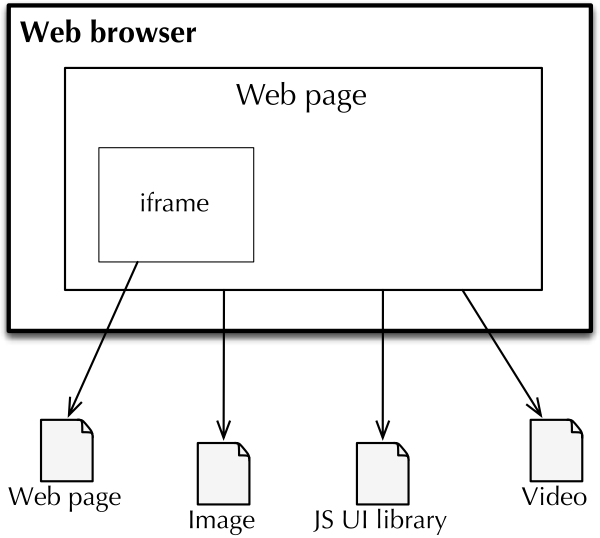
The UI is the only integration dimension

UI mashups without inter-component communication

A specialization of the basic mashup model with new elements:
user interface, templates, viewports

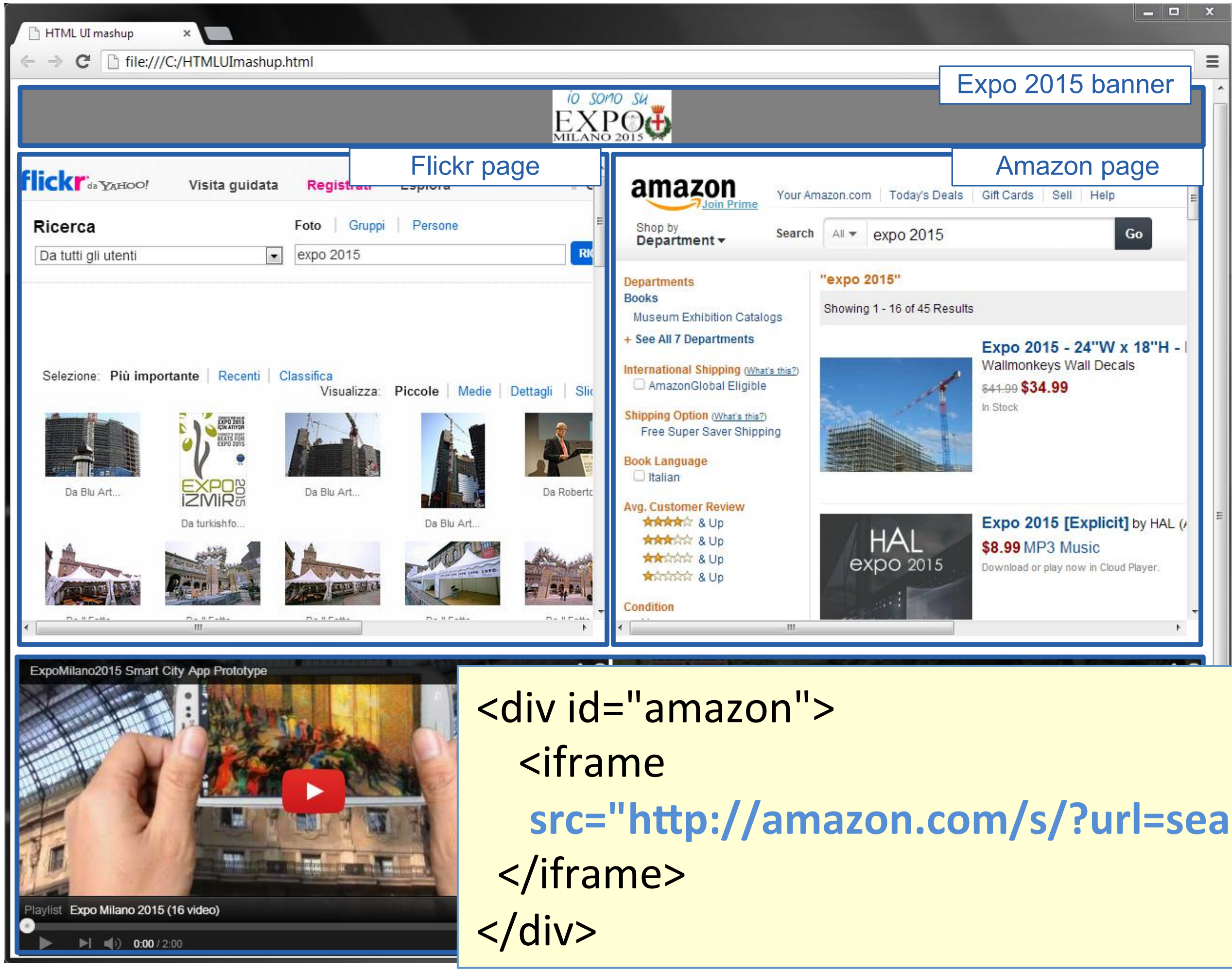
HTML UI Mashup

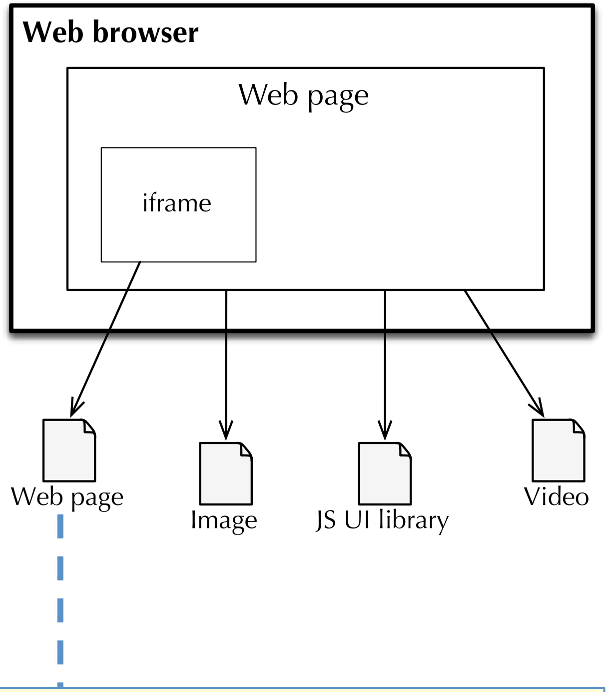




The simplest UI mashup: embedding external resources inside own HTML code

HTML UI Mashup






```
<div id="amazon">
   <iframe
    src="http://amazon.com/s/?url=search-keywords=expo 2015
  </iframe>
</div>
```

The simplest UI mashup: embedding external resources inside own HTML code

Wrapped UI Mashup

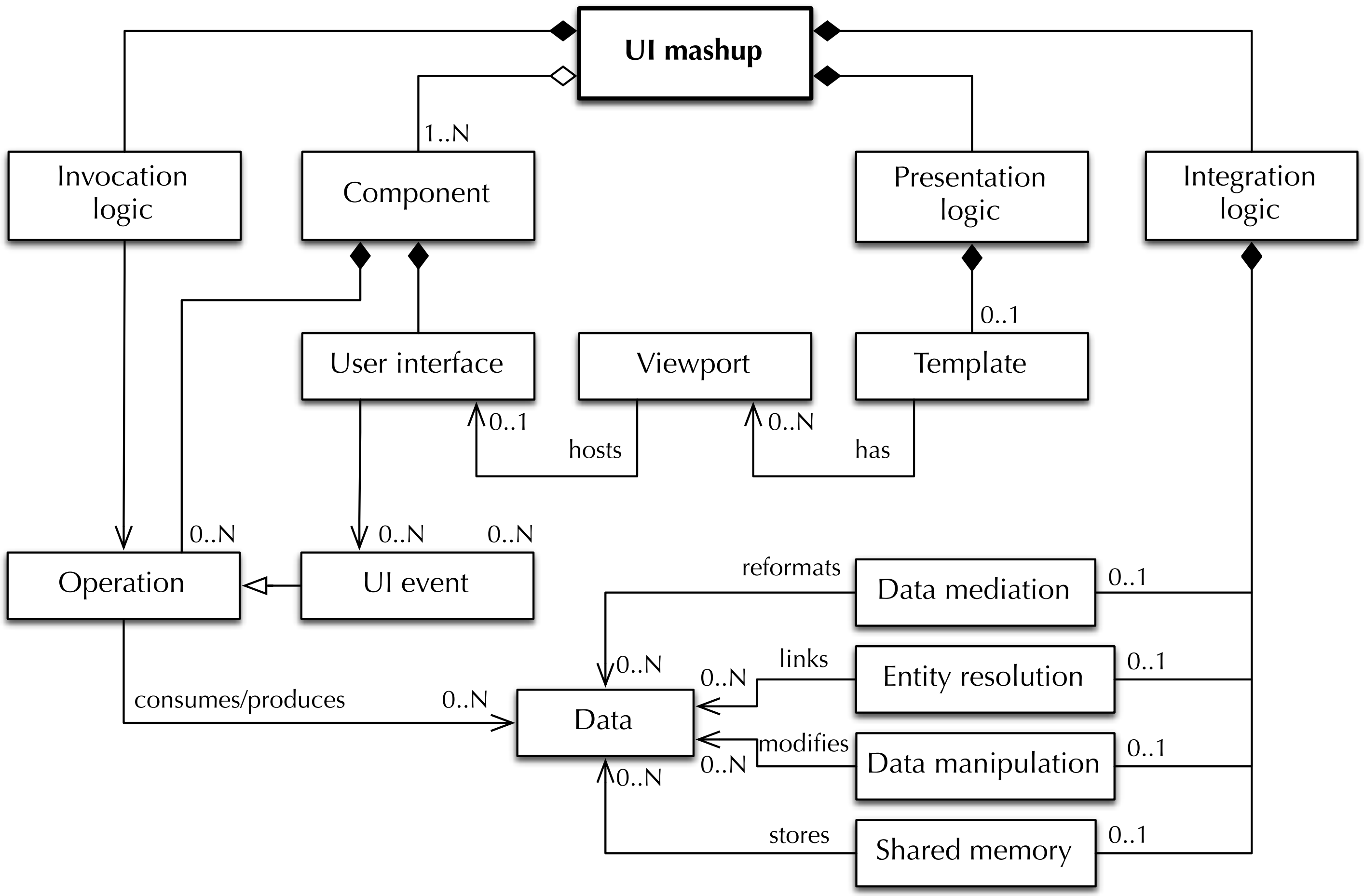
Wrappers invokes the original service, interprets and manipulates the retrieved results

- **Visualizes data according to a suitable HTML UI**
- **Captures UI events**
- **Handles external requests for operations**

Wrapped UI Mashup

Wrappers invokes the original service, interprets and manipulates the retrieved results

- **Visualizes data according to a suitable HTML UI**
- **Captures UI events**
- **Handles external requests for operations**

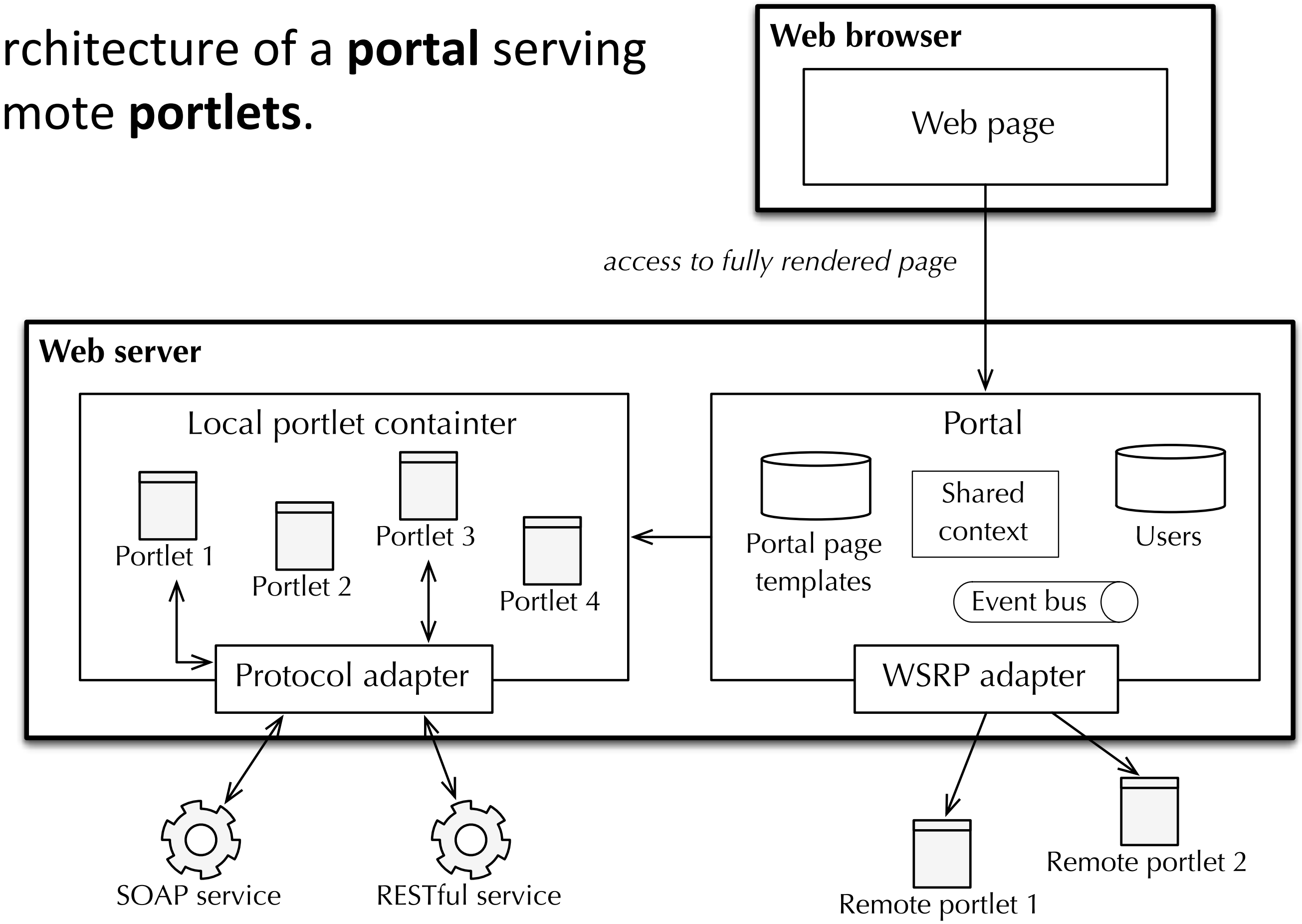


UI mashups with inter-component communication

Three new elements for the synchronization of components:
operations, UI events, shared memory

Container-based UI mashups

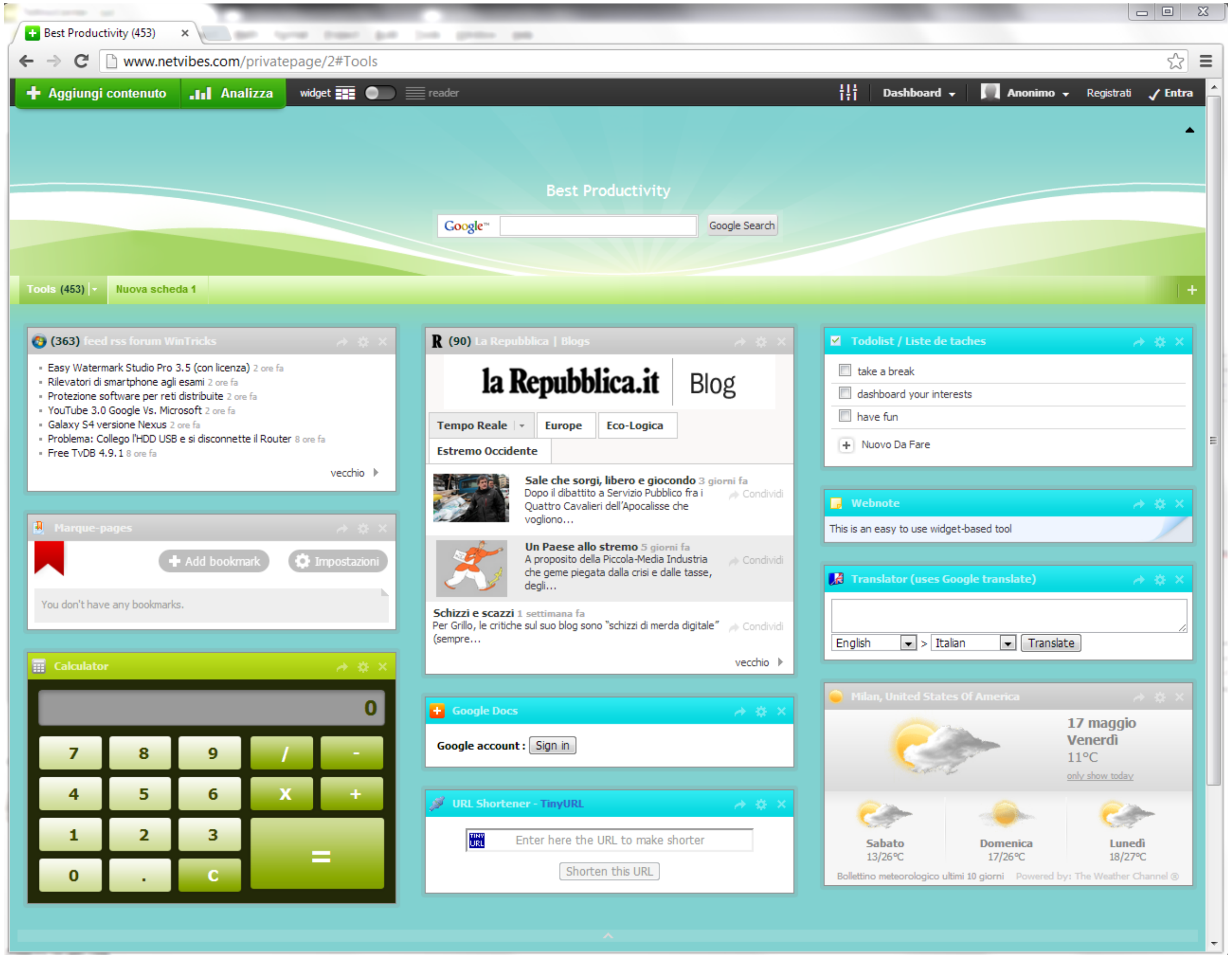
Simplified architecture of a **portal** serving local and remote **portlets**.



Container: runtime environment suporting the deployment and execution of portlets

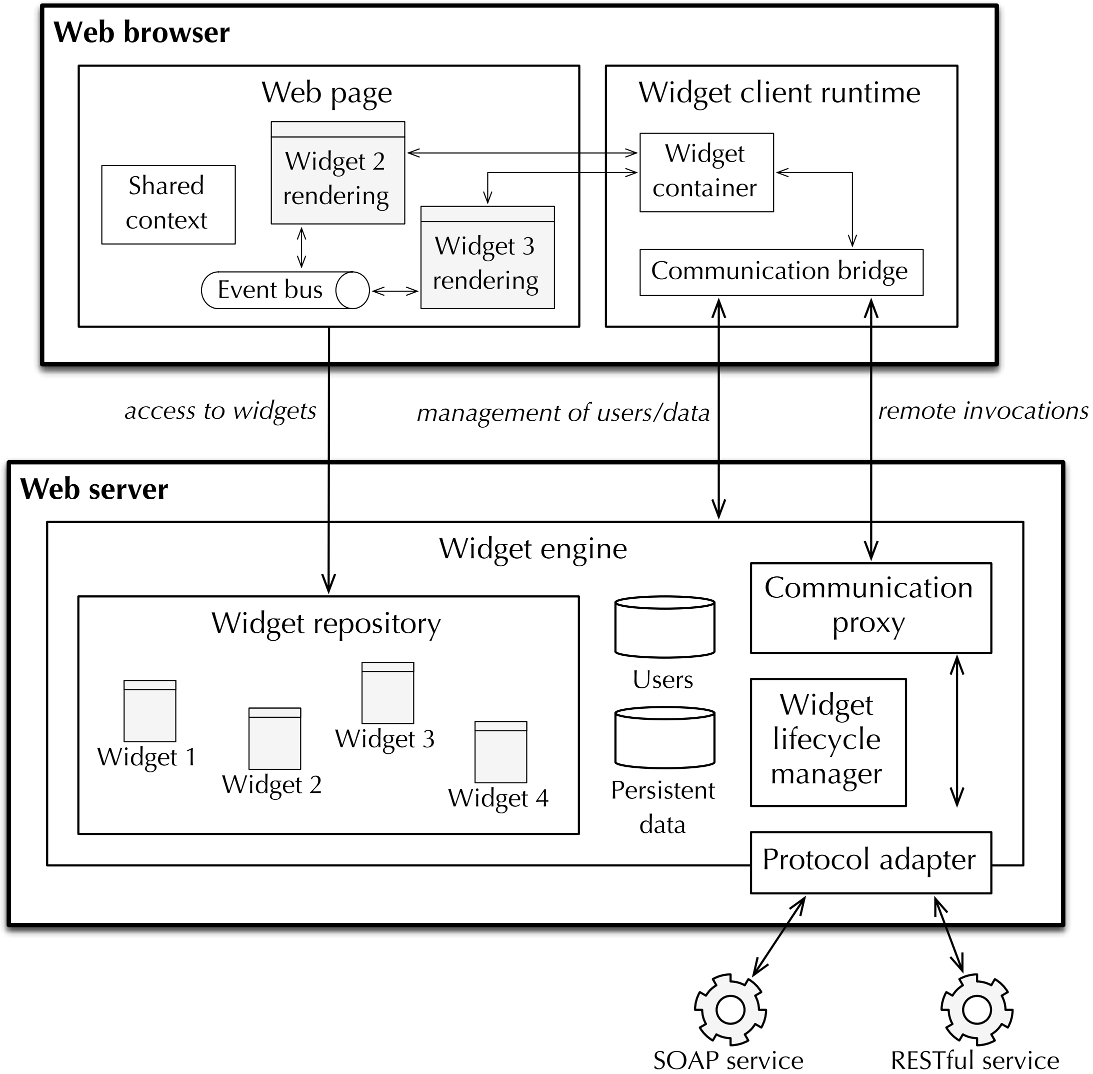
Portal: aggregates the markup of portlets and manages communications with the portlet container in a centrally mediated fashion

Widget-based UI mashups



Different areas of the page correspond to different viewports, each one displaying the content of a widget

Widget-based UI mashups



Logic Mashups

- Integrate components at the application logic layer, by enabling the **composition of functionality** published by logic or data components, and mediating data compatibility issues if necessary

- Output: **a process** that orchestrates components, in turn published as logic component, e.g., a SOAP web service or JavaScript object

- Covered by traditional practices for Service Composition – no further discussed here

Part III

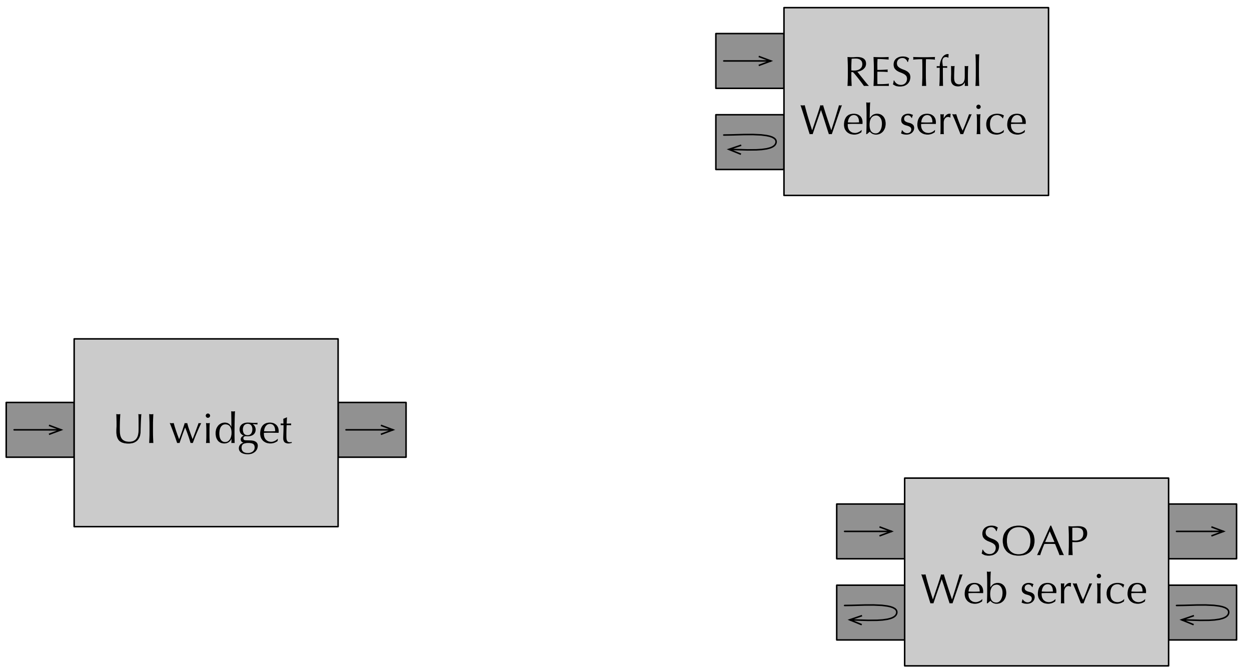
MASHUP TOOLS AND COMPOSITION PARADIGMS

Learning objective = learn how to obtain the key ingredient for a mashup tool, i.e., the **mashup language**

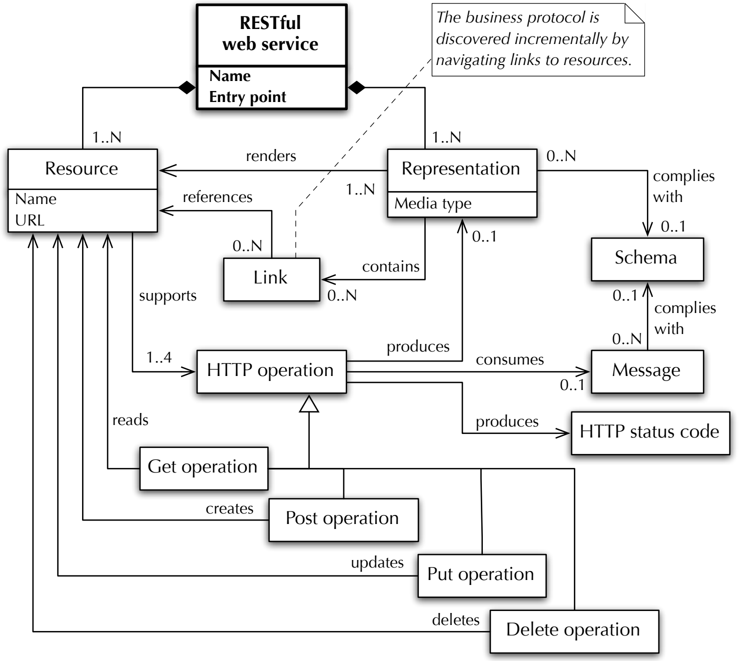
1. Mashup design concerns
2. Component abstractions
3. Graphical mashup languages
4. XML mashup languages
5. Other languages
6. Developing languages
7. Reference architecture for mashup tools

MASHUP CONCERNS

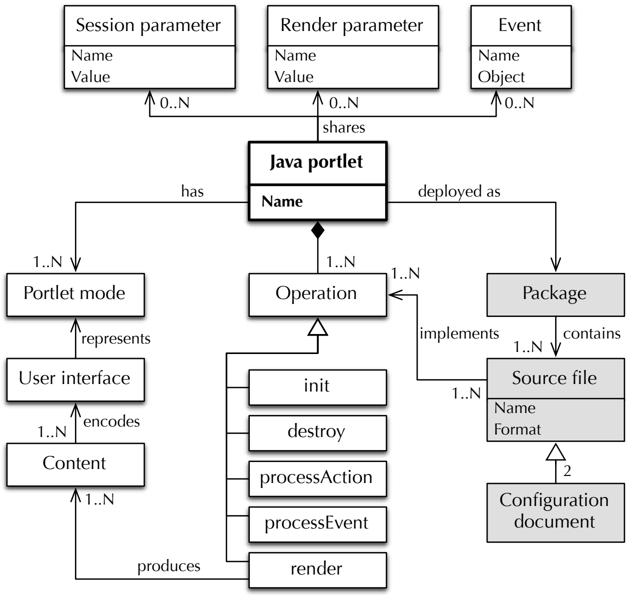
Components and component models



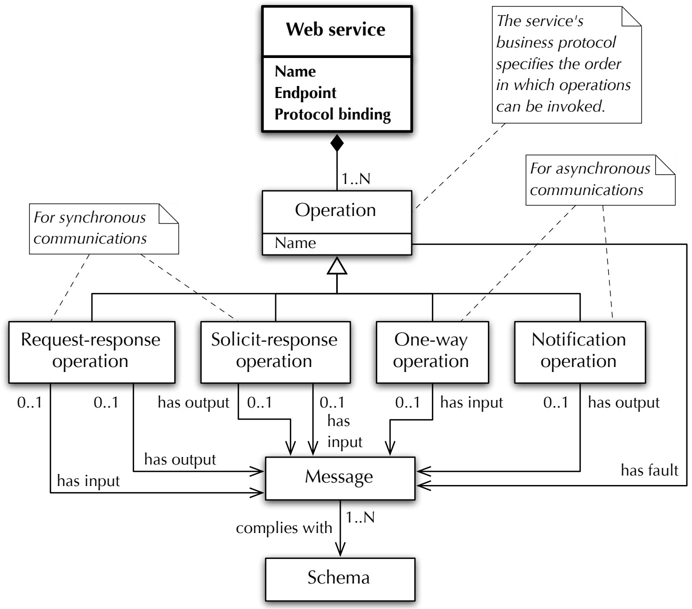
Components and component models



Component model 2

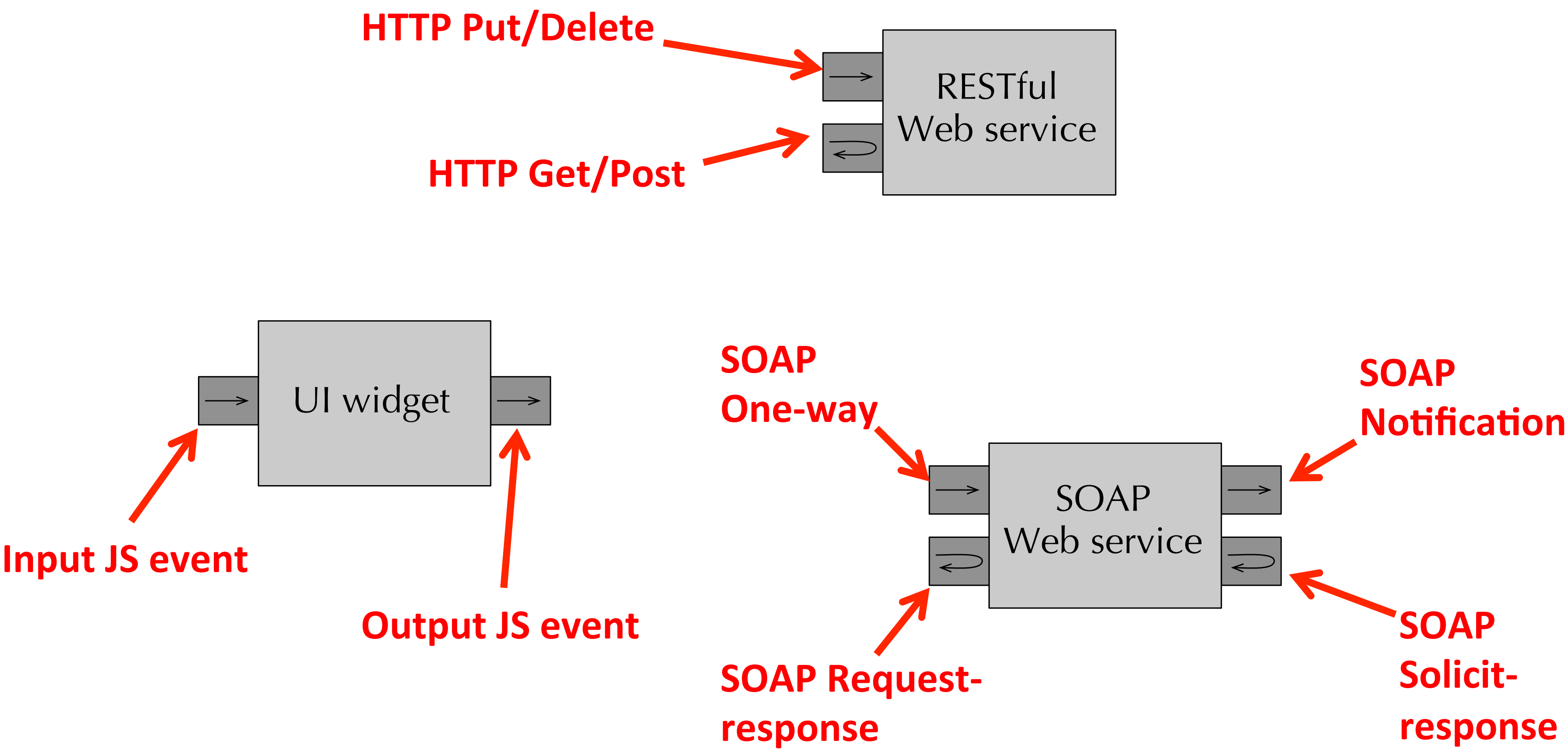


Component model 1

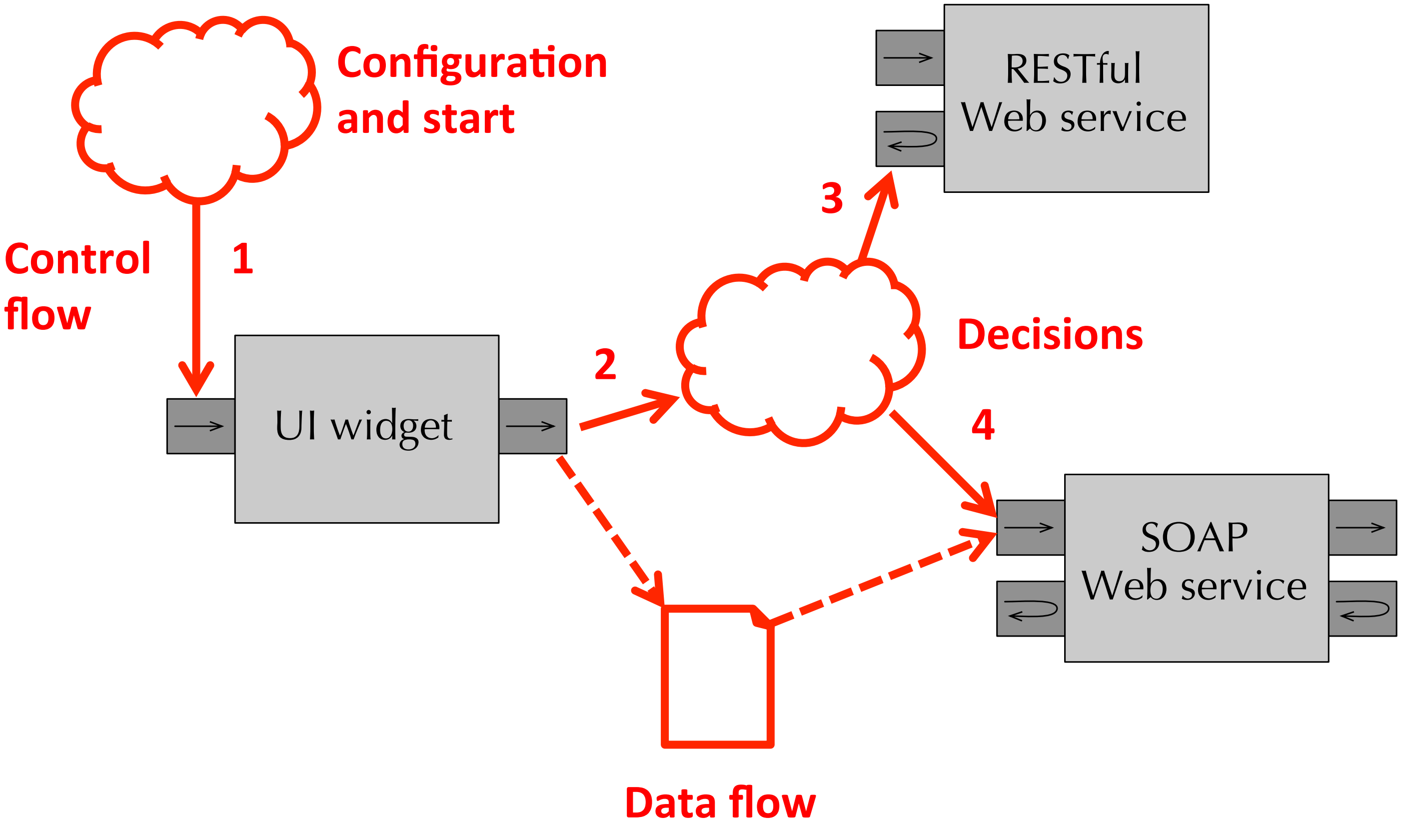


Component model 3

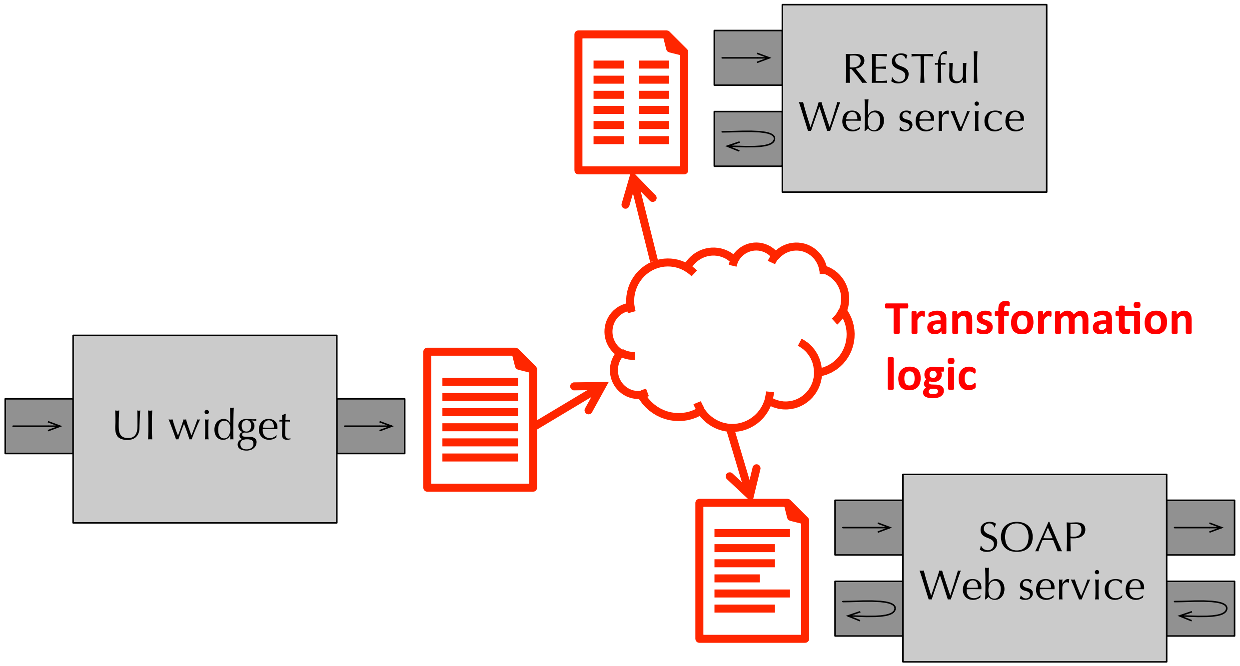
Component access



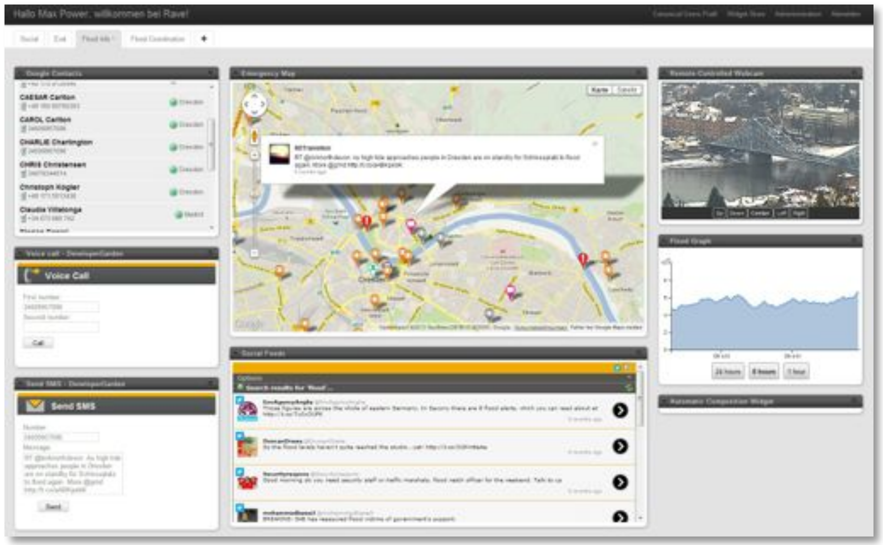
Control flow and data flow



Data transformations



User interface layout



Graphical placement of components

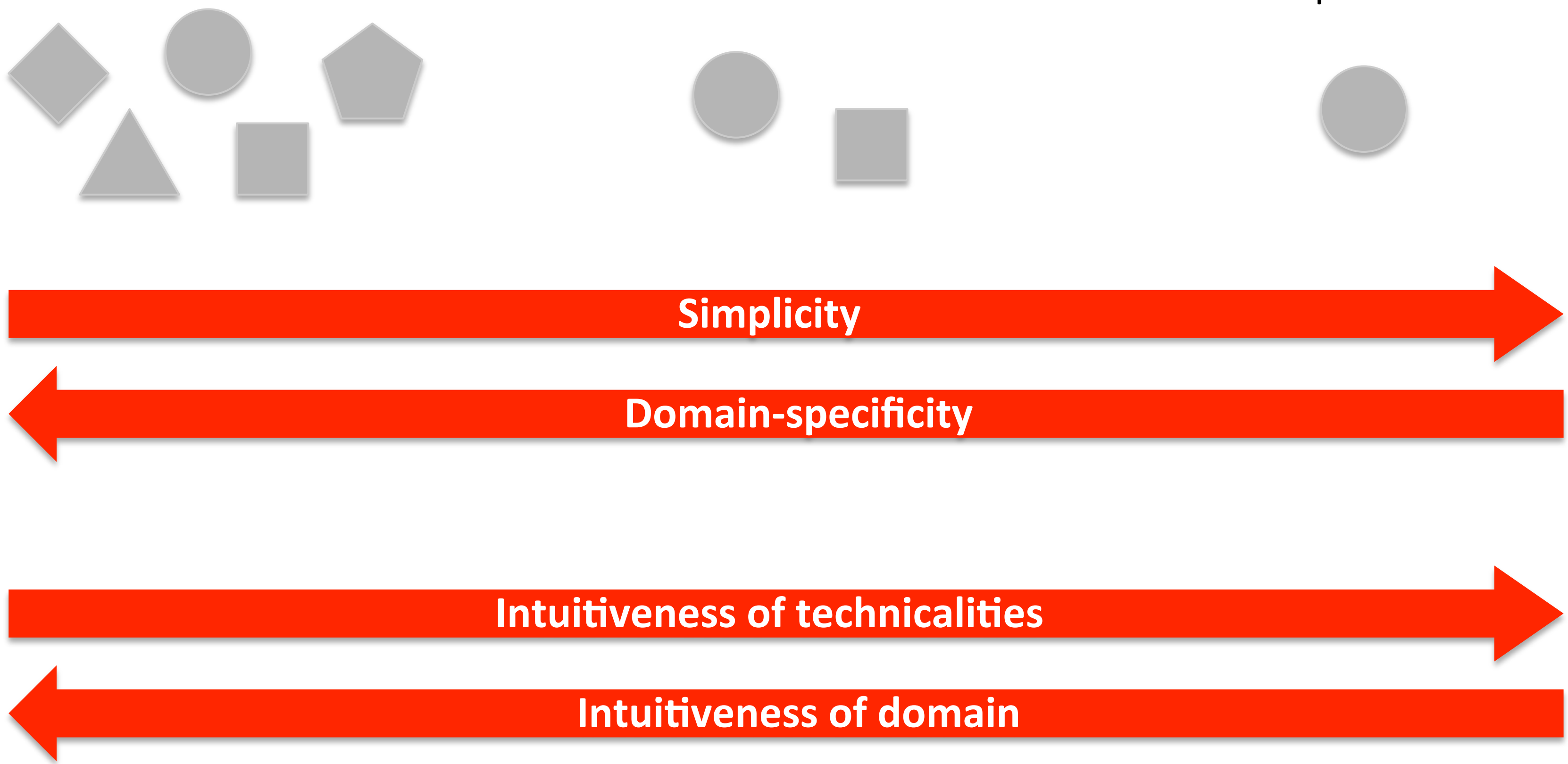
COMPONENT ABSTRACTIONS

Modeling constructs may represent….

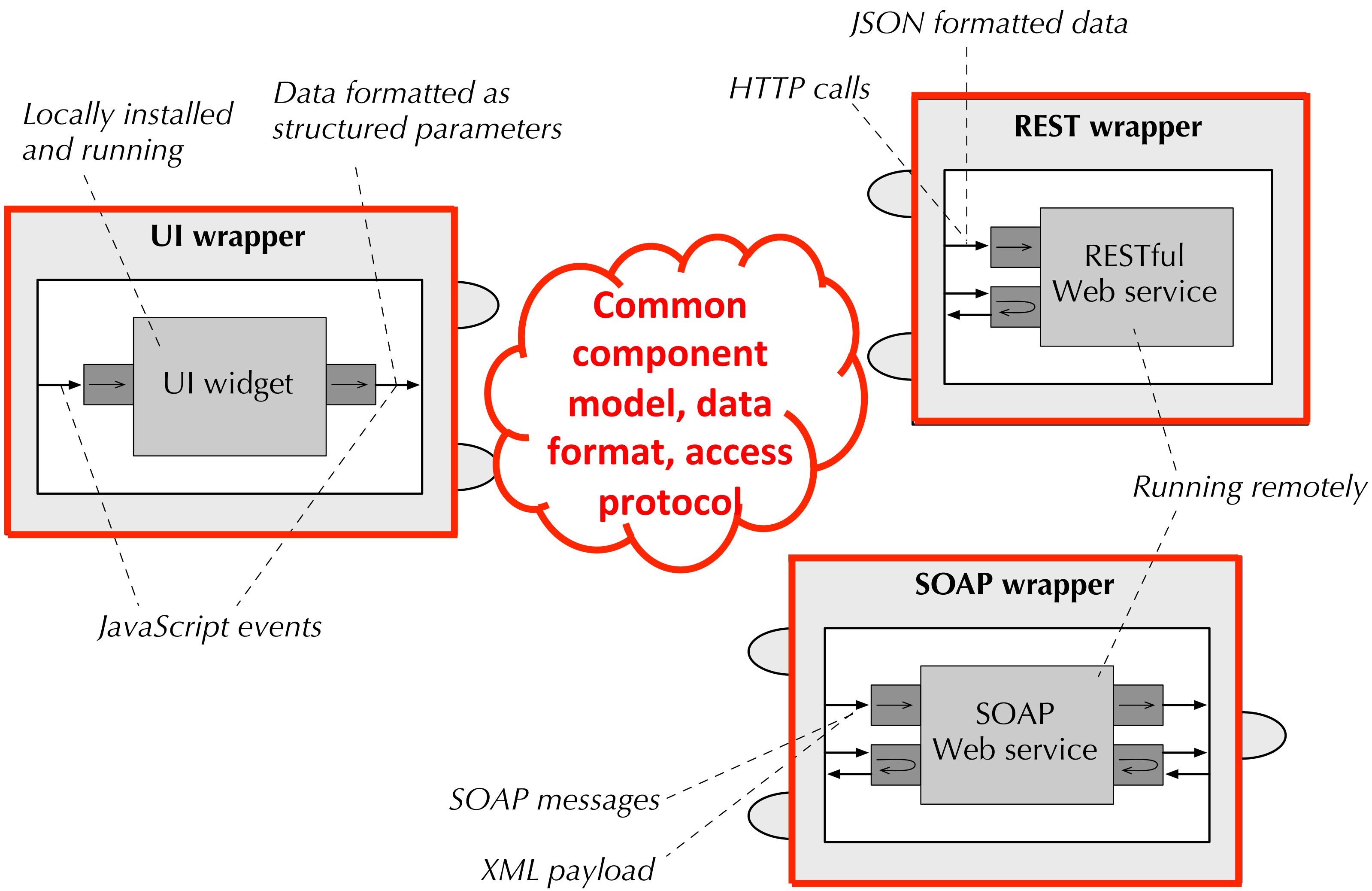
Component instances
= each component has an own construct

Component types
= similar components have the same construct

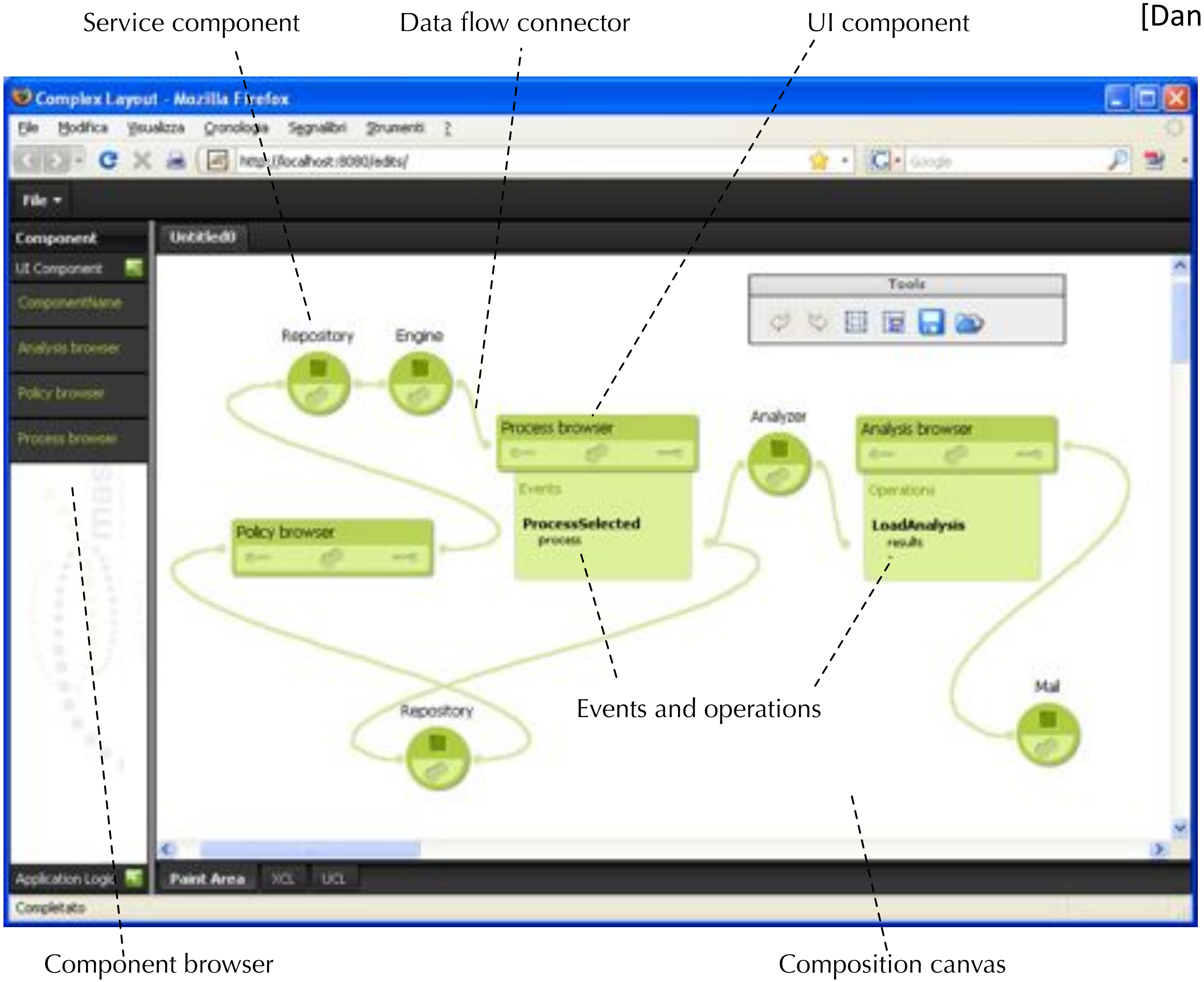
Unified component model
= one construct for all components



Abstracting = wrapping

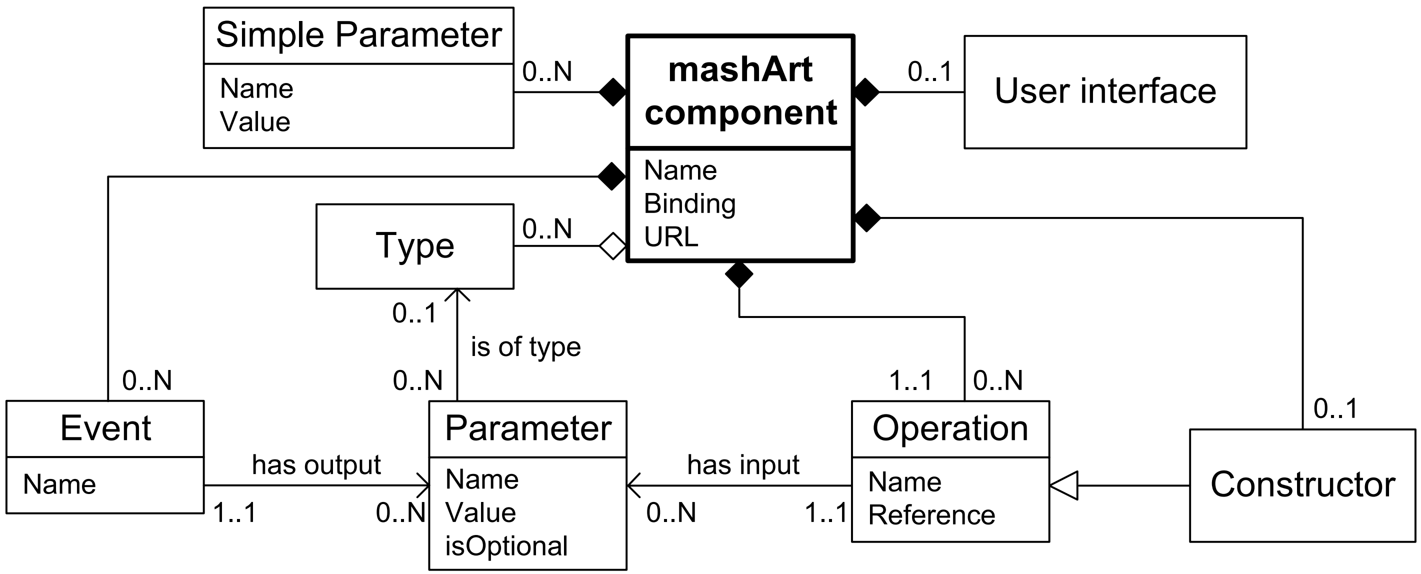


[Daniel2009]



Components in mashArt: apparently two component models

But internally mashArt uses a **unified** component model

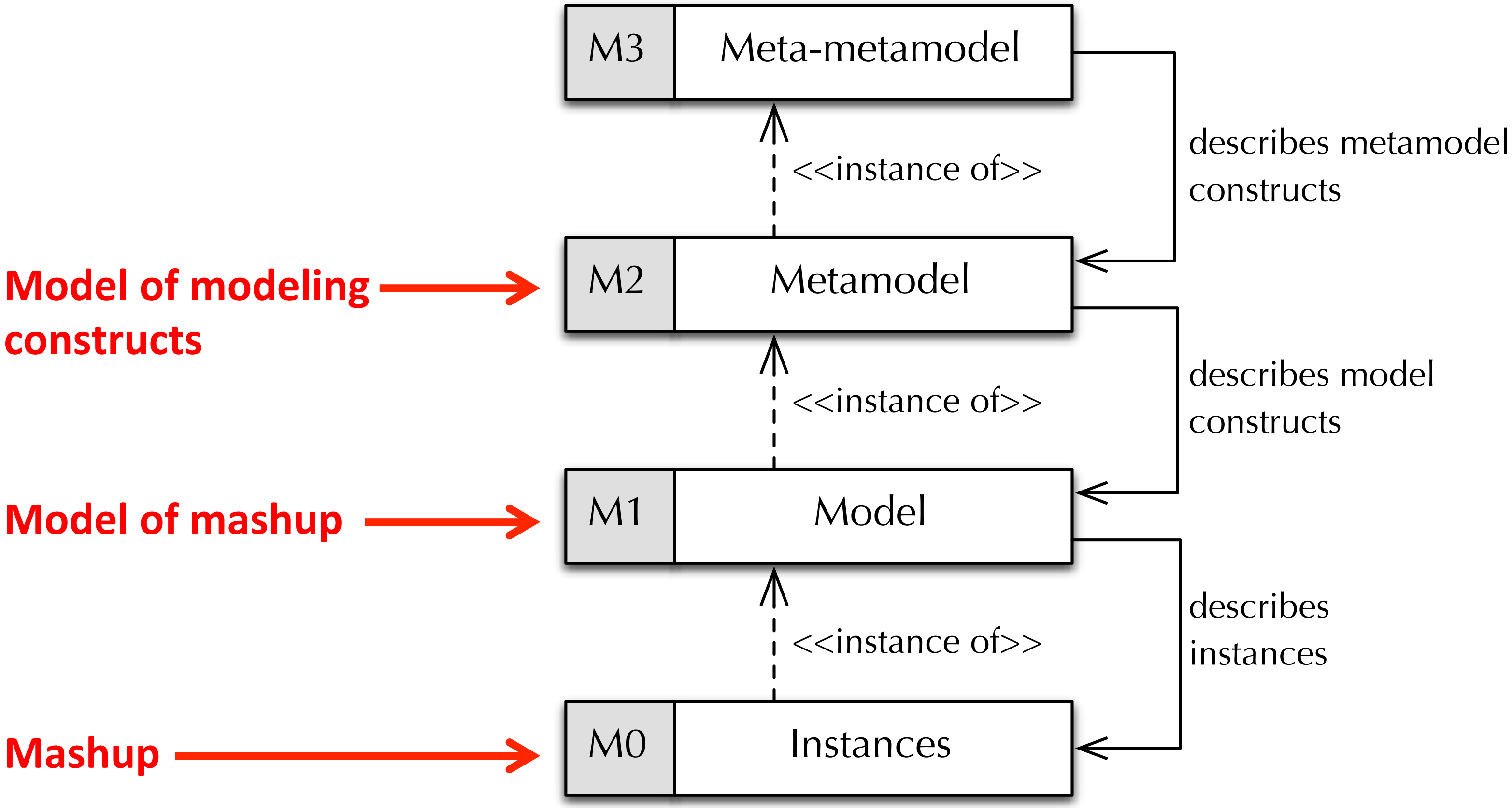


The model accommodates:

- SOAP/RESTful web services
- RSS/Atom feeds
- UI components

GRAPHICAL MASHUP LANGUAGES

Model-driven mashup development



OMG's Meta Object Facility (MOF)

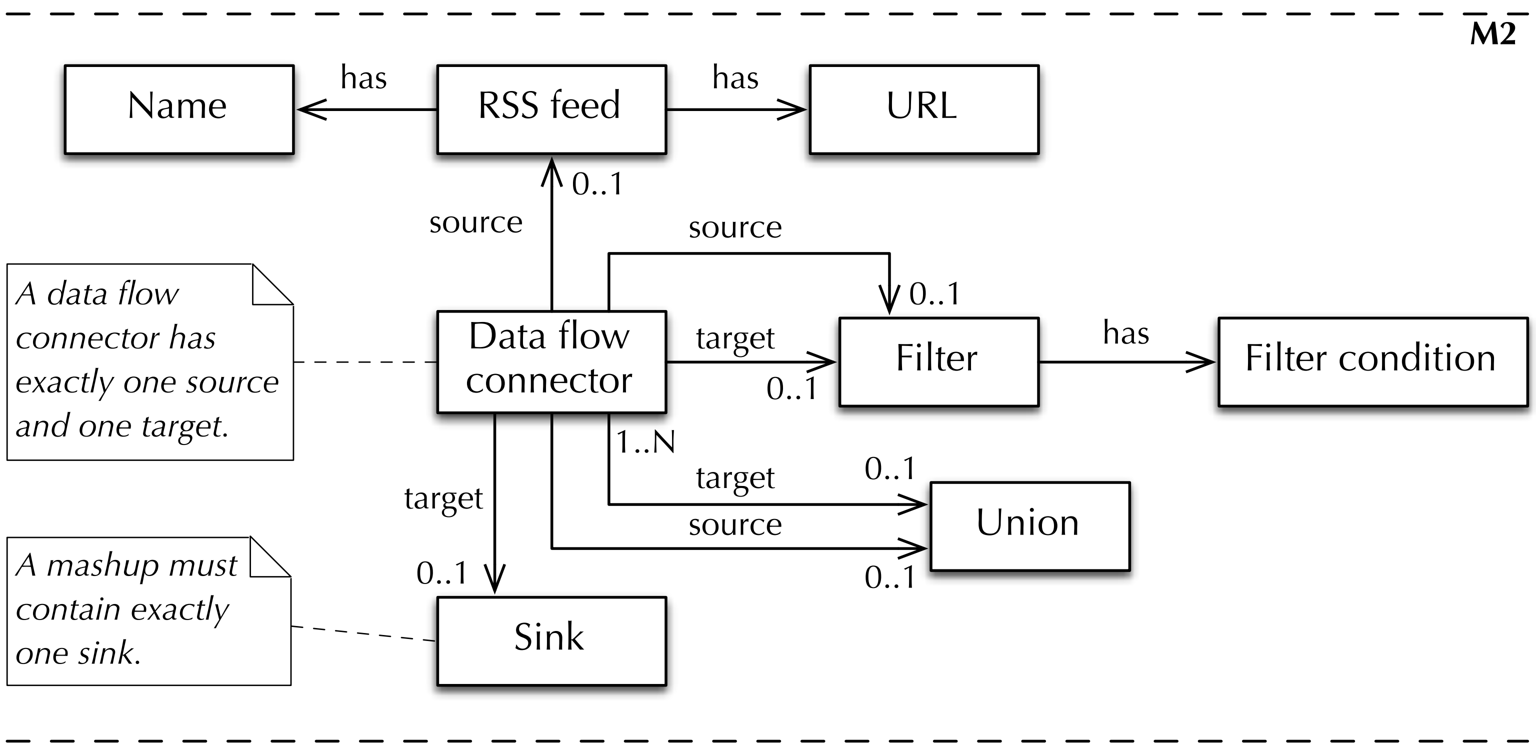
Let's proceed by **example**...

Let's design a simple **data mashup** language...

Requirements

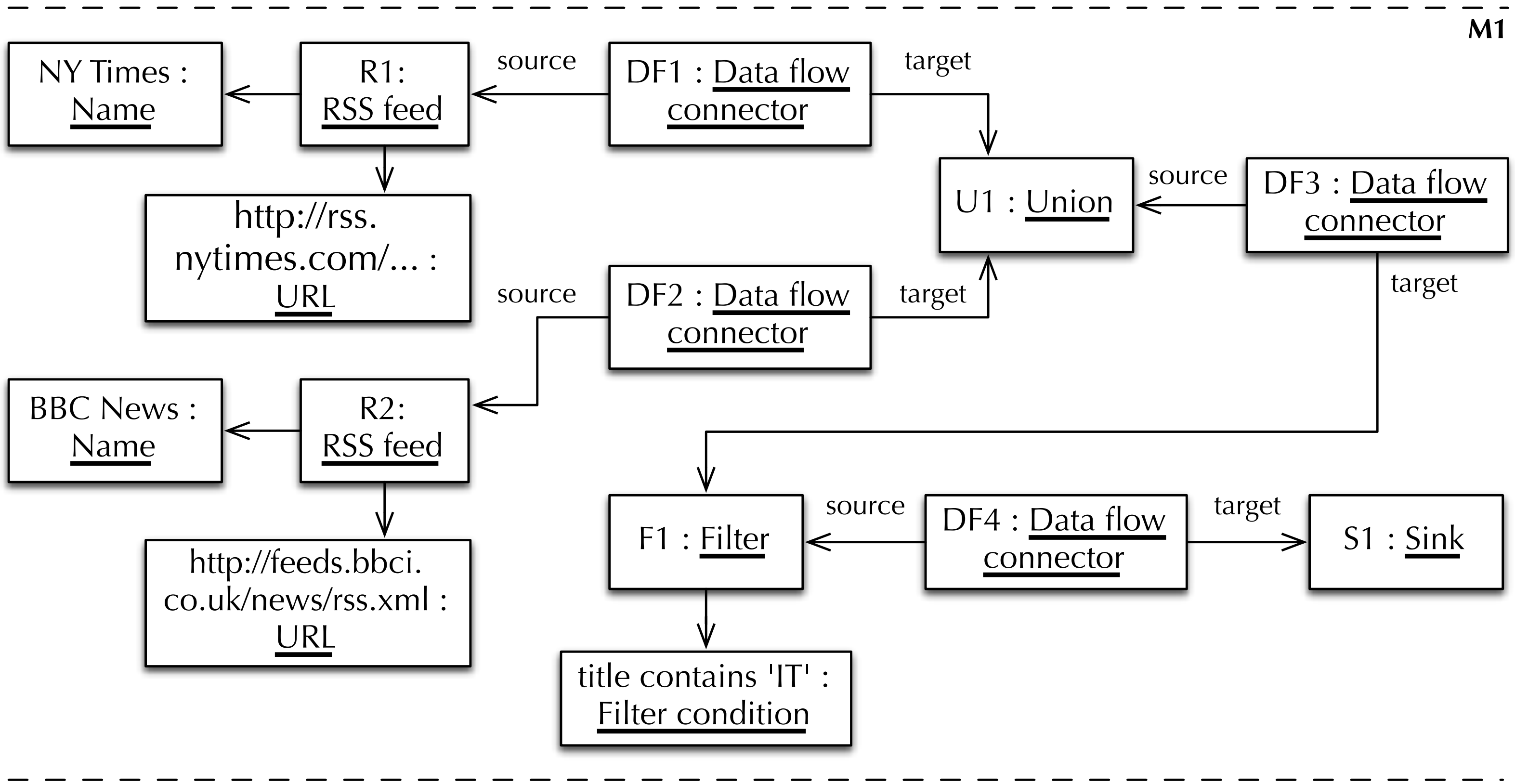
1. Integrate **RSS feeds**
2. A **Union** operator merges feeds
3. A **Filter** operator filters items by conditions
4. A **Sink** component ends processing
5. **Data flow** connectors propagate data

Metamodel

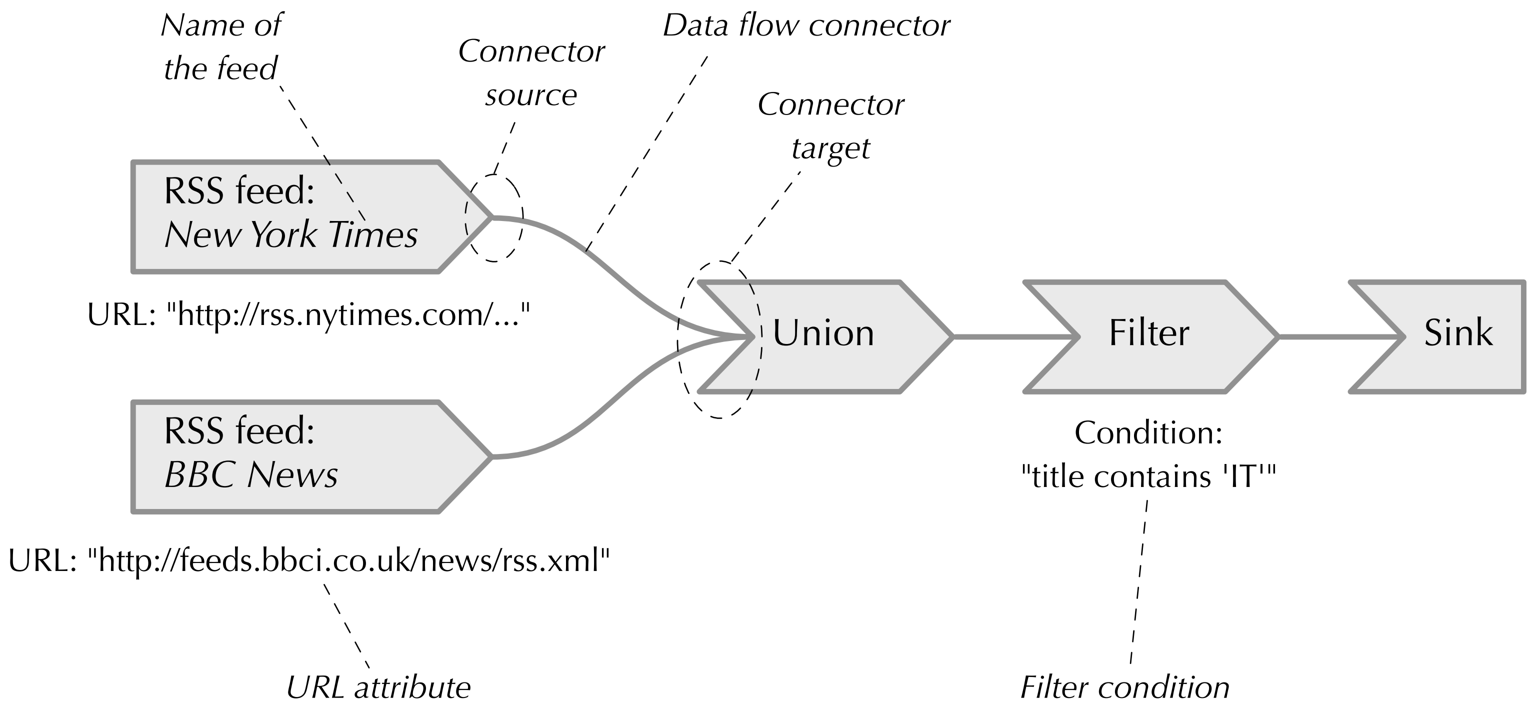


M2
Name
has
RSS feed
has
URL
0..1
source
source
0..1
A data flow connector has exactly one source and one target.
Data flow connector
target
0..1
Filter
has
Filter condition
1..N
target
0..1
Union
source
0..1
target
0..1
A mashup must contain exactly one sink.
Sink

Model (abstract syntax) = instance of metamodel

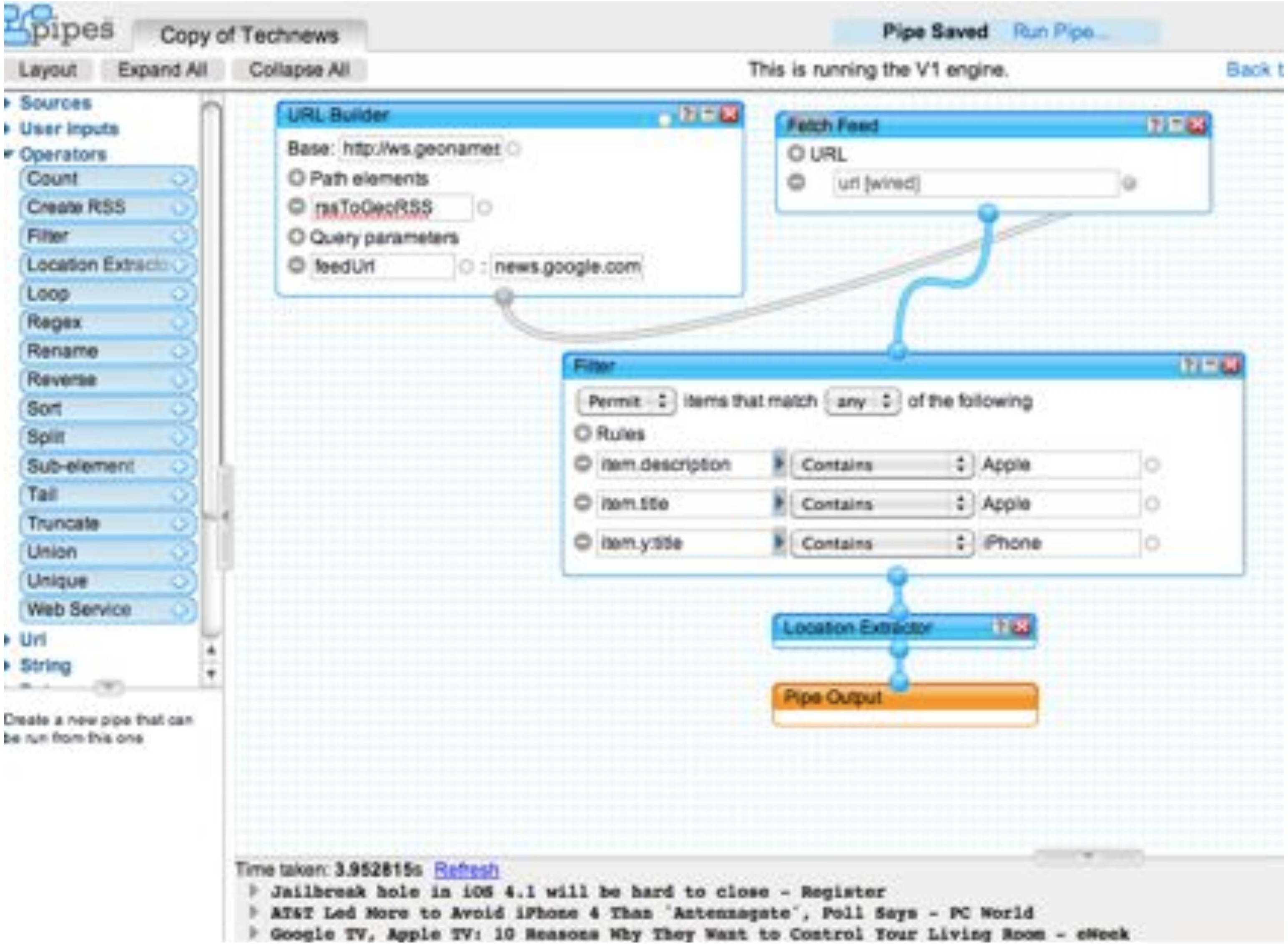


Model (**concrete syntax**) >> Human readable

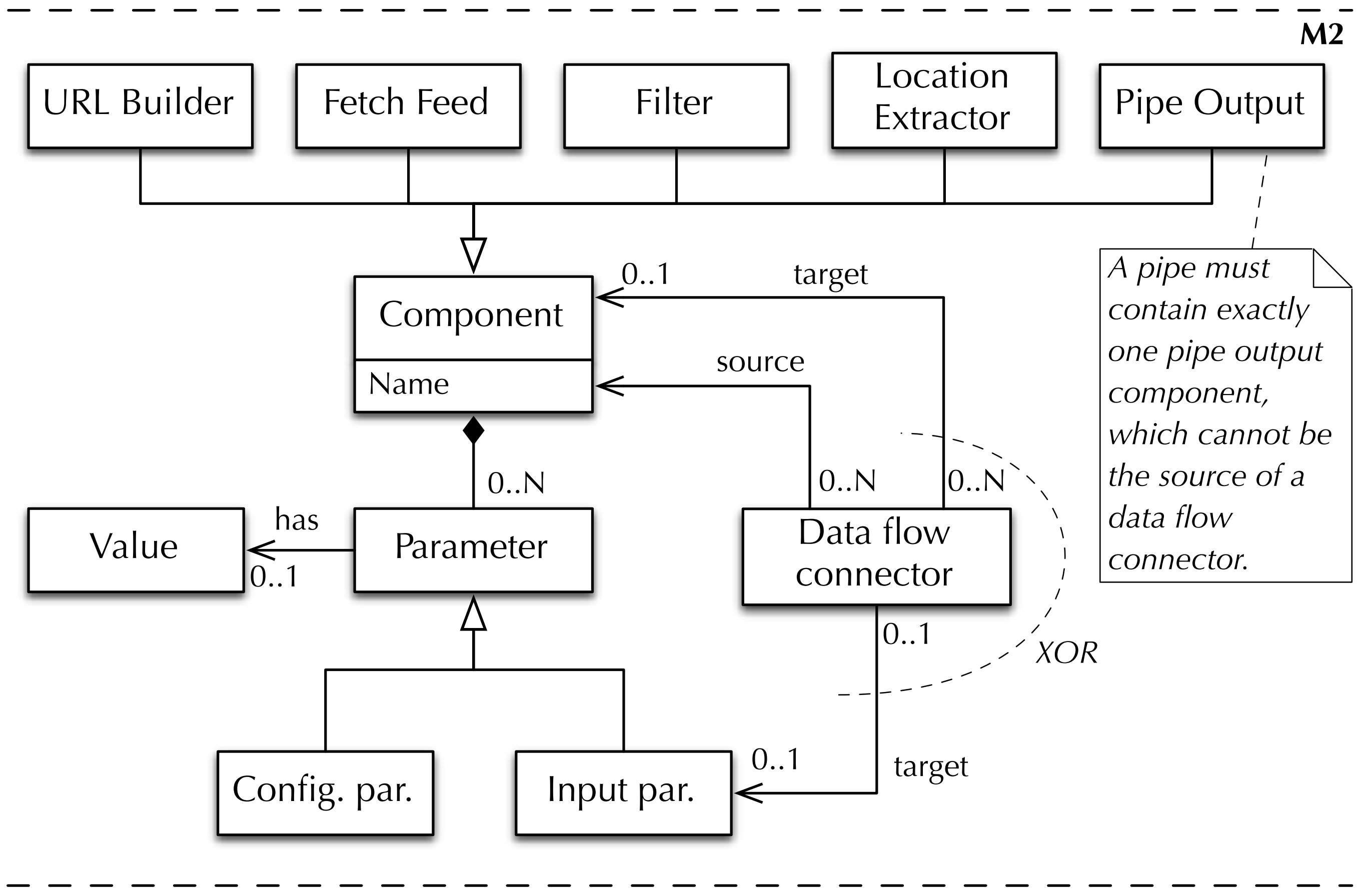


Same model as before!

Let’s **reverse-engineer** Yahoo! Pipes



And here a possible metamodel



XML MASHUP LANGUAGES

EMML, the Enterprise Mashup Markup Language

Data mashups

[http://mdc.jackbe.com/prestodocs/v3.7/emml/mashup-library-intro.html]

```
<?xml version="1.0"?>
<mashup name="Mashup News"
    xmlns="www.openemml.org/2009-04-15/EMMLSchema"
    xsi:schemaLocation="www.openemml.org/2009-04-15/EMMLSchema
      ../schema/EMMLSpec.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <variables>
    <variable name="news" type="document"/>
  </variable>
  <output name="result" type="document"/>
  <directinvoke endpoint="http://rss.nytimes.com/services/xml/
      rss/nyt/Technology.xml" method="GET"
      outputvariable="news"/>
  <filter inputvariable="news"
      filterexpr="matches($news/rss/channel/item/description,
         'Mashup')"
      outputvariable="result"/>
</mashup>
```

Variables

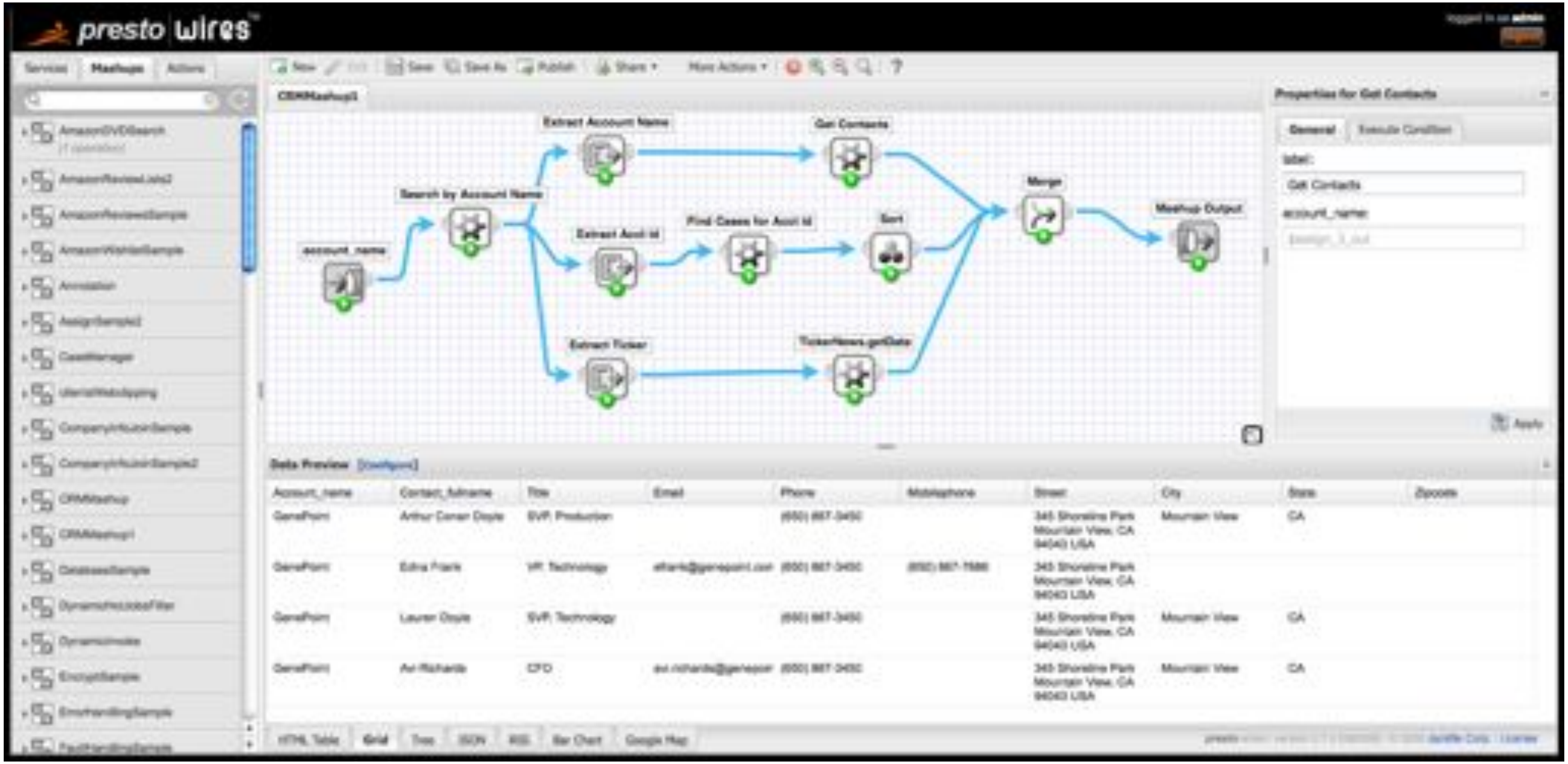
Mashup output declaration

Feed invocation

Filter over items

Connection with output channel

Software AG Presto implements EMML



JackBe has recently been acquired by Software AG, and many of the former online resources are no longer accessible

[http://omdl.org]

OMDL: the Open Mashup Description Language

UI mashups

```
<?xml version="1.0"?>
<workspace xmlns="http://omdl.org/">
  <goal>Illustrate a simple OMDL example</goal>
  <identifier>http://repo.omdl.org/mashups/...</identifier>
  <title>An OMDL example</title>
  <description>...</description>
  <creator>Florian Daniel</creator>
  <date>2013-10-08T14:23+37:00</date>

  <app id="http://repo.omdl.org/mashups/alice/CallFromMap/1">
    <type>MAP</type>
    <link rel="source" href="http://repo.omdl.org/apps/map/
        MyFancyMap" type="application/widget"/>
    <position>TOPLEFT</position>
  </app>
  <app id="http://repo.omdl.org/mashups/alice/CallFromMap/2">
    ...
  </app>

  <capabilities>
     <gps mandatory="true" accuracy="100"/>
  </capabilities>

  <layout>GRID</layout>
  <stylesheet>
      http://repo.omdl.org/mashups/alice/3/_data/special.css
  </stylesheet>
```

Usual metadata

App/widget declarations

Type

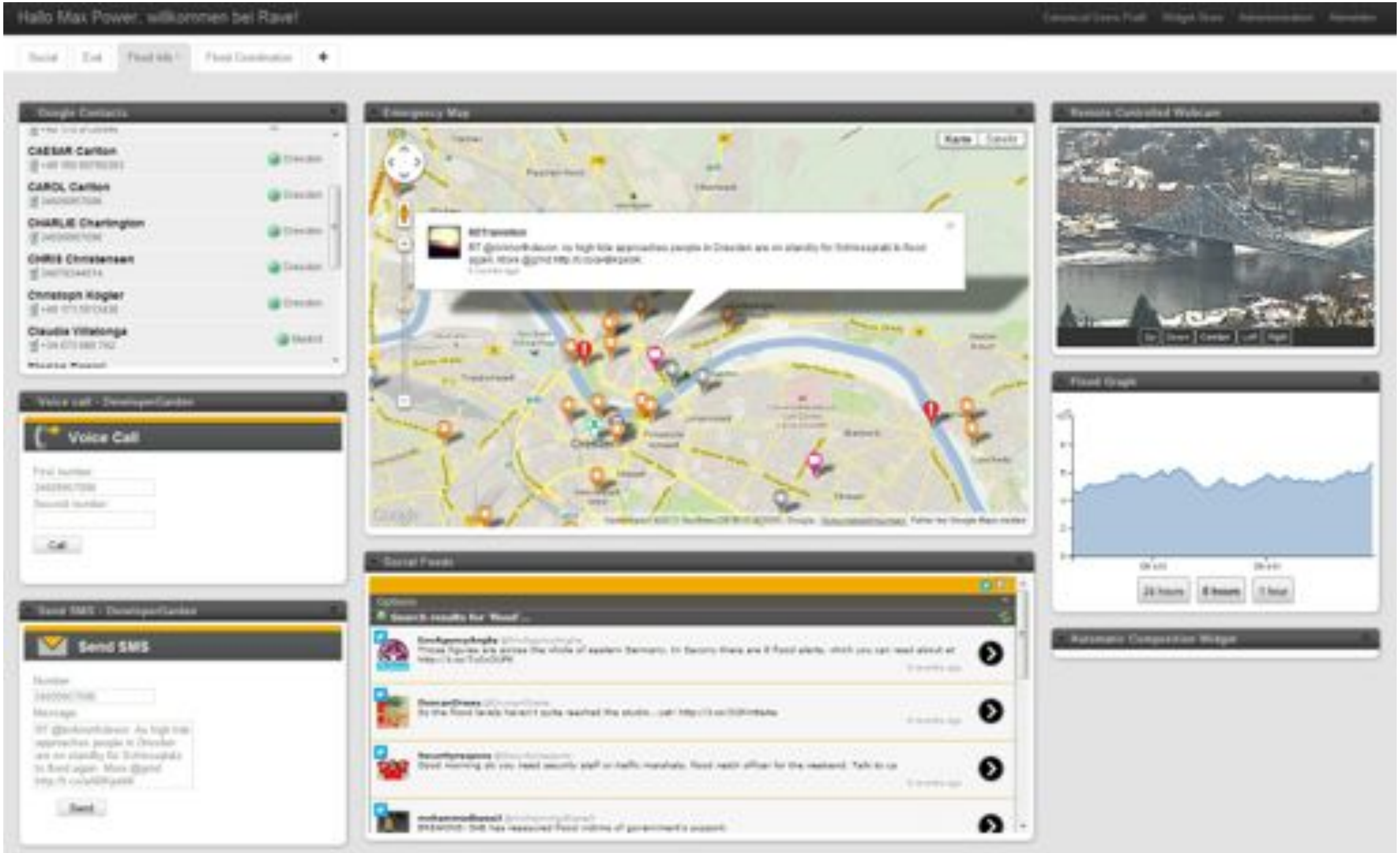
URL

Position

Capabilities required by mashup

Layout template

The **OMELETTE Apache Rave** environment has OMDL-compliance workspaces



DEVELOPING MASHUP LANGUAGES

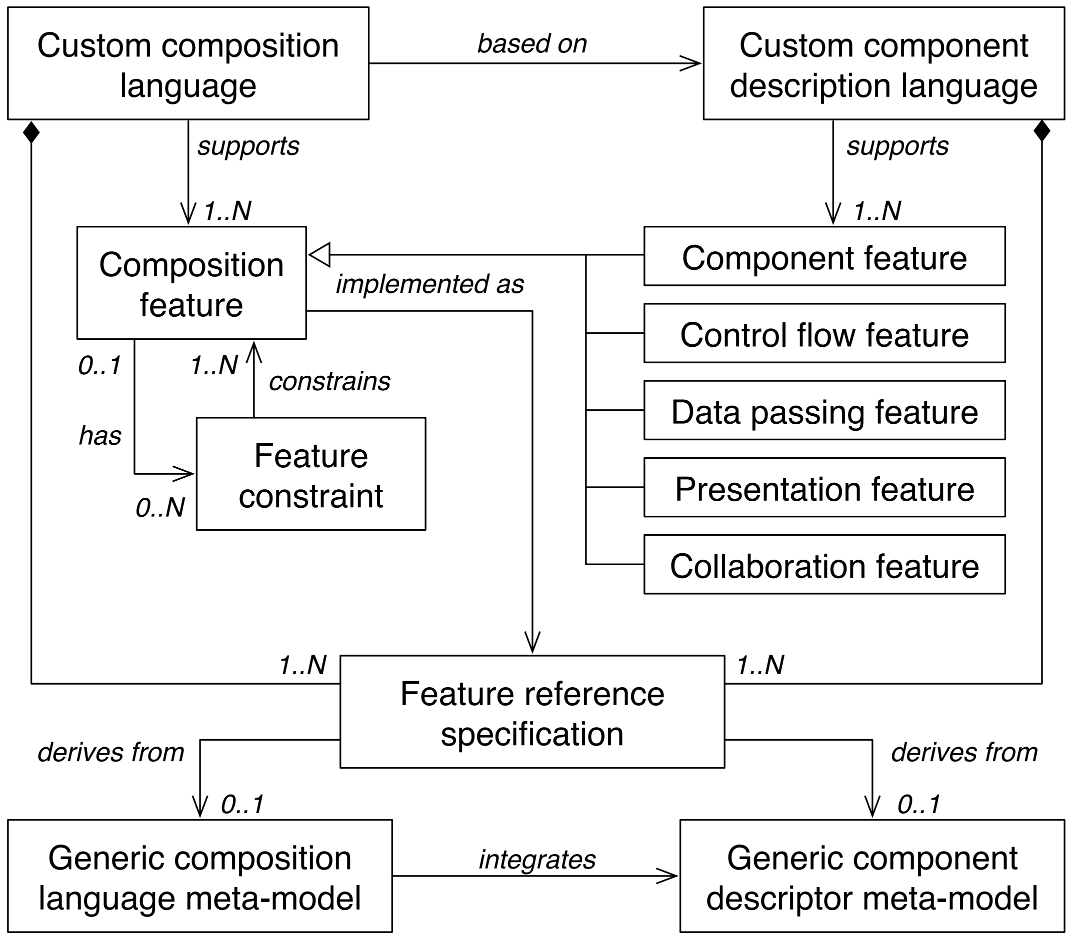
Conceptual development
of mashup languages/platforms [Soi2014]

Observation: All mashup languages share similar **features**

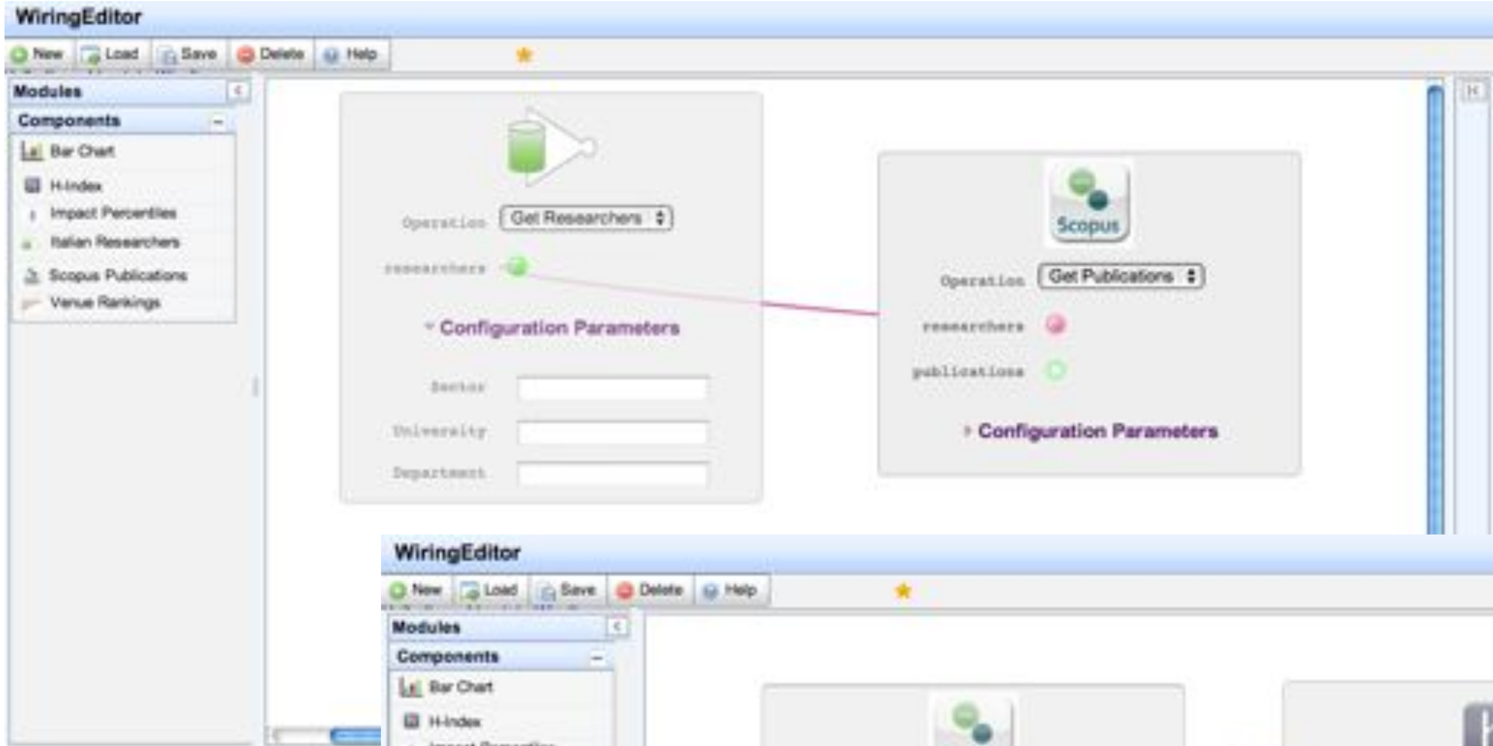
Idea:

1. **Extract/isolate** features
2. Express features as reusable mashup language **patterns** (XSD)
3. Implement a **library** of features (XSD)
4. Identify **conflicts** and inclusions (simple rules)
5. Develop a **runtime environment** that supports all features
6. Develop new languages by **assembling** features
    - Mashup language
    - Component description language
7. **Customize** the runtime environment with new language

Concept



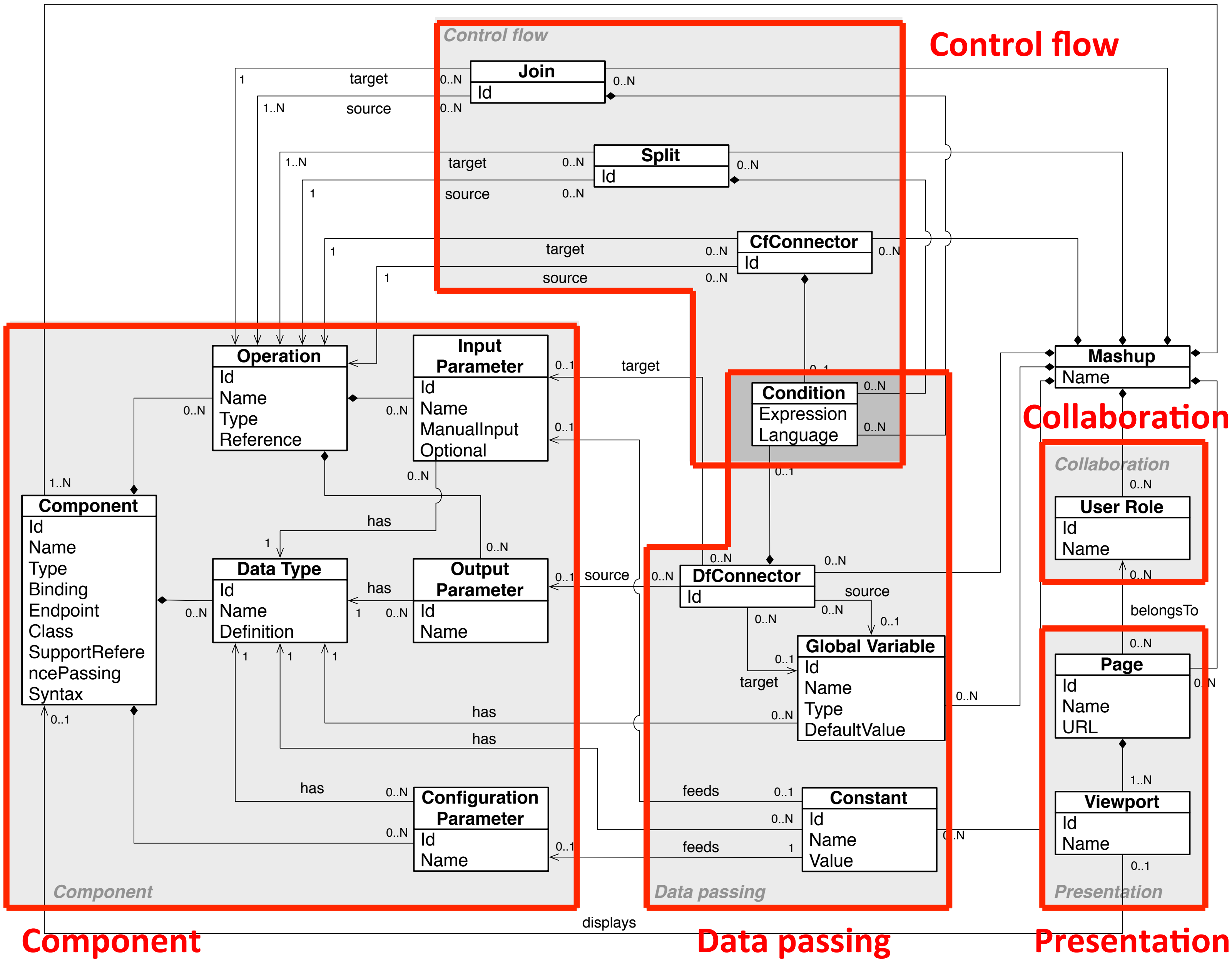
Custom composition language
based on
Custom component description language
supports
1..N
supports
1..N
Composition feature
implemented as
Component feature
Control flow feature
Data passing feature
Presentation feature
Collaboration feature
0..1
1..N
constrains
has
Feature constraint
0..N
1..N
1..N
Feature reference specification
derives from
0..1
derives from
0..1
Generic composition language meta-model
integrates
Generic component descriptor meta-model



(a) An instantiation of the mashup editor based on the **data flow** paradigm

(b) An instantiation of the editor using a **control flow** paradigm with **global variables** for data passing

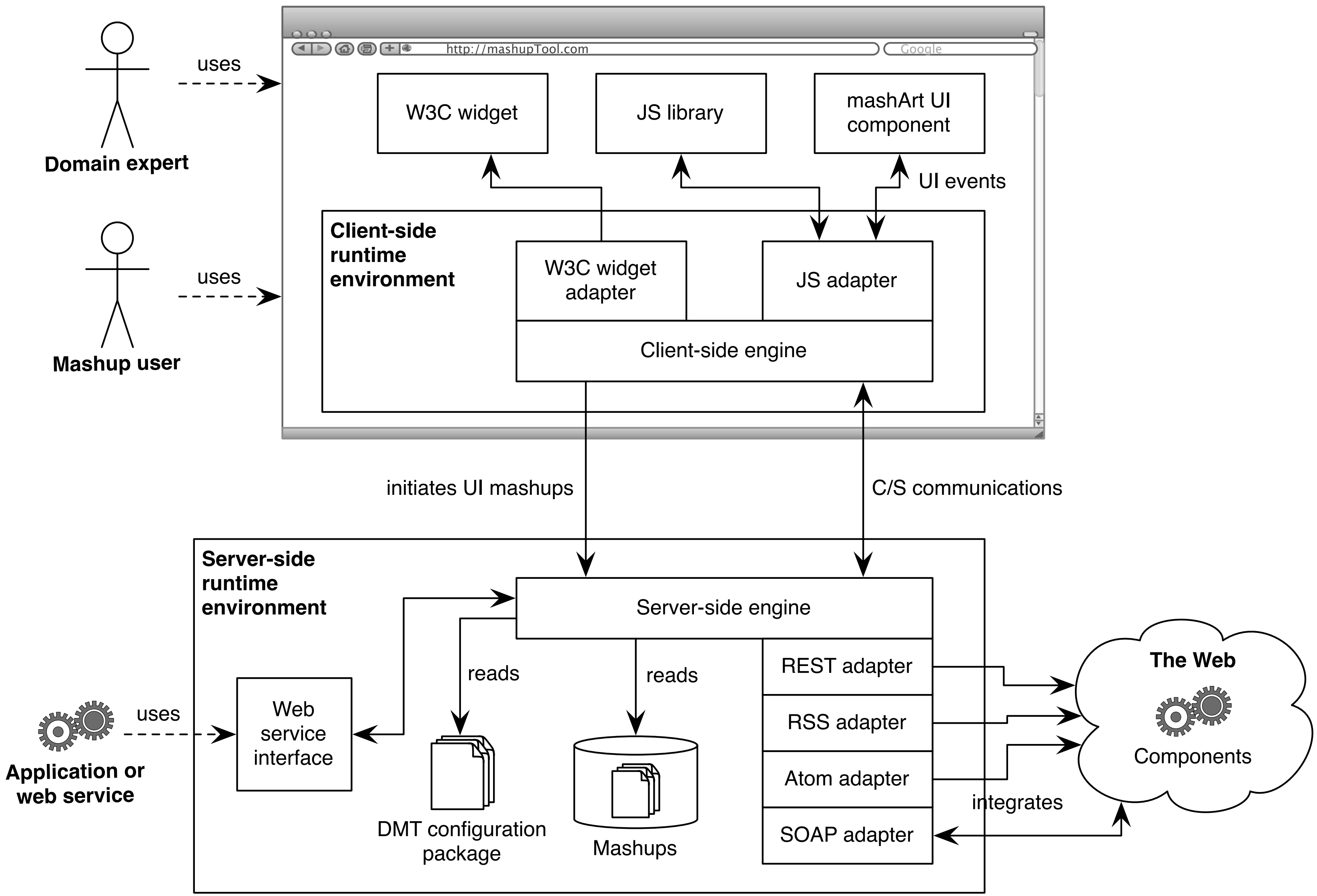
Generic mashup language model (not executable!)



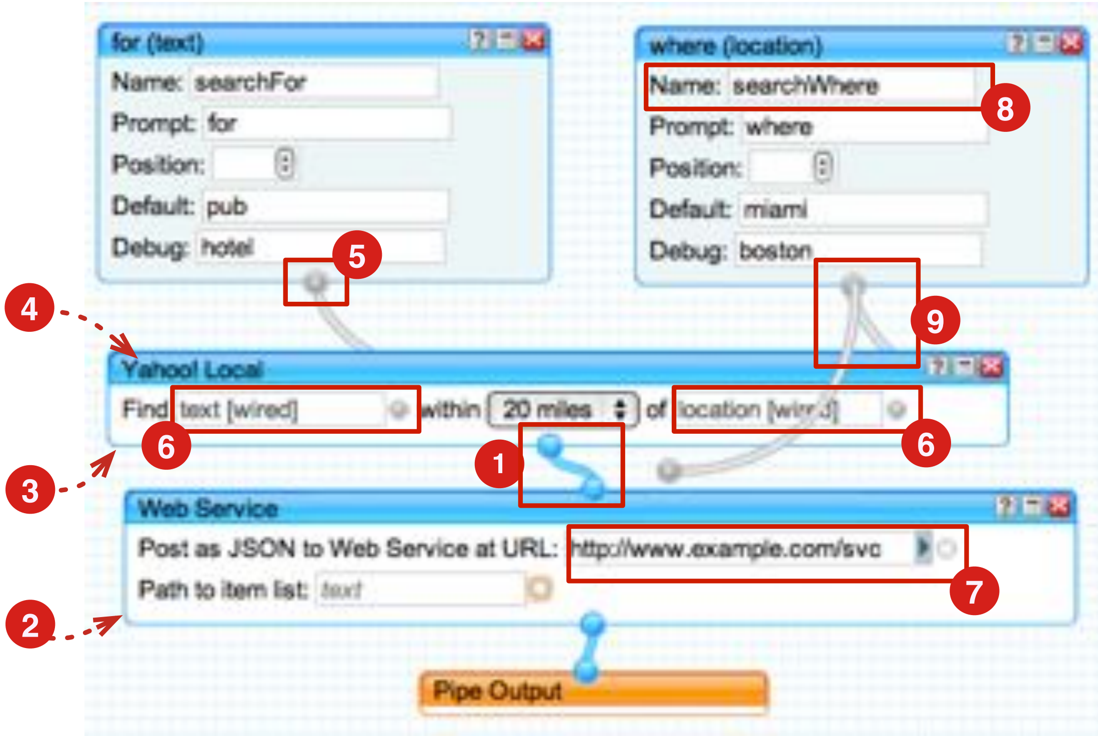
Example of **feature specification**

```
<feature name="data_flow" label="Data flow">

  <description> The composition paradigm is data flow, that is, it is possible
       to explicitly define the flow of the data among components opearations.
       In this case the data passing and the control flow overlap since
       operations triggering depends on the data flow.
  </description>

  <specification>

    <include fragments="dfConnectorDef, dfConnectorType,
         dfSourceOutputParameter, dfTargetInputParameter" />

  </specification>

  <constraints>NOT(control_flow)</constraints>
</feature>
```

Runtime environment (operational semantics of language)



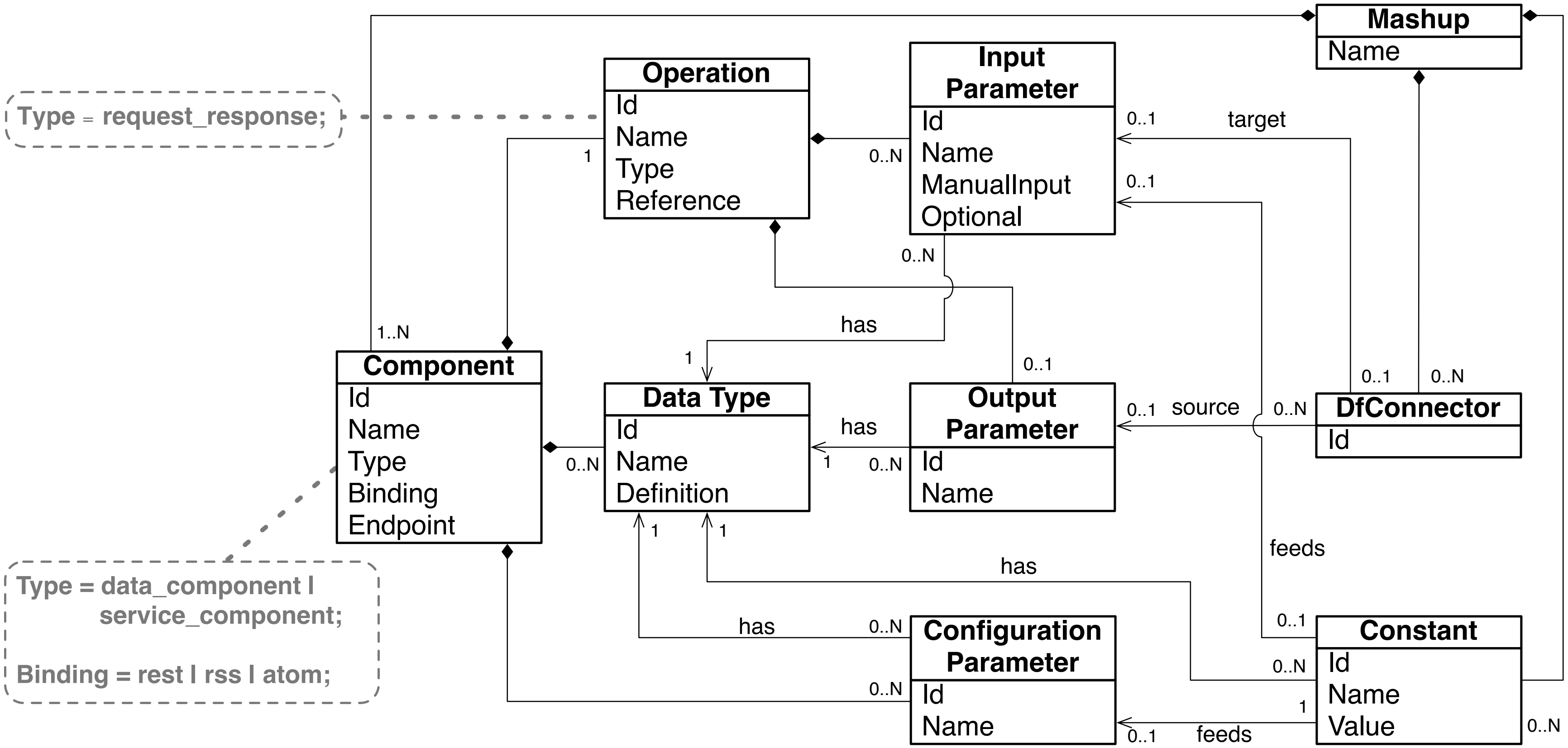
Conceptual development example: **Yahoo! Pipes**



Selected language features

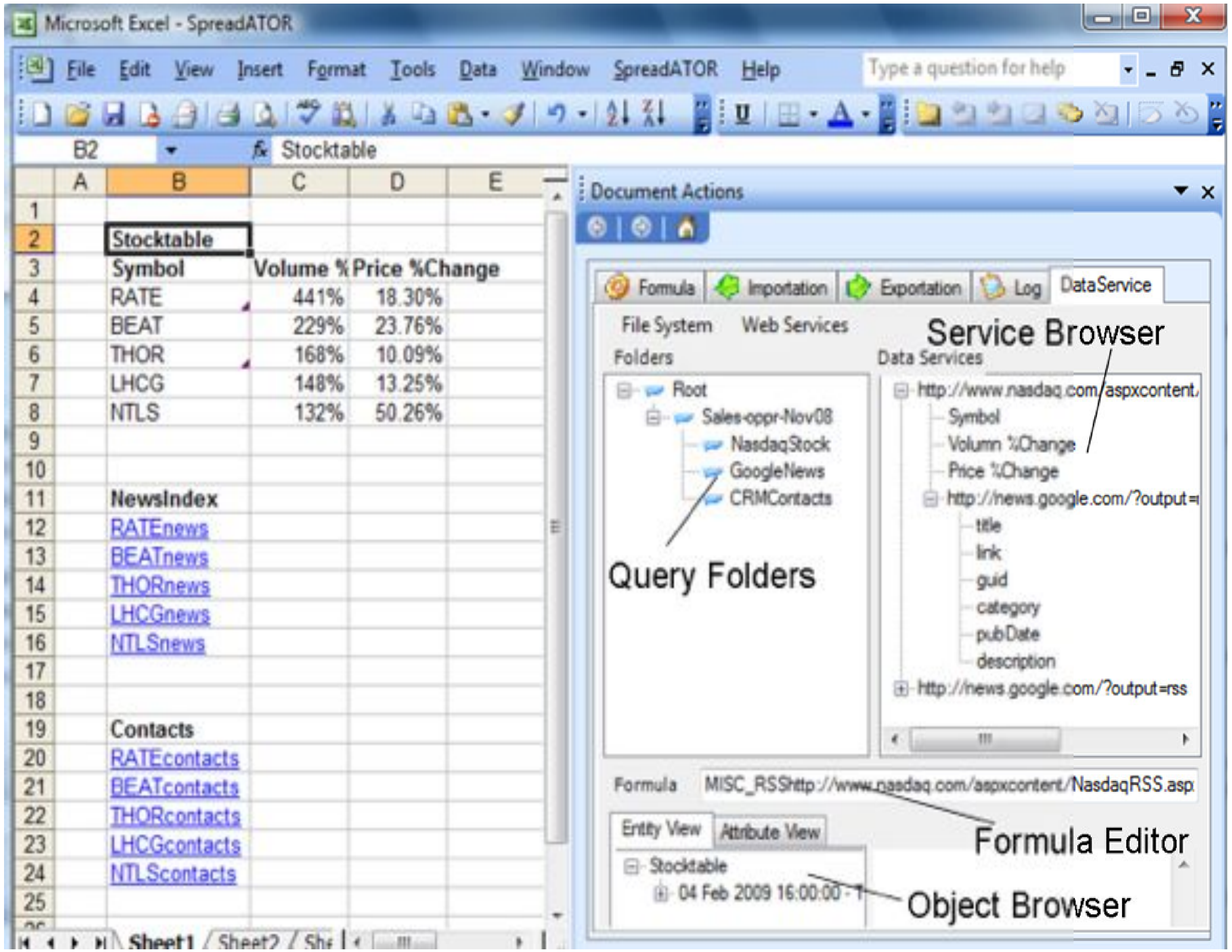
1. data_flow
2. service_component, REST_for_service
3. data_component, RSS_for_data, atom_for_data
4. min_1_operation_per_component, max_1_operation_per_component, request_response
5. min_1_output_param_per_operation, max_1_output_param_per_operation
6. min_1_intput_param_per_operation, max_N_intput_param_per_operation
7. manual_input
8. configuration_param
9. branch

Resulting mashup **language model**

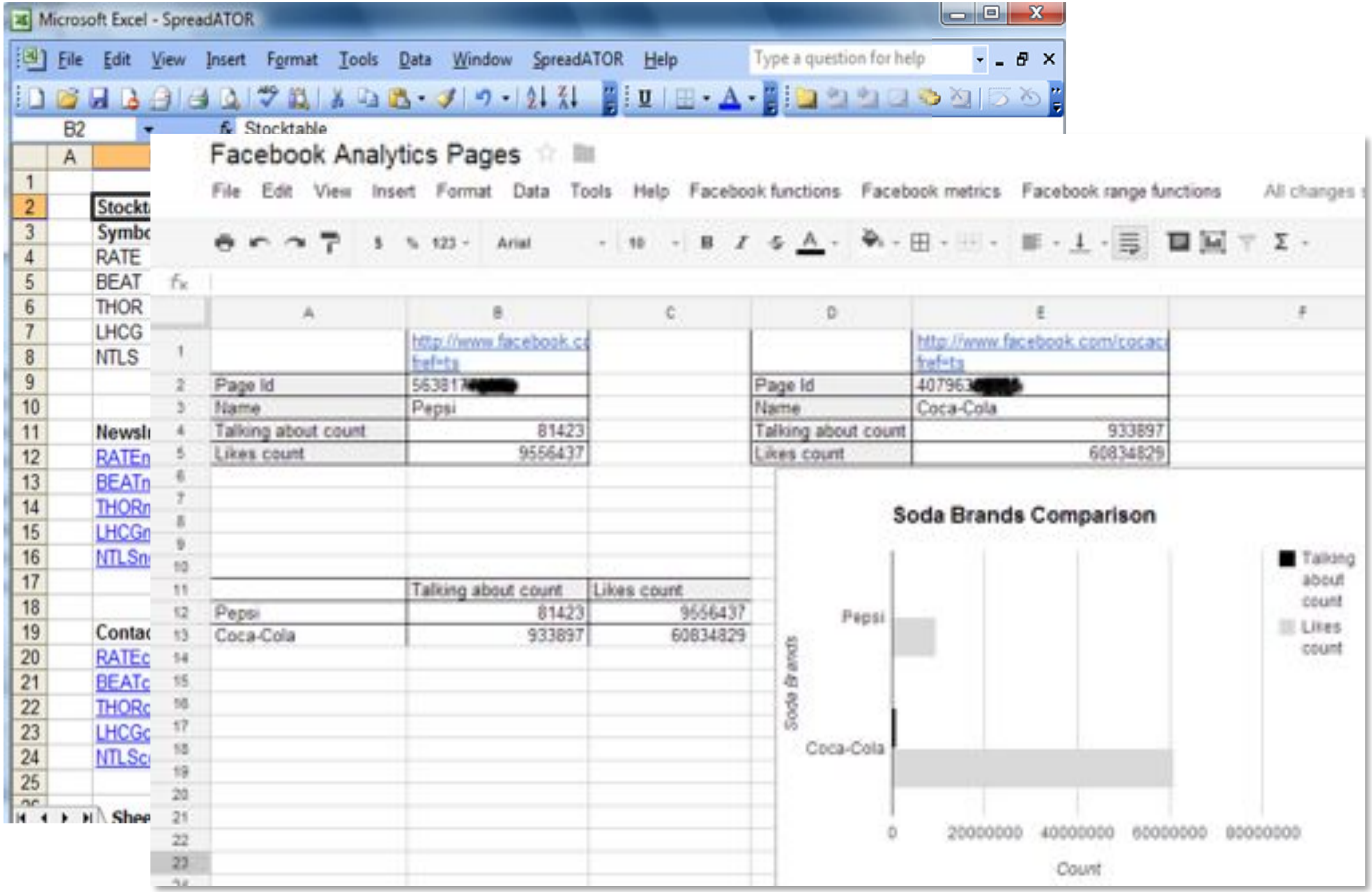


Attention: this **model** is **by design** different from the **metamodel** presented earlier!

OTHER MASHUP LANGUAGES



Spreadsheet-based mashups [Kongdenfha2009]



Spreadsheet-based mashups [Kongdenfha2009] Social Spreadsheet [Jara2013]

NaturalMash [Aghaee2013] = **controlled natural language**

Menu showing the list of available components. Users can add components in the mashups by dragging and dropping them into the itneractive workspace

Once a component is added into the workspace, its UI is immediatly displayed and its behaviour synchronizied with the other components, according to "default bindings" based on component compatibility

The user can enrich the default synchronization behaviour, and define further component couplings by selecting possible behaviours that the two components have to show within the final application

PEUDOM [Matera2013] = **live, visual programming**

REFERENCE ARCHITECTURE

Reference architecture for mashup tools

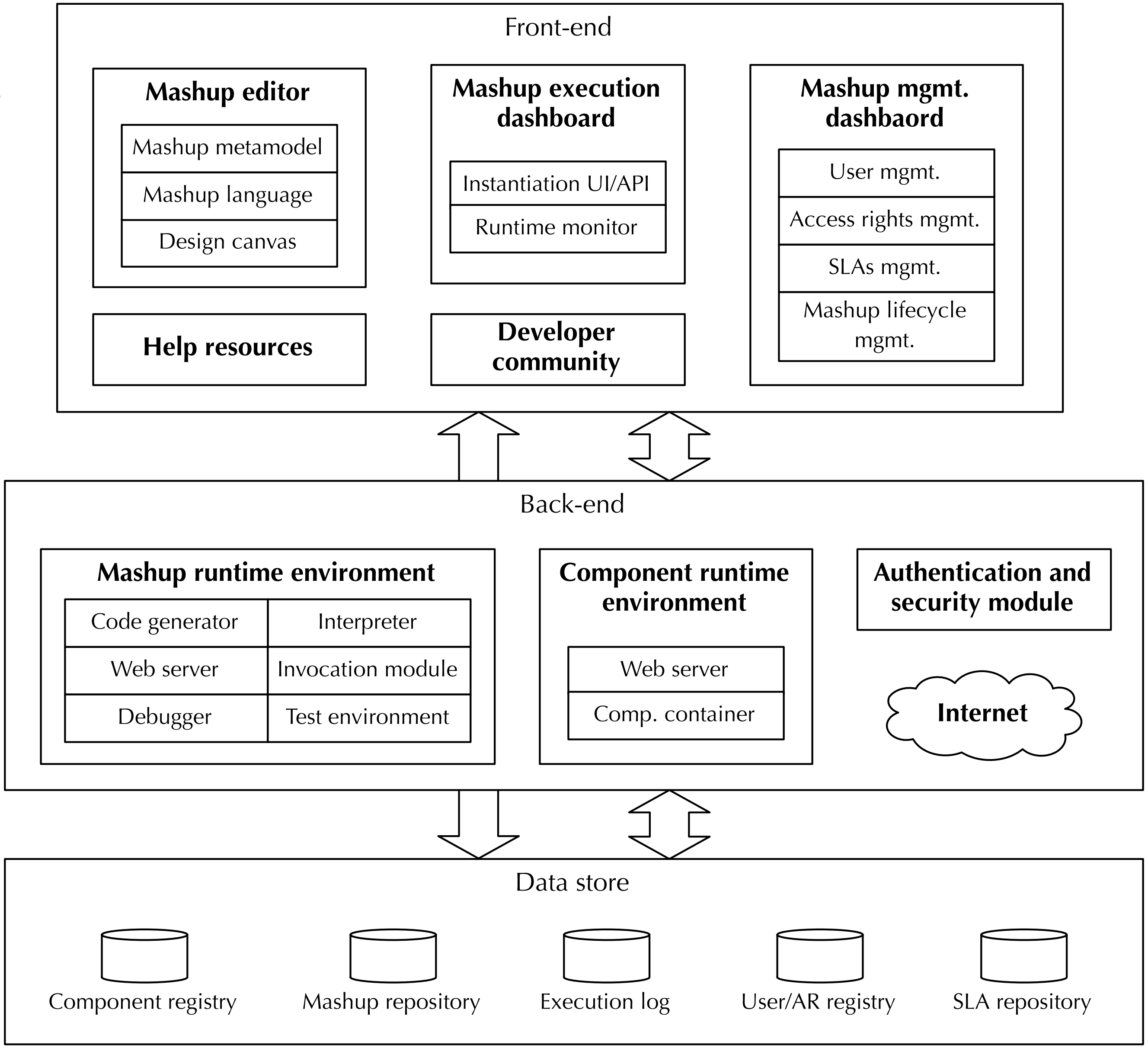


Fig. 8.15 Conceptual reference architecture of a mashup platform articulated into front-end, back-end and persistent data store.

Part IV

MASHUP QUALITY

Learning objectives

1. **Component quality**
   – Definition of the main data quality dimensions to evaluate mashup components
2. **Composition issues**
   – Issues related to the assessment of the quality of composed application
3. **Mashup quality**
   – Definition of the data quality dimensions to evaluate mashup applications

The importance of quality

Garbage in → garbage out

Quality

"Even though quality cannot be defined, you know what it is…."

Robert Pirsig

Is Quality Measurable?

Quality Assessment

What?

Dimensions

How?

Metrics

We need a quality model!

Mashups

"Mashups are Web applications that integrate inside one web page *two or more* heterogeneous resources…."

….Integration of two or more heterogeneous sources…

Quality of a composed object

The quality of the composed objects depends only on the quality of the components?

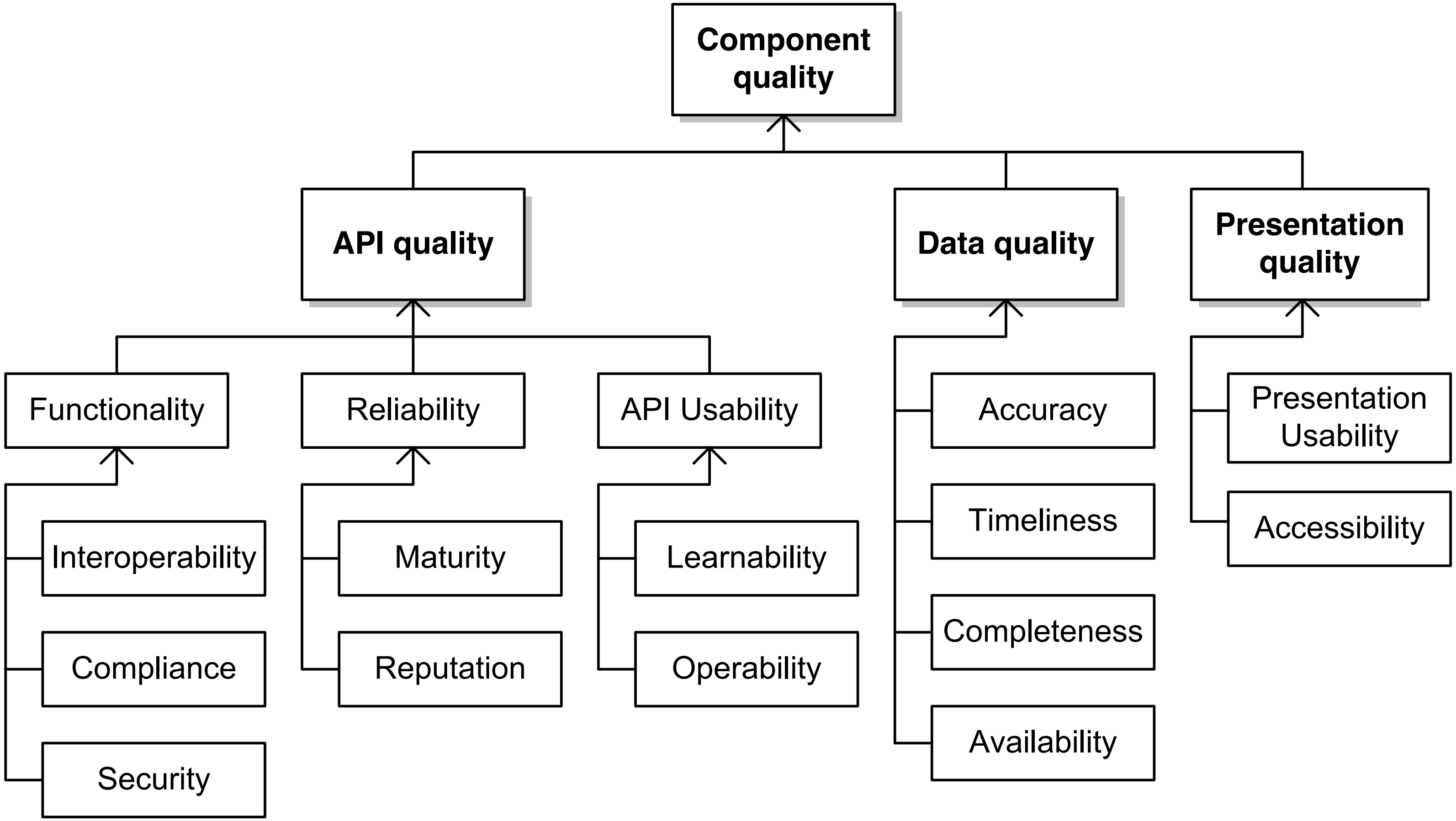
FROM THE COMPONENTS …A QUALITY MODEL

The structure of an API

Quality contributions

- Software quality dimension: ISO standard
- Contributions addressing quality of software components: complexity, modularization, cohesion, coupling

The quality model an overview…



[Cappiello2009]

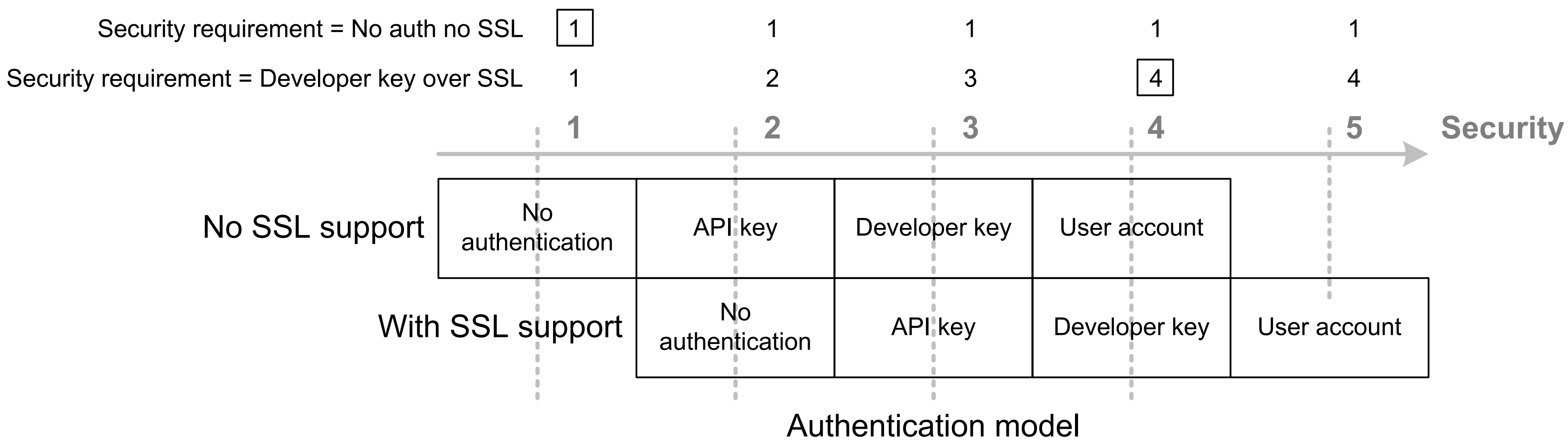
API quality - Functionality

Interoperability **Compliance**

Protocols
Language
Data formats

API quality – Functionality

Security



API Quality - reliability

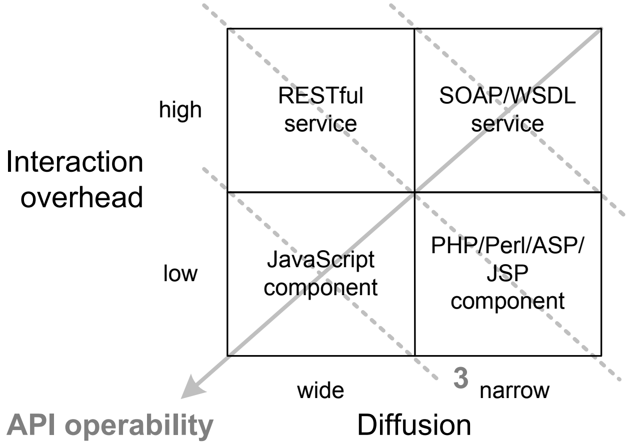
$$Maturity_{comp} = max(1 - \frac{CurrentDate_{comp} - LastUseDate_{comp}}{\frac{CurrentDate_{comp} - CreationDate_{comp}}{|V_{comp}|}}; 0)$$

Age

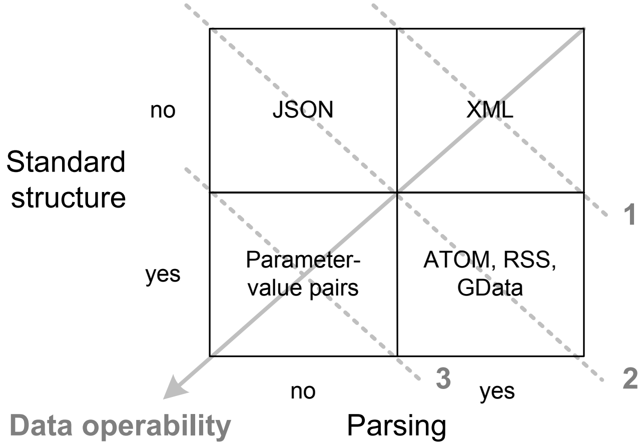
Usage

Maintainance

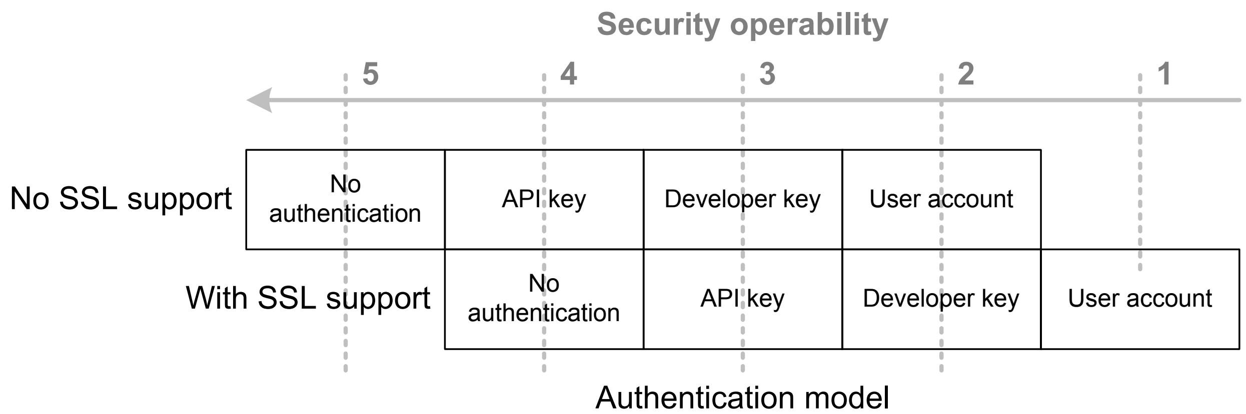
API usability - operability



(a) A metric to measure operability of API types



(b) A metric to measure the operability of data formats



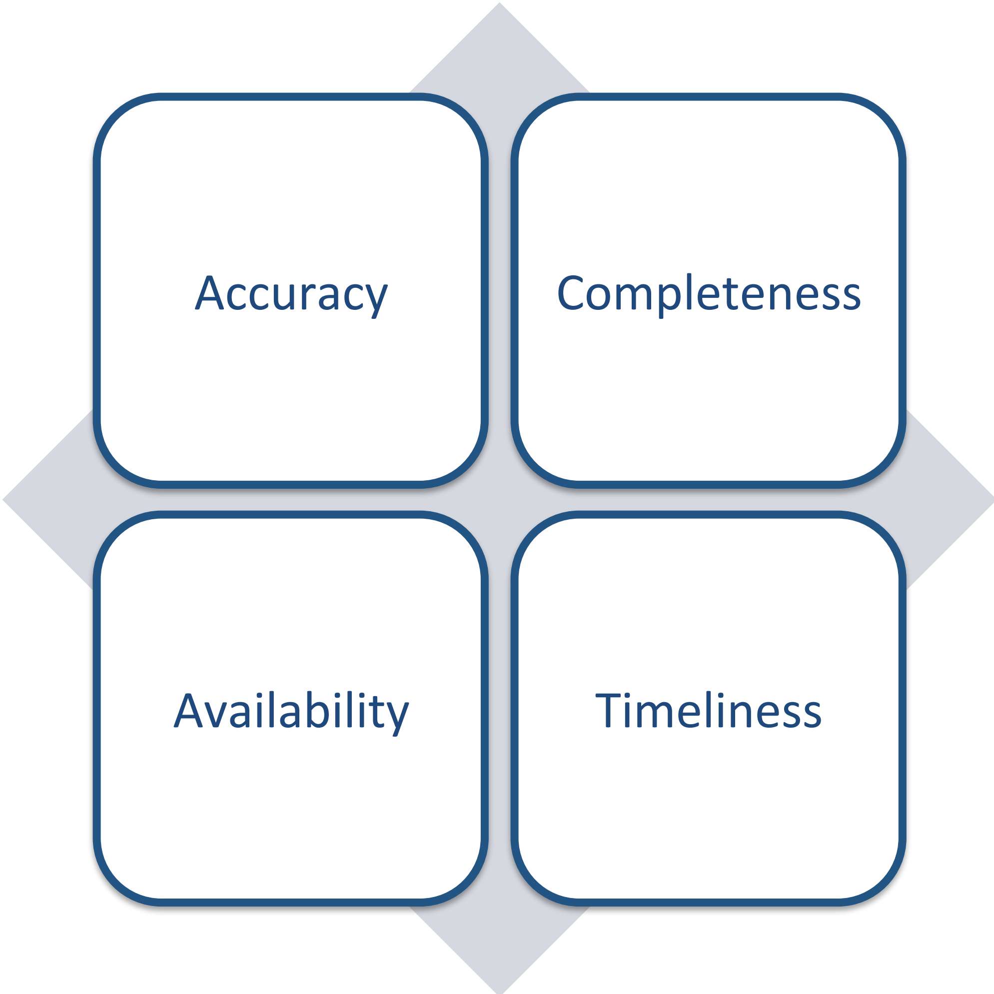
API usability - learnability

Documentation

Examples

```
import java.util.Scanner;
import org.scribe.builder.*;
import org.scribe.builder.api.*;
import org.scribe.model.*;
import org.scribe.oauth.*;

public class TwitterExample
{
    public static void main(String[] args)
    {
        OAuthService service = new ServiceBuilder()
            .provider(TwitterApi.class)
            .apiKey("6icbcAXyZx67rBuTAUM5Qw")
            .apiSecret("SCCAdUUc6LXxiazxH3N0QfpNUvlUy84mZ2XZKiv39s")
            .build();
        Scanner in = new Scanner(System.in);
        Token requestToken = service.getRequestToken();

        System.out.println(service.getAuthorizationUrl(requestToken));
        System.out.println("And paste the verifier here: ");
        Verifier verifier = new Verifier(in.nextLine());

        Token accessToken = service.getAccessToken(requestToken, verifier);
    }
}
```

Data Quality



Presentation Quality

Usability

Accessibility

Reputation

TO THE COMPOSITION…A QUALITY MODEL

Mashup quality: main aspects

Mashup component:
Google Maps

+

Mashup component:
Flickr

Garbage in → Garbage out

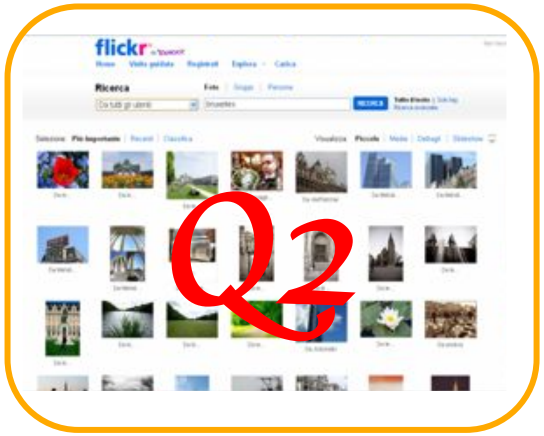
Garbage inside → Garbage out

Mashup application:
e.g., BrusselStripstad.be

Quality assessment?

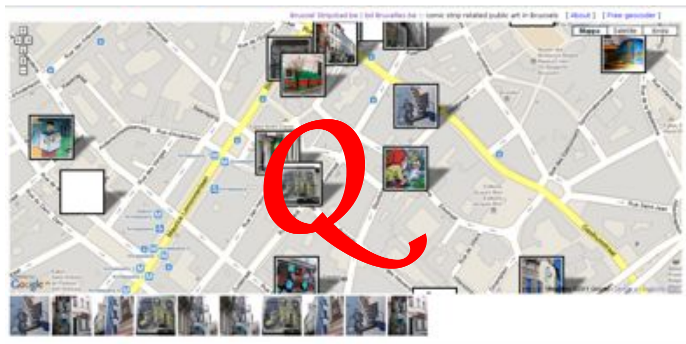
Mashup component:
Google Maps

+



Mashup component:
Flickr

=



?

Mashup application:
e.g., BrusselStripstad.be

Quality assessment: a first experiment

Mashups are accessible as **normal Web pages**...
...can we use the **same models** and tools developed for quality assessment of traditional web pages?

- We assess (by using automatic tools) the quality of 68 mashups on the basis of four criteria:
- ***Usability***: measures the ease of use of the mashup. (SiteAnalyzer)
- ***Readability***: measures how easy or difficult it is to read and understand the text rendered in the mashup. (Juicy Studio)
- ***Accessibility***: measures how well the mashup complies with the W3C web accessibility guidelines. (SiteAnalyzer)
- ***Performance***: measures the loading time of the mashup till all elements of the application are rendered in the page. (Pingdom)

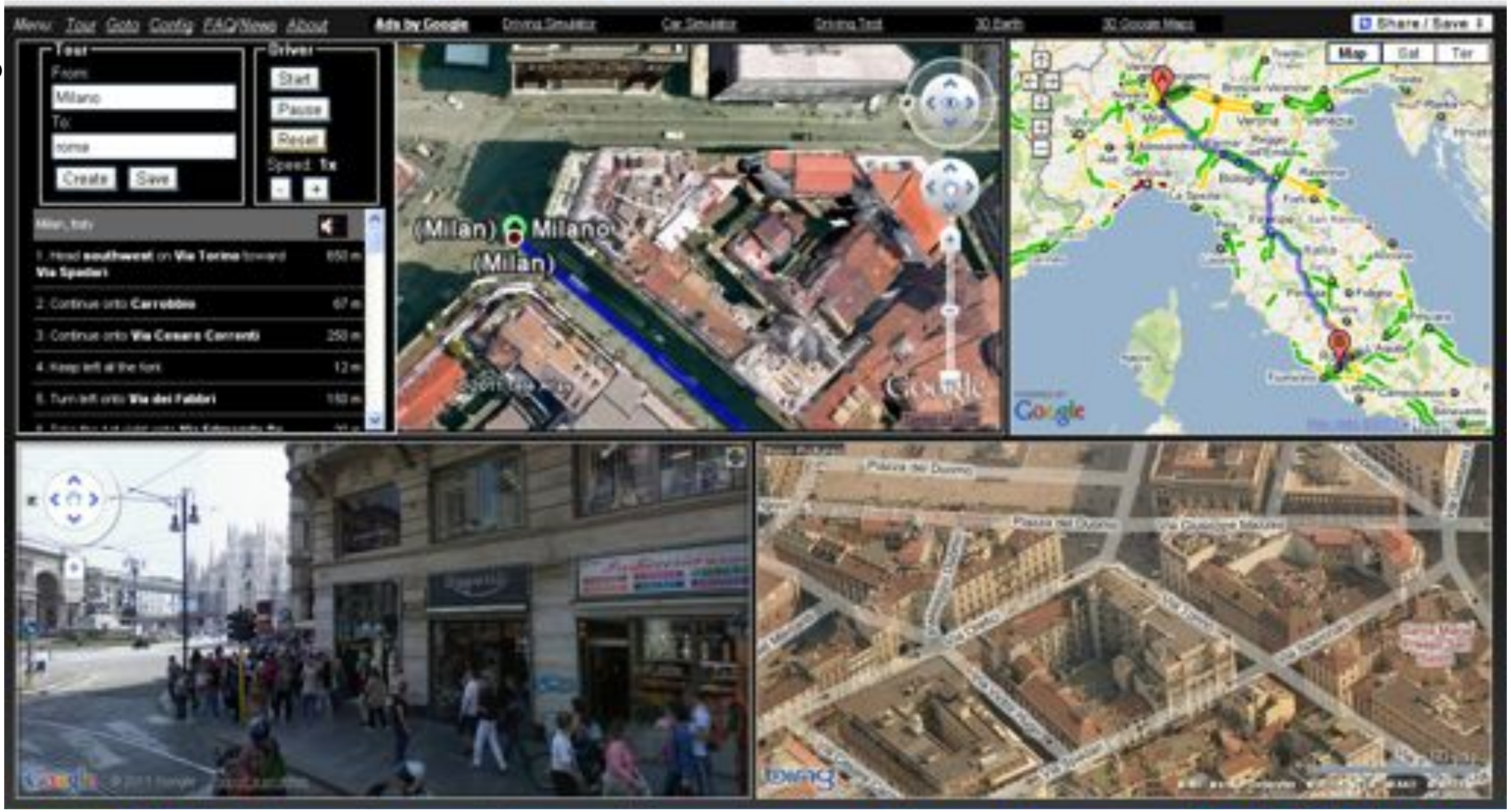
Results of the experiment: five “best" and five “worst" mashups.

Rating	Mashup	Usability	Readability	Accessibility	Performance
1	A Paris Hilton video site	83.9%	65.2	85.5%	3.1 sec.
2	Sad Statements	81.4%	35.2	80.5%	5.1 sec.
3	ShareMyRoutes.com	79.8%	62.9	78.4%	2.8 sec.
4	DiveCenters.net	79.5%	75.0	73.3%	1.4 sec.
5	Cursebird	79.1%	78.1	79.3%	2.1 sec.
...	...	...	...	...	...
64	CityTagz	65.0%	53.3	64.2%	3.2 sec.
65	Blue Home Finder	64.9%	77.7	63.9%	7.3 sec.
66	Gaiagi Driver - 3D Driving Simulator	64.8%	53.9	64.6%	6.3 sec.
67	2008 Beijing Olympics Torch Relay Path	62.2%	10.5	58.7%	2.0 sec.
68	Tidespy: Tide Charts with Best Fishing Times	61.2%	58.2	64.3%	5.7 sec.

- Are these results **reliable**? We conducted five independent evaluations by manually inspecting the same mashups and we compared the two evaluations and we found…

... a counterexample...

•



Mashup quality: the need for a quality model

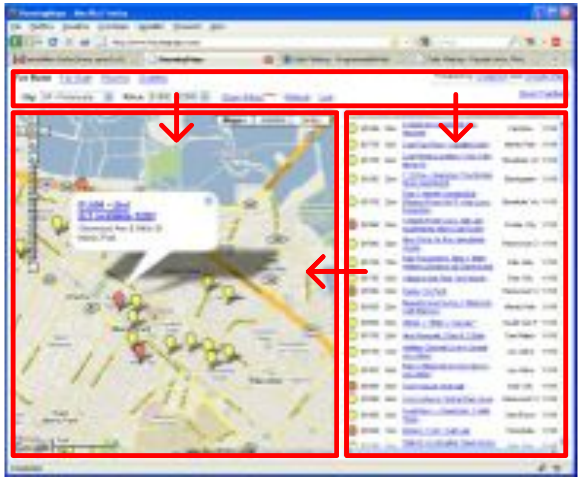
- High focus on **composition** aspect in mashups:
  - Data integration
  - Service orchestration and UI synchronization
  - Layout

- The success of a mashup is certainly influenced by the **added value** that the final combination of components is able to provide.

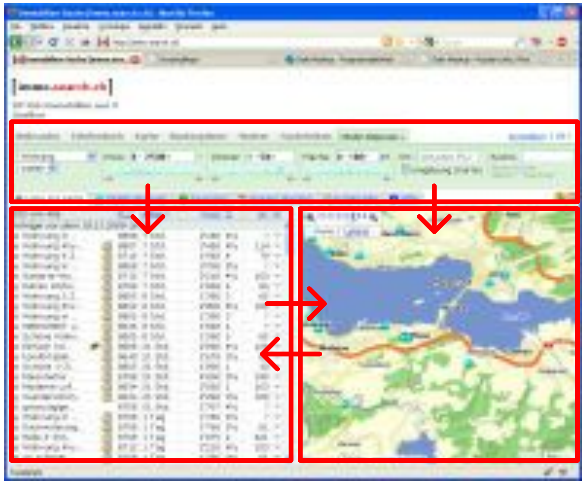
Composition patterns [Cappiello2010]

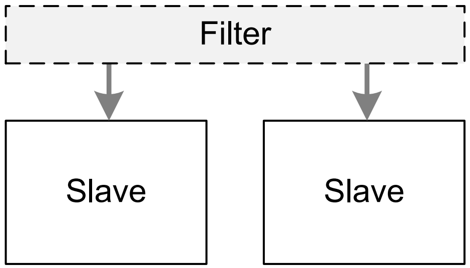
http://dailymashup.com/

http://www.housingmaps.com/

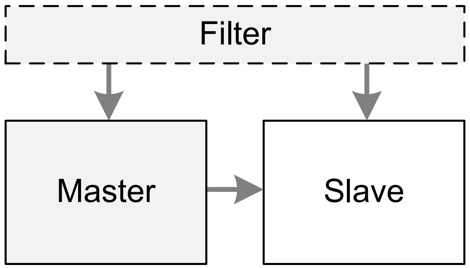


http://immo.search.ch/

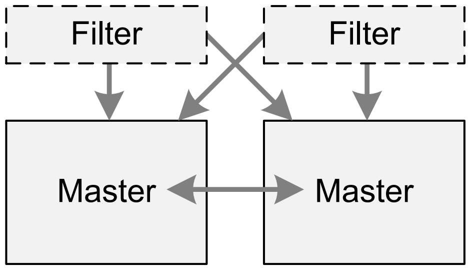




(a) Slave-Slave pattern



(b) Master-Slave pattern

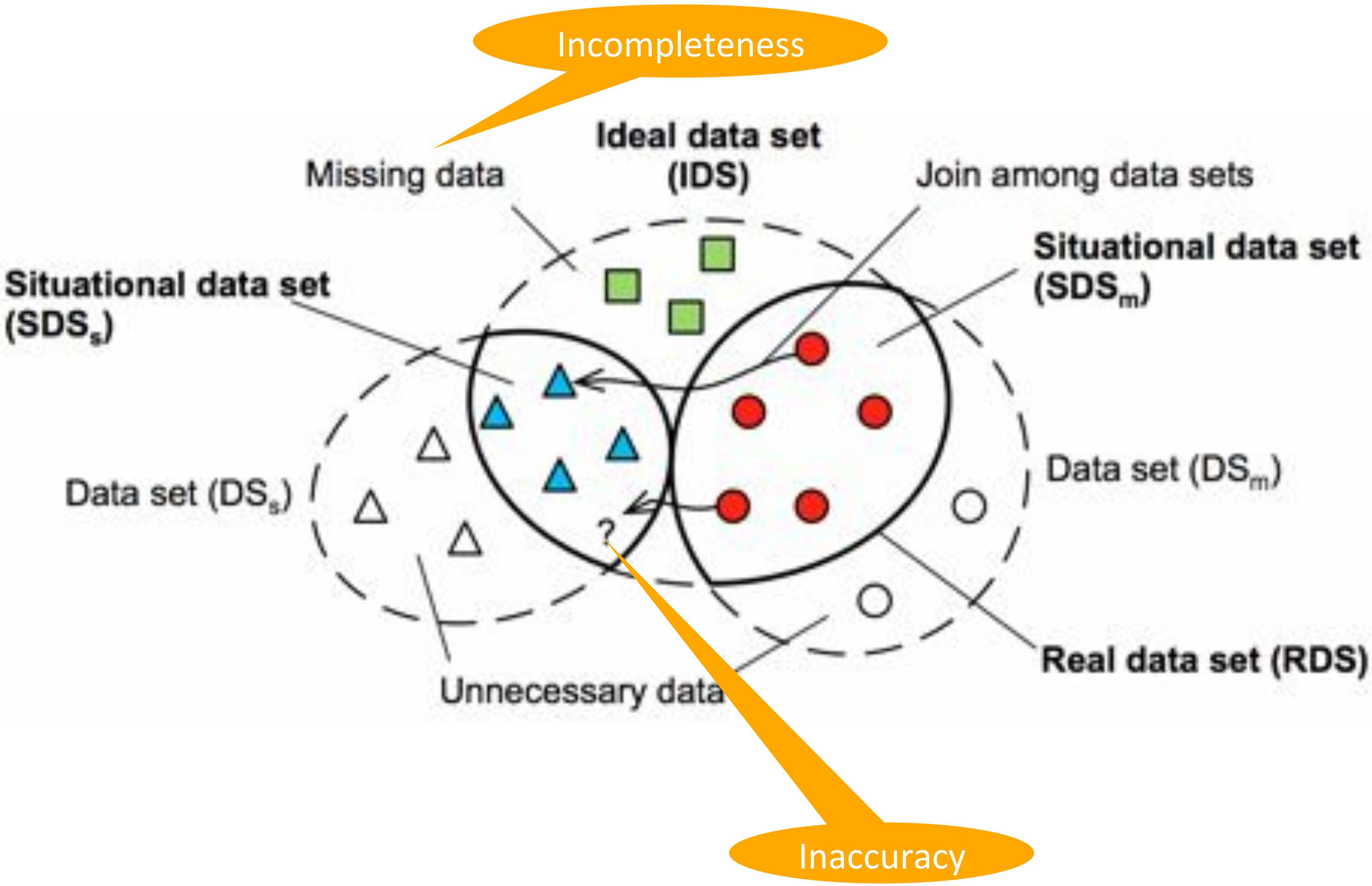


(c) Master-Master pattern

The quality model

[Cappiello2011]

Data Quality



Presentation quality

•Usability

•**traditional** dimensions such as orientation, users control, predictability, layout consistency

•**Learnability**: the mashup features should be visible enough and the corresponding commands should be self-expressive so that even naive users can easily master the mashup execution.

•**Layout consistency**

•Accessibility

•Accessibility criteria do not need to be specialized for mashups.

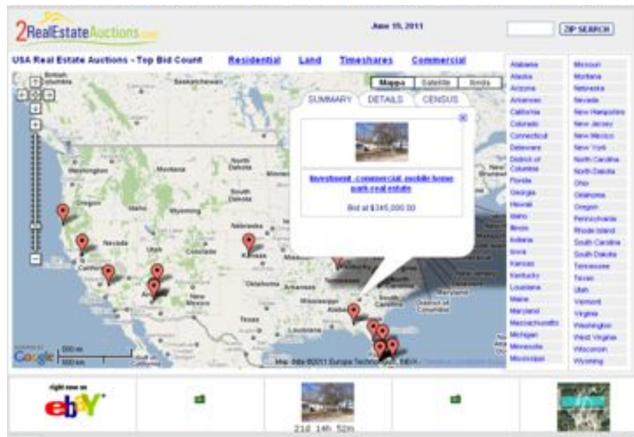
•Usability

•navigability and richness of links, or any other criteria addressing the hypertext structure

•readability, cohesion or coherence

Composition Quality: added value

- The added value of the composition can be related to the amount of provided **features and/or offered data**. The mashup has an added value if it provides at least more functionality or data than the ones provided by its components



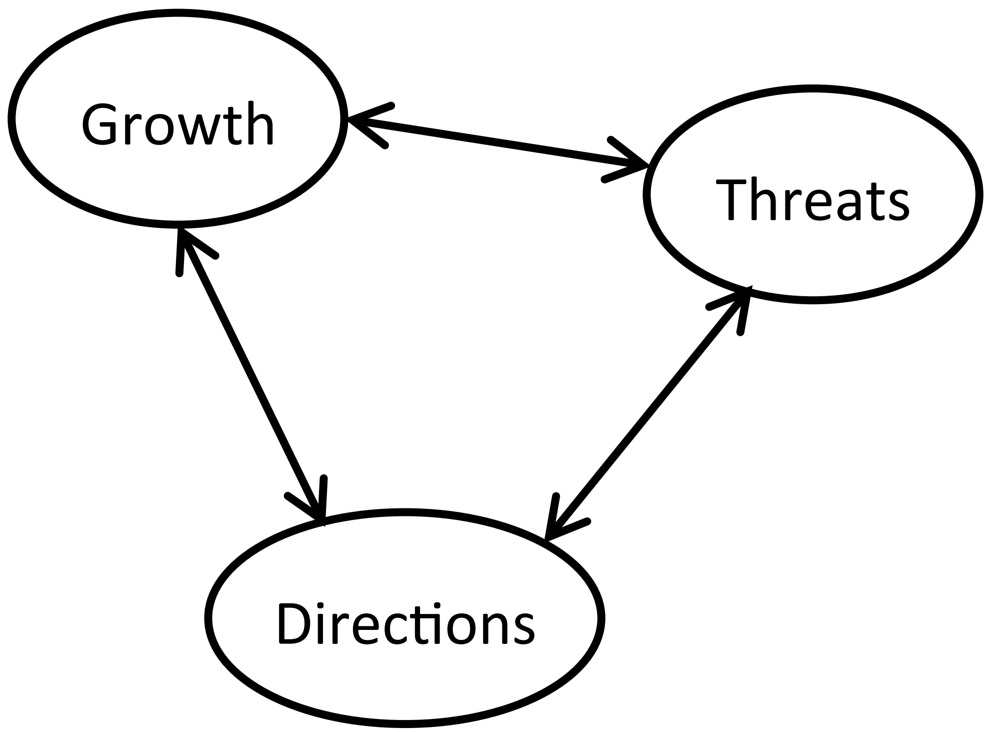
Composition Quality: other dimensions

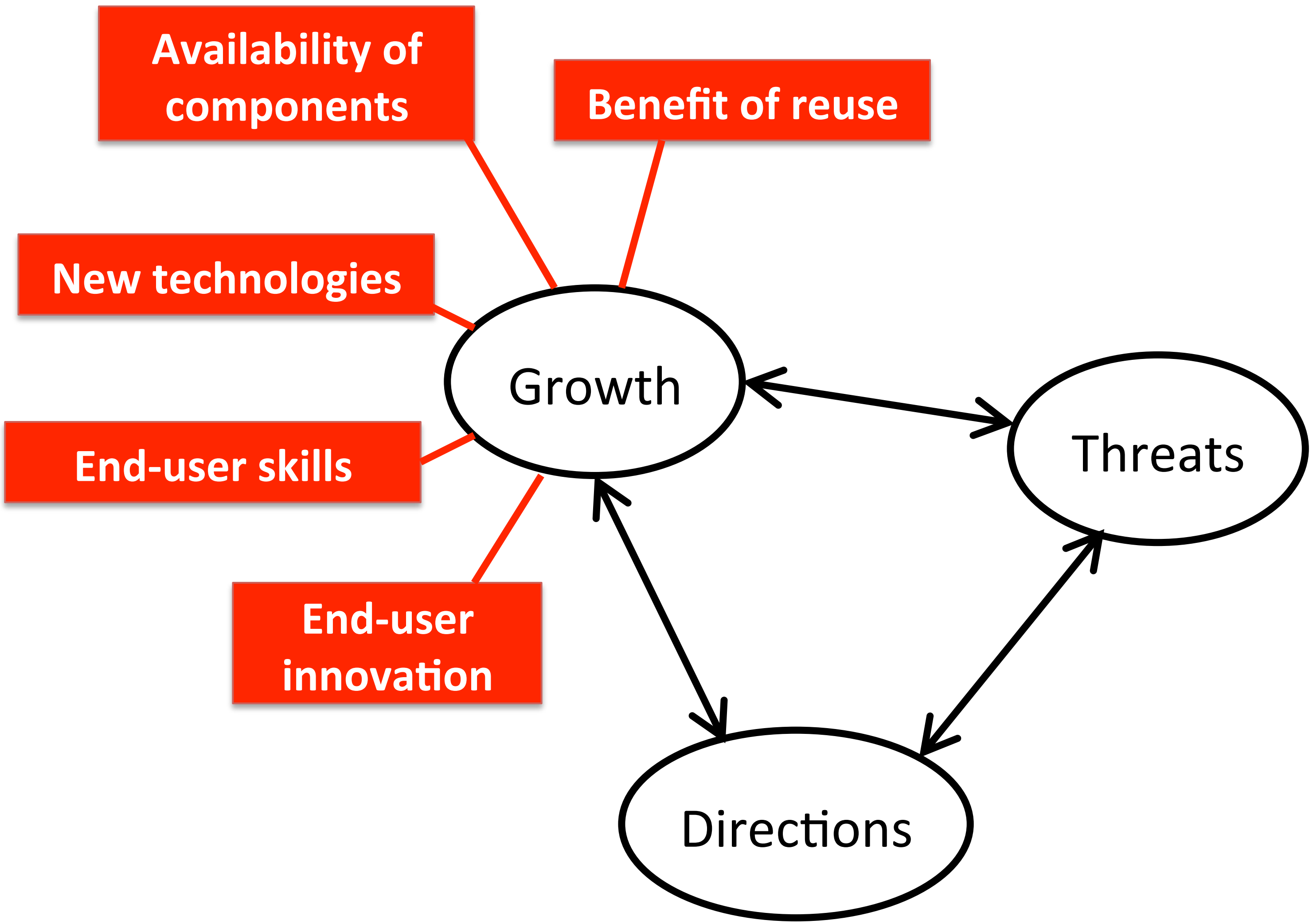
- **Component suitability**: it refers to the appropriateness of the component features and data with respect to the goal that the mashup is supposed to support.
- **Component usage**: it may happen that, even though a component is very rich from the point of view of data and functionality, it is improperly used within a composition.
- **Consistency**: poor quality compositions can also be caused by inconsistencies at the orchestration level.
- **Availability**: the degree in which the mashup can be properly accessed during a given time interval. It depends on the availability of the components and on their role in the composition.

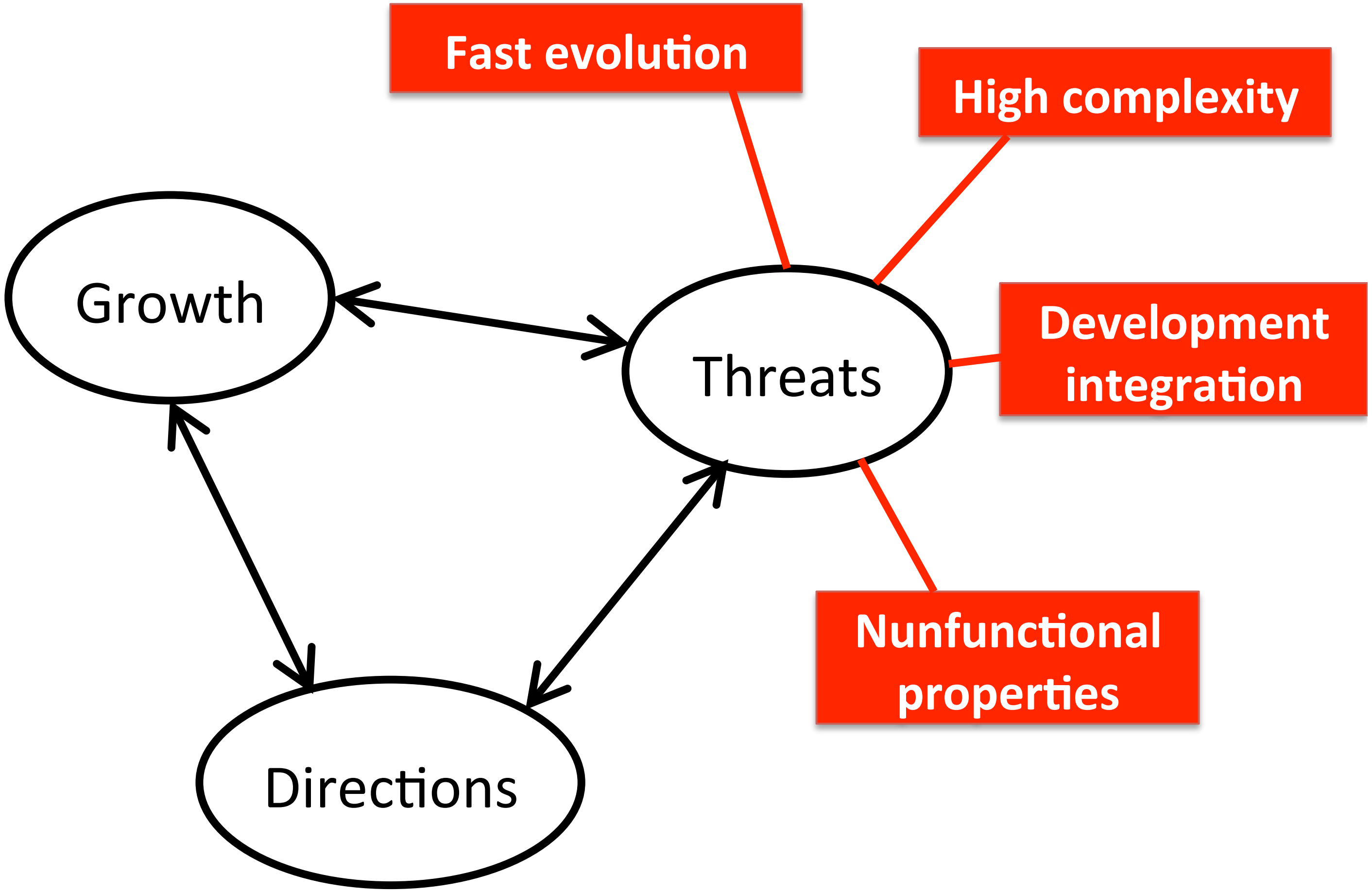
Remember….

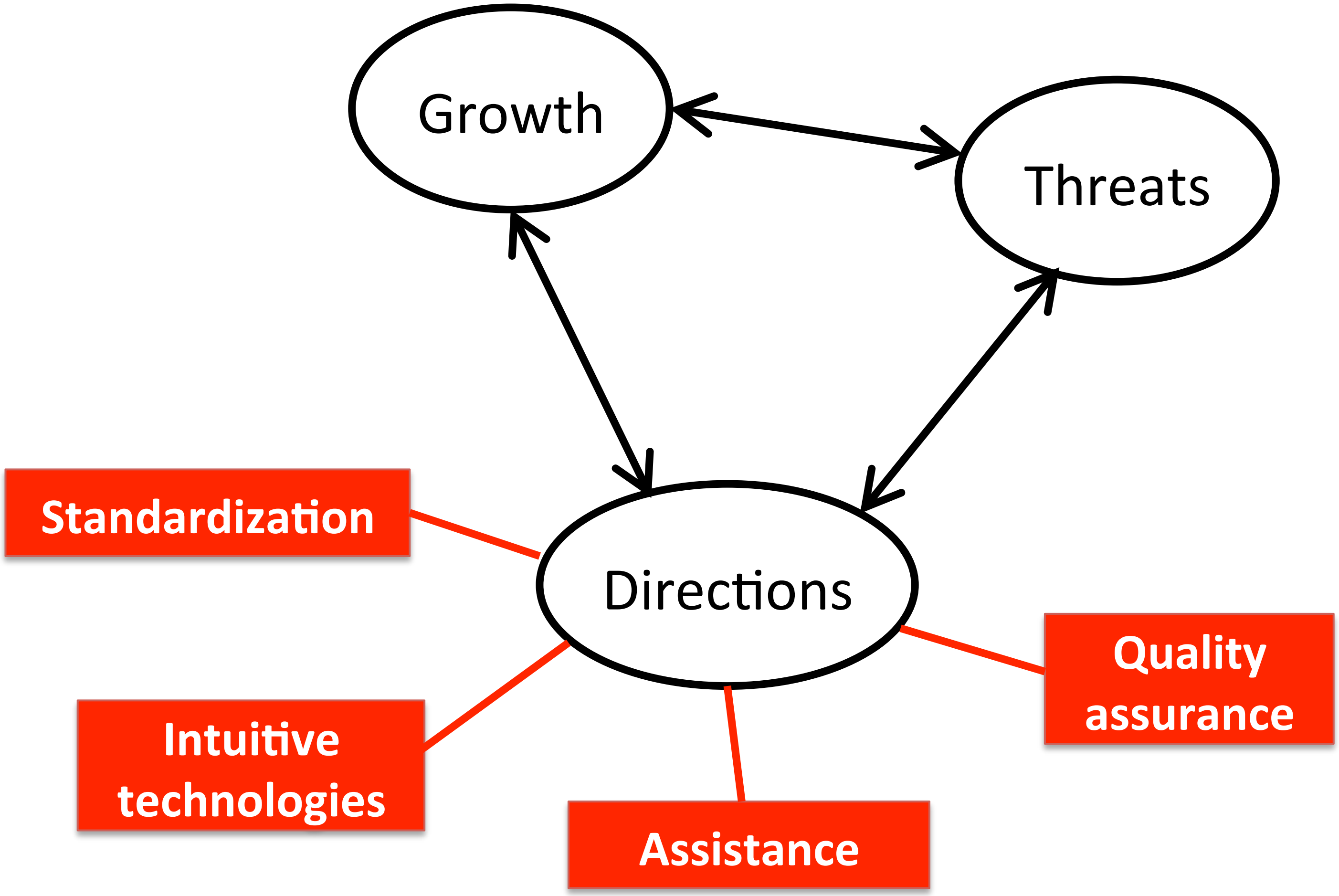
Garbage in → garbage out

OUTLOOK

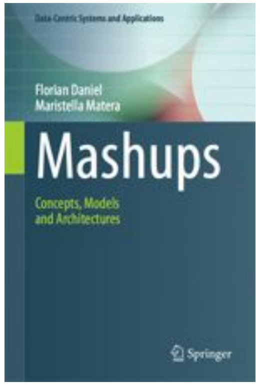

Growth
Threats
Directions


Availability of components
Benefit of reuse
New technologies
End-user skills
End-user innovation
Growth
Threats
Directions


Fast evolution
High complexity
Growth
Threats
Development integration
Directions
Nunfunctional properties


Growth
Threats
Directions
Standardization
Intuitive technologies
Assistance
Quality assurance

Main reference



[Daniel2014] F. Daniel and M. Matera. *Mashups: Concepts, Models and Architectures*. Springer, 2014. ISBN 978-3-642-55048-5.

Other references

[Soi2014] S. Soi, F. Daniel and F. Casati. Conceptual Development of Custom, Domain-Specific Mashup Platforms. *ACM Transactions on the Web*, 2014, accepted for publication.

[Aghaee2013] S. Aghaee, C. Pautasso, A. De Angeli. Natural End-User Development of Web Mashups. *VL/HCC 2013*: 111-118.

[Matera2013] M. Matera, M. Picozzi, M. Pini, M. Tonazzo. Peudom: A mashup platform for the end user development of common information spaces. *ICWE 2013*, pp. 494–497.

[Jara2013] J. Jara Laconich, F. Casati, M. Marchese: Social Spreadsheet. *ICWE 2013*: 156-170

[Rümpel2013] RAndreas Rümpel, Vincent Tietz, Anika Wagner, Klaus Meißner, Modeling and Utilizing Quality Properties in the Development of Composite Web MashupsCurrent Trends in Web Engineering Lecture Notes in Computer Science Volume 8295, 2013, pp 54-65

[Cappiello2011] Cappiello, C., Daniel, F., Koschmider, A., Matera, M., Picozzi, M.: A Quality Model for Mashups. In: Auer, S., Diaz, O., Papadopoulos, G.A. (eds.) ICWE 2011.LNCS, vol. 6757, pp. 137–151. Springer, Heidelberg (2011)

[Cappiello2010] Cinzia Cappiello, Florian Daniel, Maristella Matera, Cesare Pautasso: Information Quality in Mashups. IEEE Internet Computing 14(4): 14-22 (2010)

[Cappiello2009] Cappiello, C., Daniel, F., Matera, M.: A Quality Model for Mashup Components. In: Gaedke, M., Grossniklaus, M., Diaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 236–250. Springer, Heidelberg (2009)

[Daniel2009] F. Daniel, F. Casati, B. Benatallah and M.-C. Shan. Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. *ER 2009*, Pages 428-443.

[Kongdenfha2009] W. Kongdenfha, B. Benatallah, J. Vayssière, R. Saint-Paul, F. Casati: Rapid development of spreadsheet-based web mashups. *WWW 2009*: 851-860

[Ogrinz2009] M. Ogrinz: Mashup Patterns: Designs and Examples for the Modern Enterprise. Addison-Wesley Professional, 2009

[Abiteboul2008] S. Abiteboul, O. Greenshpan, T. Milo: Modeling the mashup space. WIDM 2008: 87-94

[Yee2008] R. Yee: Pro Web 2.0 Mashups: Remixing Data and Web Services. Apress, 2008

[Sarkar2007] Sarkar, S.; Rama, G.M.; Kak, A.C., "API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization," Software Engineering, IEEE Transactions on , vol.33, no.1, pp.14,32, Jan. 2007

[Olsina2005] Olsina, L., Covella, G., Rossi, G.: Web Quality. In: Web Engineering, pp. 109–142. Springer, Heidelberg (2005)

[Calero2004] Calero, C., Ruiz, J., Piattini, M.: A Web Metrics Survey Using WQM. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 147–160. Springer, Heidelberg (2004)