Software Services

By Michał Antkiewicz
http://gsd.uwaterloo.ca/mantkiew
@ WCI CS Club

Context

- Software Engineering
  - Methods and processes of software development
- Evolving enterprise software ecosystem
  - Hundreds of systems, systems of systems
  - Enable “business agility”: the ability to rapidly respond to the ever-changing market
- Service-Oriented Architecture
  - Break monolithic systems into software services

“We move our attention from solving low-level, algorithmic problems, to the large scale software systems running in enterprises”

Reference http://pubs.opengroup.org/architecture/togaf9-doc/arch/
TOGAF is an international standard for enterprise architecture.

“What is an enterprise? What is enterprise architecture?
TOGAF defines "enterprise" as any collection of organizations that has a common set of goals. For example, an enterprise could be a government agency, a whole corporation, a division of a corporation, a single department, or a chain of geographically distant organizations linked together by common ownership.
The term "enterprise" in the context of "enterprise architecture" can be used to denote both an entire enterprise - encompassing all of its information and technology services, processes, and infrastructure - and a specific domain within the enterprise. In both cases, the architecture crosses multiple systems, and multiple functional groups within the enterprise.
Confusion often arises from the evolving nature of the term "enterprise". An extended enterprise nowadays frequently includes partners, suppliers, and customers. If the goal is to integrate an extended enterprise, then the enterprise comprises the partners, suppliers, and customers, as well as internal business units.
The business operating model concept is useful to determine the nature and scope of the enterprise architecture within an organization. Large corporations and government agencies may comprise multiple enterprises, and may develop and maintain a number of independent enterprise architectures to address each one. However, there is often much in common about the information systems in each enterprise, and there is usually great potential for gain in the use of a common architecture framework. For example, a common framework can provide a basis for the development of an Architecture Repository for the integration and re-use of models, designs, and baseline data. “

What is service-oriented architecture?

It is a set of architectural principles postulating that instead of building traditional, monolithic applications, enterprise software systems should be organized into a reusable software services with clearly defined interfaces and functions. Such software services are accessible over a standard protocol and can be used by other applications/services.

Divide & Conquer

- Break the system into modules that can be
  - Independently developed
  - Easily integrated
- What's a good way to divide a big system?

"Divide & Conquer is a standard principle for addressing complexity of any kind.

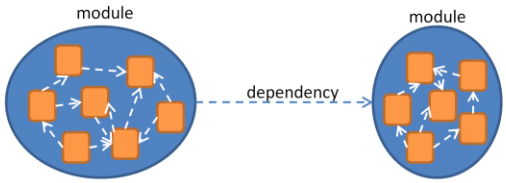
In software design, it means that in order to successfully build a large system, it first needs to be broken down into modules that can be independently developed and easily integrated.

Traditionally, a module is a part of the software that can be assigned to a single team for development.

But there are many ways in which the big system can be divided. What is a good way?"

High Cohesion & Low Coupling

- Standard software design principle
  - Closely related elements form a cohesive module
  - Loosely-coupled modules enable interoperability and reuse



- How to do it in the context of an enterprise?

"High cohesion & low coupling are standard software design principles saying that a good module is cohesive, that is, composed of closely interrelated elements (e.g., classes) and that modules should be loosely coupled, that is, the dependencies among the modules should be minimized.

Figure: "The orange boxes represent elements of the module and the dashed arrows represent dependencies. As can be seen there are many dependencies among the elements of the module but just a single dependency between the modules."

Background: what is a "dependency"?

In software design, dependency means that one element needs to rely on some functionality provided by another element. The elements can be of different kinds:

1) Procedures that depend on other procedures for some computation,
2) Classes that depend on other classes by using them as types of fields, invoking methods, accessing fields,
3) Modules that depend on other modules by accessing some internal elements via the module interface,
4) Applications that depend on certain technologies or other applications by accessing data/functions/exchanging files/etc.

How to best achieve high cohesion and low coupling in the context of the enterprise?"

Align the modules to business services

- Business services are what enterprises provide
- For example, a bank provides
  - Chequing accounts
  - Credit cards

At the enterprise level, one way is to align the modules to business services.

For example...

When we do that... <next slide>

Align the modules to business services

- Modules become *software services*
  - *"A software service is a **coarse-grained**, **discoverable**, and **self-contained** software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model [BJK02]."*
- Goal: "Achieve business-IT alignment" to improve business agility

[BJK02] A. Brown, S. Johnston, and K. Kelly. *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*. Cupertino, CA: Rational Software Corporation. 2002

Go through the definition.

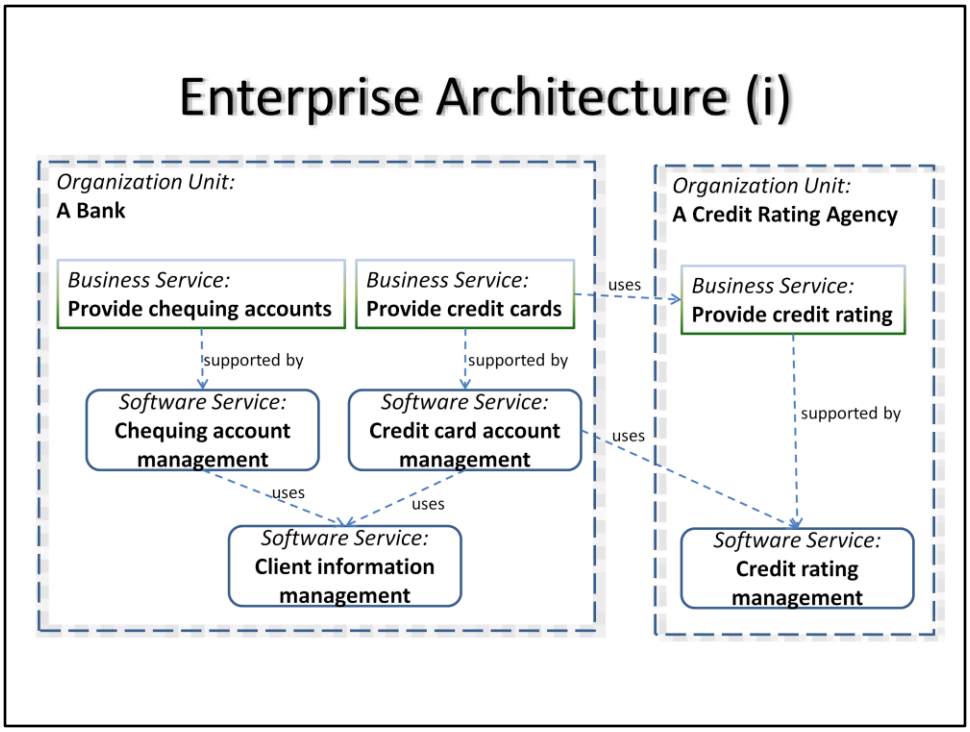
This help achieving an important goal of the enterprises: "improve business agility" that is supported by a strategy "achieve business-IT alignment".

Background:

1) business agility is the ability of the business to quickly react to the ever changing market and take advantage of arising opportunities. Example follows.
2) business-IT alignment is recognized by industry as a key strategy for enabling business agility. Alignment means that there is a clear correspondence between business services and processes and software systems that support them.

Example: a Bank

- Business services of the **bank** are supported by software services:
  - Chequing account management
  - Credit card account management
  - Client information management
- Business and software services of a **credit rating agency** that are used by the bank:
  - Provide credit rating
  - Credit rating management



An enterprise architect can create such a diagram to explain how business and software services of the bank and the credit rating agency interact. The “uses” and “supported by” are dependencies. The dashed boxes represent organization unit boundaries.

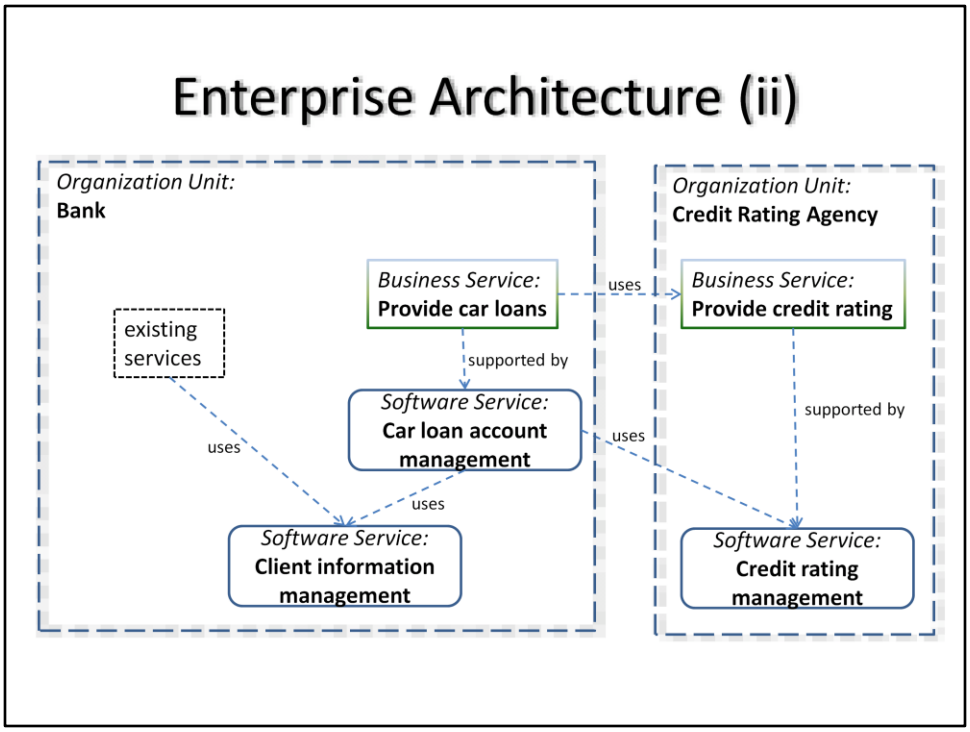
It is a contrived example but is shows the main principle of business services being supported by software services that can have arbitrary structure and dependencies. In this example, since client information management is a shared it has been modularized into a separate software service.

The example also illustrates the idea of SaaS – Software as a Service – the credit agency does not physically sell its software but rather provides a service based on software that banks can access for a fee.

Example: Need for Business Agility

- New market for car loans
- New business service needed
  - Provide car loans
- New software service needed
  - Car loan management
  - ...
- How to quickly respond to the new market?

Let's say that a new market opportunity arises – car loans. How can an enterprise quickly respond to such an opportunity?



The architect evolves the architecture by adding new business and software services and showing their dependencies. As you can see (despite the contrived example), the business can quickly design a new solution, minimize required work by reusing existing services and developing only the truly new functionality. Traditionally, this has not been the case and the duplication of data and similar functionalities was (and still is in legacy systems) causing many problems.

Of course, evolution of such a service-oriented architecture poses other specific challenges but given specific principles it becomes more engineering than art as previously.

Key concepts

- Web services
  - Software services available on the web
  - E.g., credit scoring, credit card payment processing
- Service orchestration
  - Arrangement of services into a process
  - E.g., opening a chequing account
  - Tasks automated by service invocation

Languages and protocols

- Web Service Description Language (WSDL)
  - Defining service interface
- Business Process Modeling Notation (BPMN)
  - Business tasks both manual and automatic
- Business Process Execution Language (BPEL)
  - For execution on a workflow engine
- REST, SOAP, RSS, JSON, ...
- Prefer using models to programming

<This slide could be removed as it introduces more problems than benefits. For example, students were asking for examples of how a protocol or process models look like. That is outside of the scope and causes unnecessary confusion. Indeed the technical space is confusing even to professional software engineers.>

Tool demonstrations

- Two examples of the easiest to use tools for defining and orchestrating web services
  - Yahoo Pipes
  - Tarpipe
- Not enterprise-grade
  - but a useful illustration of the concepts

Pipe & Filter Architecture

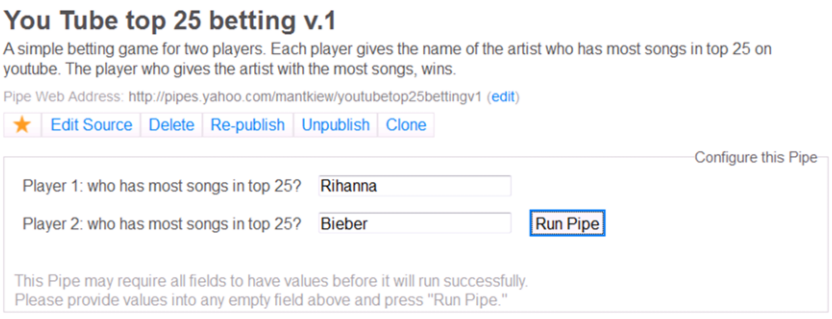
- Data sources
  - Invokes a web service to get data
- Pipe
  - Transfers data
- Filter
  - Processes data
- Data sinks
  - Invokes a service to store results / perform actions

The two tools implement the standard pipe & filter architecture. We have four kinds of elements: ...

Example 1: YouTube top 25 betting v1

- Using Yahoo Pipes
- A game for two players
  - The player who gives the name of the artist who has most videos in the top 25 list wins
- Data source:
  - "You Tube most viewed" web service
  - Artist name provided by each player
- Filters
  - For each player, select items containing the given name and count them
  - Select the player who's name appeared the most

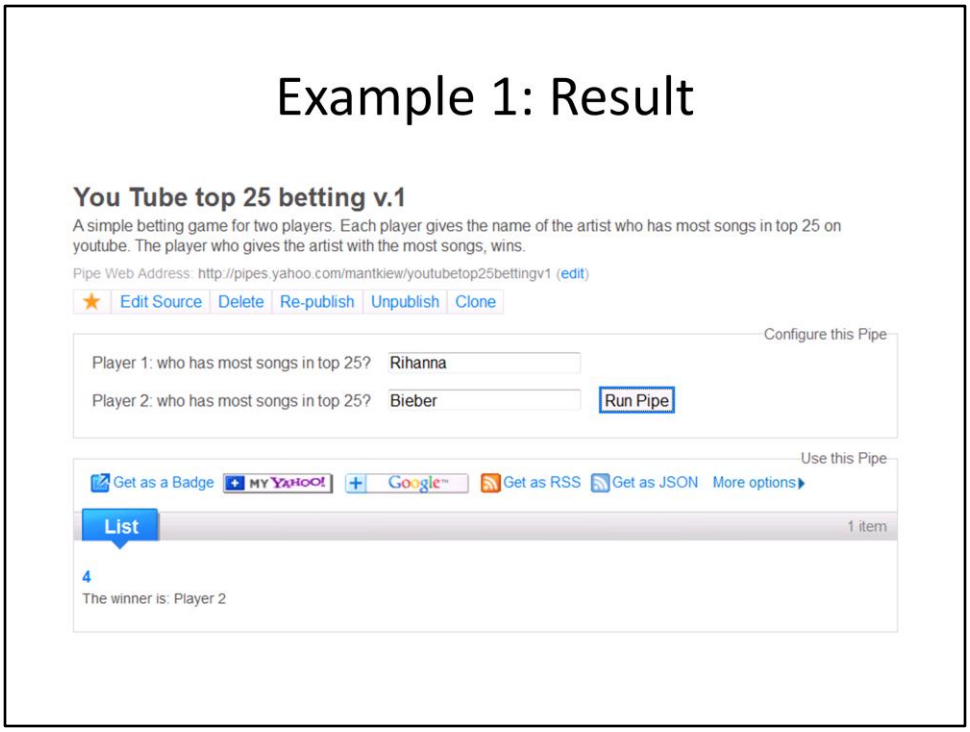
Example 1: Place bets



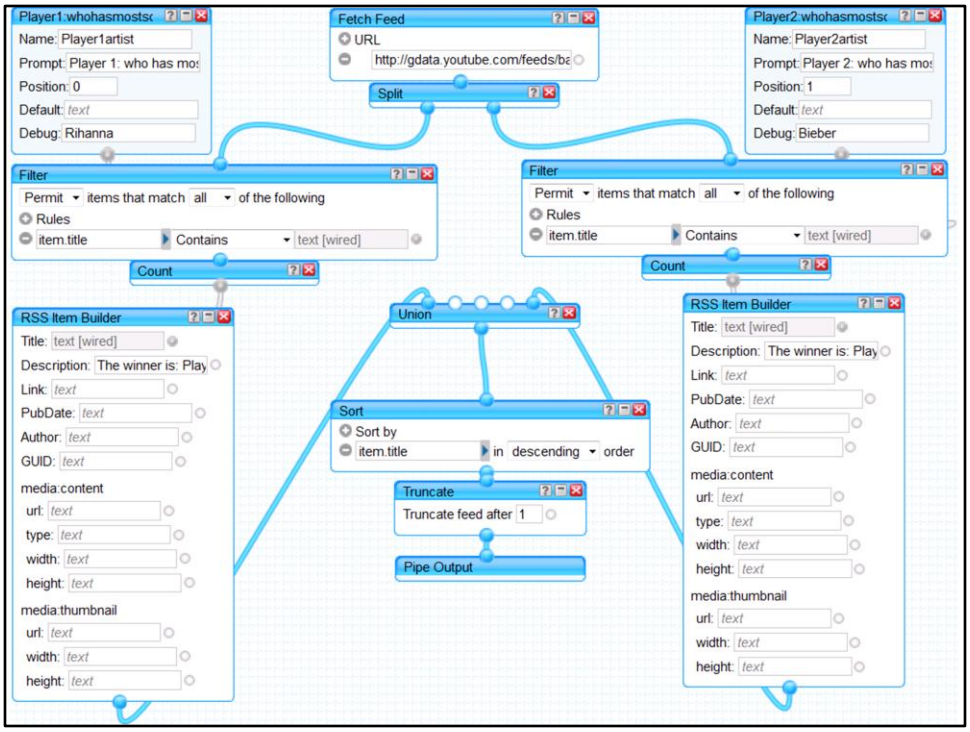
<That's runtime>

The players provide their bets.

Example 1: Result



Results of running the pipe.



Design of the pipe.

Top row we have data sources: text inputs and a fetch feed that calls the top 25 web service.

The filter “Split” creates two identical copies of the data feed.

For each player, the data feed is filtered by permitting only items that contain the provided bet in the ‘item.title’ field.

The filter “Count” outputs a singe number – the number of items from the previous filter.

The filter “RSS item builder” creates a new item with the given description and the count plugged into the field title. <see Title: text [wired] – that means the value of this field comes from a pipe coming from the filter “count”>

The filter “Union” combines the two RSS items for each player into a single feed.

The filter “Sort” sorts the feed by item.title in descending order (the item for the player with the biggest count will be first).

The filter “Truncate” cuts everything except the first item.

Finally, the data sink “Pipe Output” consumes the single item of the player who is the winner.

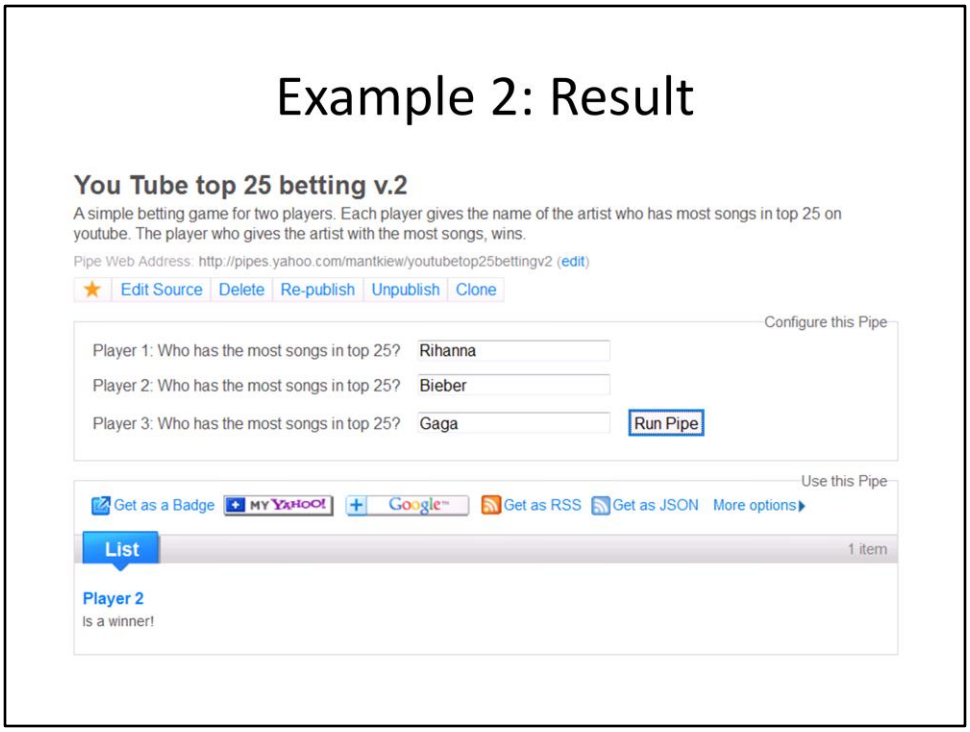
Example 2: YouTube top 25 betting v2

- Using Yahoo Pipes
- A game for three players
  - Factored out a common web service for a single player
  - Used the service three times

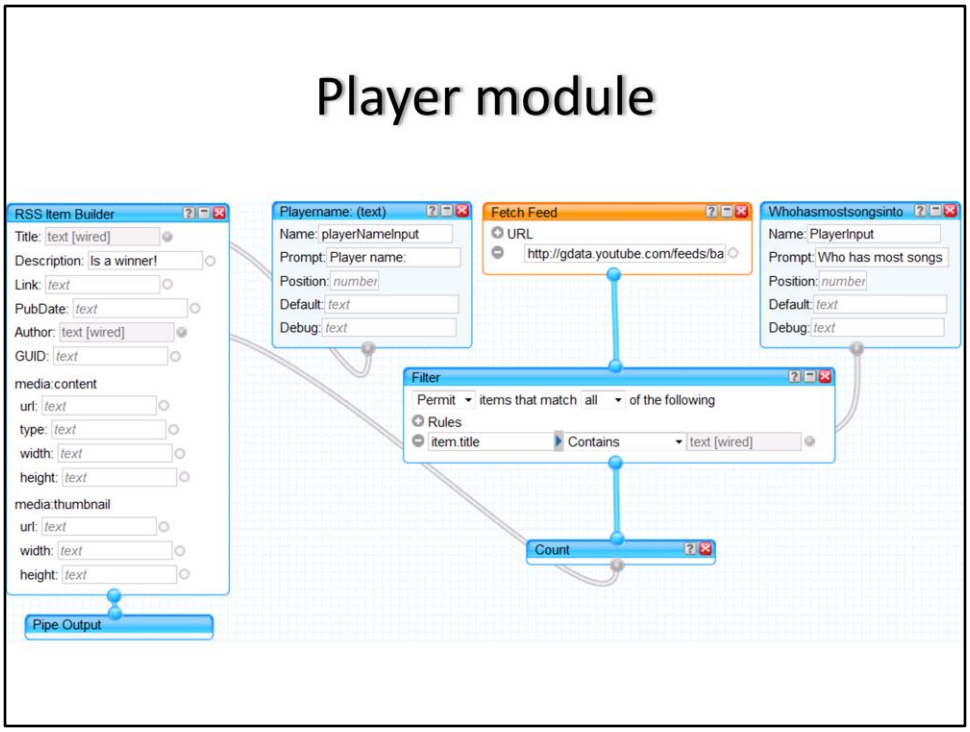
We observe that the part for each player is identical and therefore can be extracted into a separate, reusable module.

It will allow building games with more players without duplicating the code. In this example, we create a game for three players and reuse the module three times.

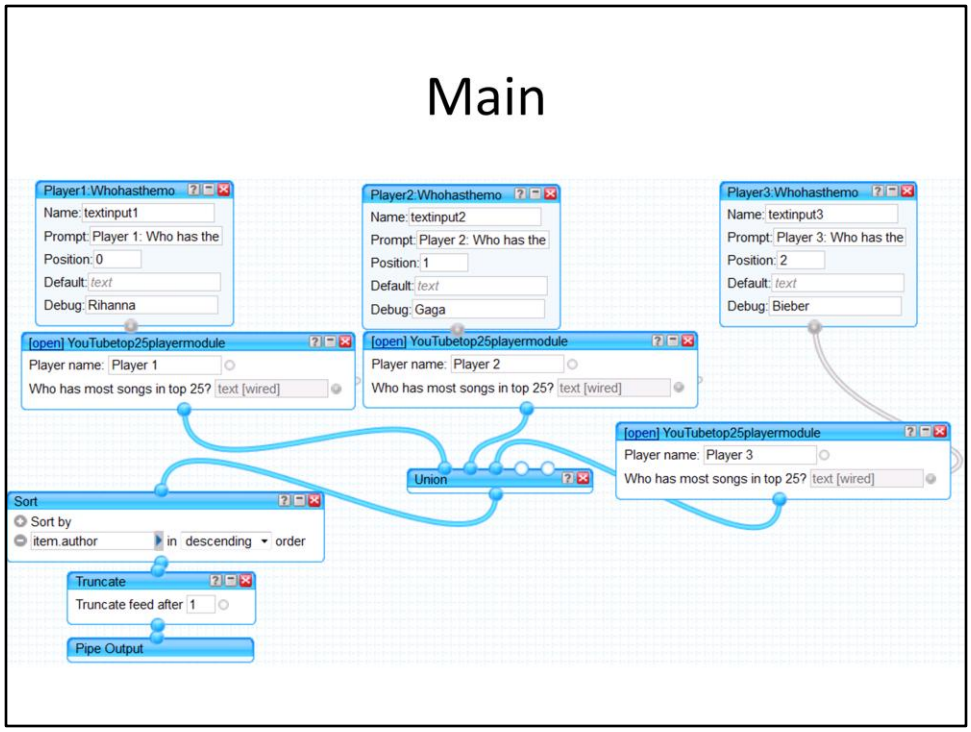
Example 2: Result



<runtime>



This is the extracted Player module. As compared to the previous design, the module has an additional data source: a text input for providing player's name/label.

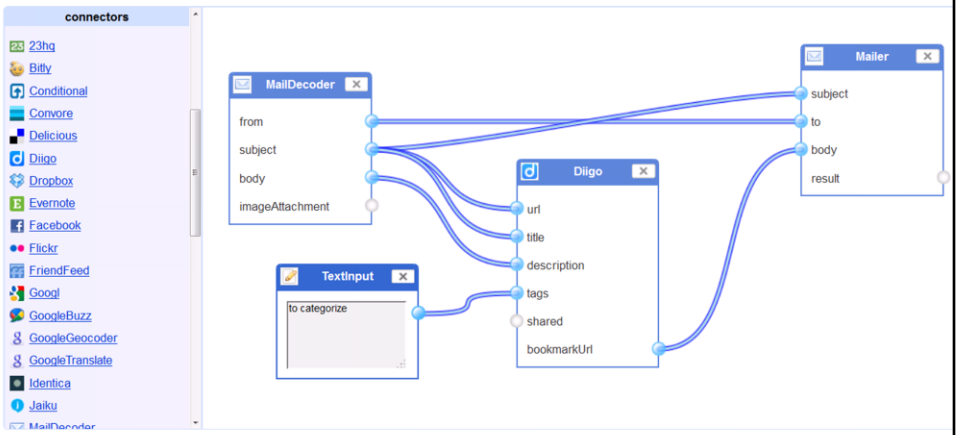


This is the main code for the game. It has three data sources for providing player's bets.

Below, we can see the three invocations of the player module. Each invocation is configured with a player name/label (Player 1, 2, 3). The remainder of the pipe is the same.

Example 3: e-Mail→Diigo

- Using Tarpipe
- Ability to add entries to Diigo by sending email



Another example was done using Tarpipe.

Tarpipe offers a very large number of connectors for most popular web services. In this example, we create a workflow which automatically adds a bookmark in Diigo from information provided in an email.

The data source “Mail decoder” receives an email sent to a predefined address and extracts ‘from’, ‘subject’, and ‘body’ fields.

The filter “Diigo” invokes the Diigo web service and creates a bookmark with the information from the email. The filter then outputs a ‘bookmark Url’.

Finally, the data sink “Mailer” emails back the ‘bookmark Url” as the body of an email to the original sender.

Tarpipe workflows can also be started when another service publishes new information such as “a new RSS item” in a feed.

Example 4: TranslateAnywhere

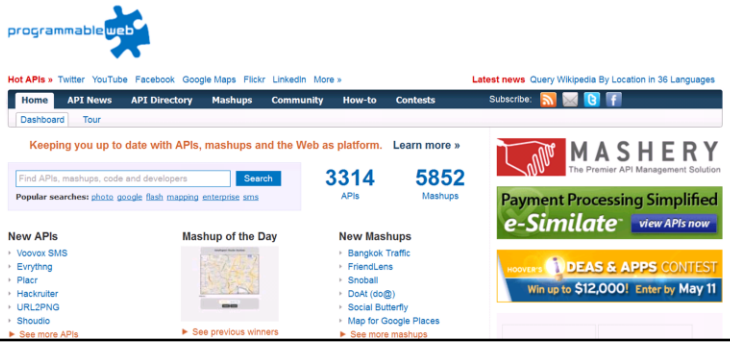
- Using Tarpipe
- An existing 3rd party example of complex web service orchestration:
  - Photograph hand written text
  - Send to tarpipe TranslateAnywhere workflow
  - Text recognition by Evernote's web service
  - Translate by Google Translate web service
  - Receive an email with the translation

This is a very interesting 3rd party example.

<The link is to the YouTube video>

How to find web services?

- In Yahoo Pipes, use *Feed Auto-Discovery* data source
- In general, see
  - http://www.programmableweb.com/



Programmable Web is a tremendous resource featuring thousands of published web services (referred to as APIs (Application Programming Interfaces)) and thousands of web mashups.

Thank You!

Questions?

Comments?

Ideas for nice web mashups?

<Idea: should we perhaps have some homework assignment to create a mashup?>