2009

Generic Web Services Mash-up Integrated Development Environment

Tim Cheeseman, Ngoc Nguyen, and Jordan Osecki

Final Write-up, Independent Study, Professor Regli

Drexel University
9/1/2009

Table of Contents

Overview

The following paper is a report describing a generic web services mash-up integrated development environment. This paper builds on the papers and research conducted by the team members on Semantic Web, OWL-S, Planning, Workflows, and Process Modeling Tools to describe a system application that will serve as our Drexel University Senior Design Project for the Year 2009-2010.

The following paper will describe what a generic mash-up editor will do, common themes, components, architectural design, data feeds, web services, data sources, data representation, data access, service discovery, survey of other related editors, and three case studies using this editor. The rest of this section describes some very basic information about the application.

The goal of this application is to create a portal which is a tool that will let programmers build an application of mashed-up services. The system will have a pallet of services that are clustered by their inputs and outputs. The framework will allow users to play with these services and have a Yahoo! Pipes-like interface to produce elaborate outputs. It would involve service connection and composition (like Yahoo! Pipes offers), data visualization, and writing and editing OWL-S descriptions.

The program result would be an attempt to create an environment for programmers which would make it easier to combine multiple services and have them interact than by scraping different websites and using web services manually, needing to code their integration every single time you are trying to make a new application.

The program would need a registrar (a way to find all services/feeds). It would also need a way to operate on each of the services. Finally, it would need a way to constrain and restrict some of the parameters of the services (and not just their inputs and outputs).

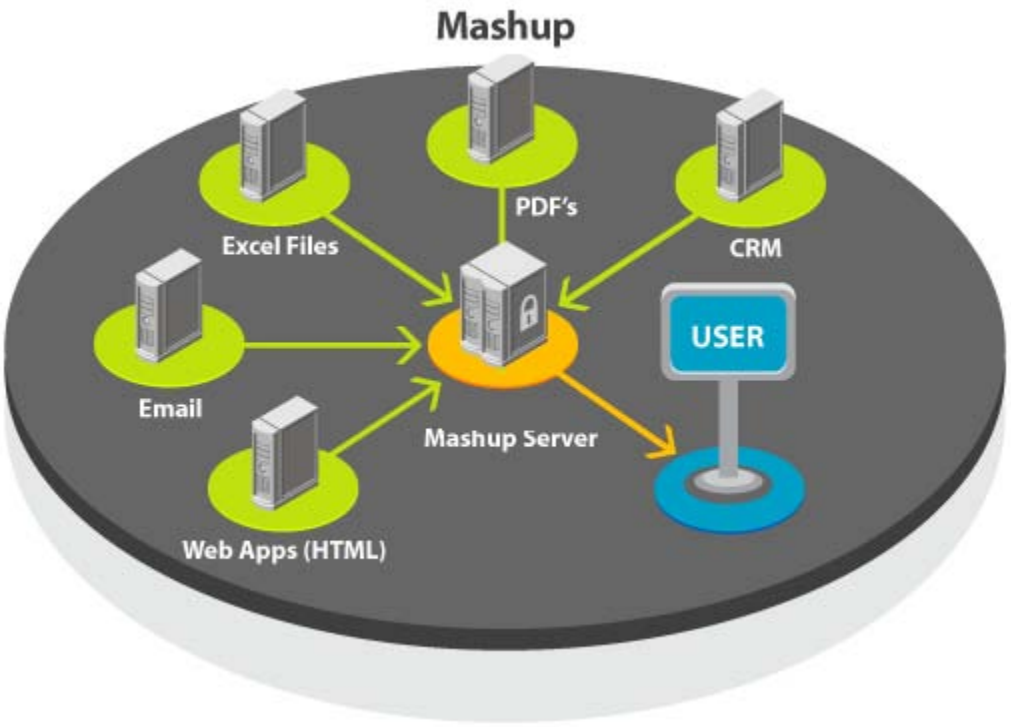
Some example outputs of this environment would be Zillow.com, an application that combines multiple housing services, Spotcrime.com, an application combining crime and housing services, or a weather radar application based on location and other service collaborations.

The program will produce an environment that would support rapid creation of service mash-ups by both programmers and people who don't know about web services alike. The features would need to be awesome, tactile, and visual.

The software would take a type of modality and re-create it, but have housed behind it these services that can be mashed together. There would be a template or palette of web services and structures to connect them. One main problem with this is that there is no notion of data flow, so we would have to figure that out and also have a way to populate a palette of services right away. The concept that the application will need to satisfy is less about visual programming and more like visual database query production.

Survey of Current Development

Below is a diagram that shows a generic mash-up, pulling in sources from web applications, e-mail, excel files, PDF's and more. It is fed to a mash-up server and then presented to the user. This section describes some of the currently developed mash-up editors on the market.



Source: http://www.rareplay.com/uploads/images/RP-WebGlue-Mashup_02.jpg

Current and past efforts toward service mash-ups like Yahoo! Pipes, Google Mash-up Editor, Microsoft Popfly, Intel Mash Maker, and IBM Lotus Mashup Maker fall into two main categories:

- pure editors
- editors with community based mash-up recommendations

The majority of these editors focus on web services, with the aim at assisting experts and novices at creating mash-ups of services that will be accessible via the Internet. This allows users to created service compositions from many different sources and aggregates them on a common interface, such as a map, table, etc. This allows for large amounts of data to be interpreted according to the user's desires, without out laborious manual coding.

However, the web services available to users are limited to web services offered by the mash-up editors the user is implementing their mash-ups on. If the program doesn't expand by offering more web services, they are in trouble. However, another problem is that if they expand by adding more web services, it is harder to sort through them to find the ones that you want to use.

For example, one of the limitations of Yahoo! Pipes is that it does not truly specify which inputs can link to which outputs. Currently, there are many difficulties in finding correct feeds and this problem will only grow more challenging with the ever increase numbers of data feeds and web-services.

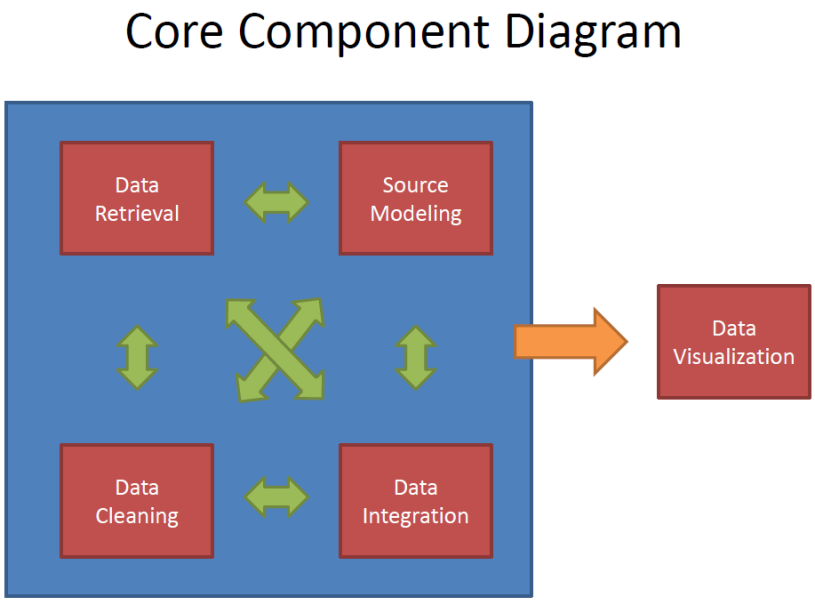
Other limitations are that the user is limited by the pre-define data feeds and web-services offered, limiting the creativity of the user. A code-based editor like Google Mashup Editor does not have such restrictions. Though this is a novel idea for web developments, the over potential for this technology has not fully captured. Perhaps a more concrete example of the mashup used in real world scenario would highlight the real potential of combining services.

Components

Below are the five components required for a generic mash-up editor:

- Data retrieval: Extracting data from web pages into a structured data source.
- Source modeling: Assigning the attribute name for each data column for relationship purposes.
- Data cleaning: Transforming extracted data into an appropriate format.
- Data integration: Combining two or more data sources together.
- Data visualization: Displaying the final data generated by the user.

An intricate interaction between these five components is what will be used to produce a generic web services mash-up editor. Below is a diagram which shows some of the complex interactions that will be needed:



Source: Jordan Osecki

Data retrieval involves the act of extraction from other sources. The next section will describe data feeds versus web services and how to extract from them.

Source modeling involves interpreting the data to determine what it really means. For example, if the data is being brought into a table, it would be determining the column names. If the data is being brought into a map, it is determining the address field and interpreting them.

Data cleaning involves transforming the data uniformly into a consistent content. For example, if you are presented with “Computer Science Course 360” and want to transform it to “CS360”, this is data cleaning.

Data integration involves actually integrating multiple data sources and this is the most complicated part. Yahoo! Pipes does it by allowing users to explicitly match up inputs and outputs and use filters, but this application will attempt to do it more with visual programming.

Data visualization is how to display it at the end, whether it be on a website or in an application. The final product should be highly customizable.

Below is a screenshot from the program Karma, produced by Knoblock. It uses creating mash-ups by example to accomplish these five main components.

Example – Karma, GUI

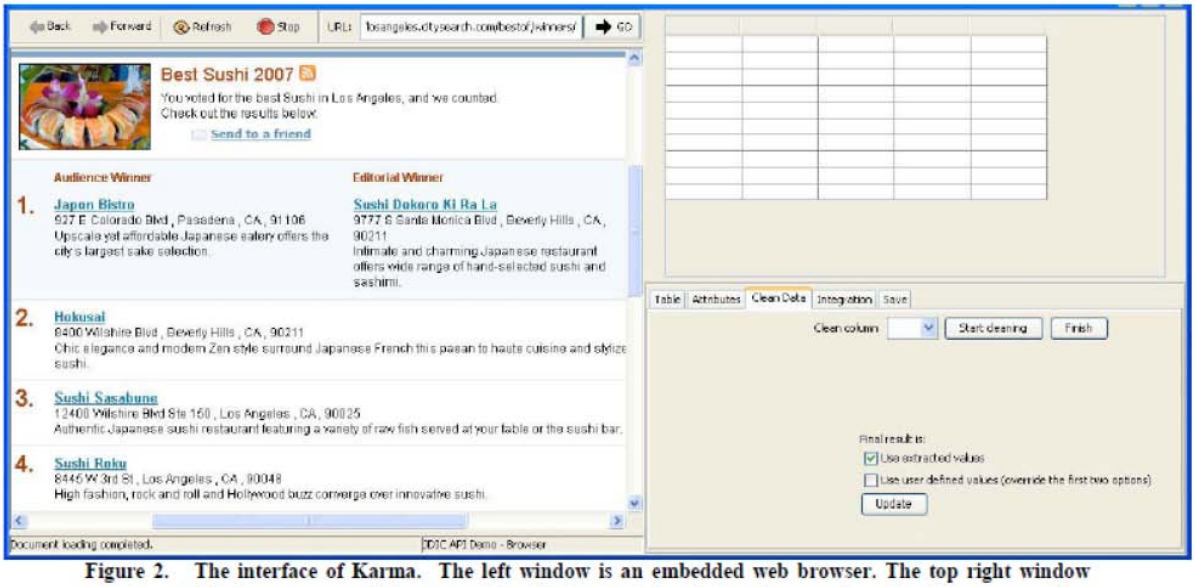


Figure 2. The interface of Karma. The left window is an embedded web browser. The top right window contains a table that a user would interact with. The lower right window shows options that the user can select to get into different modes of operation.

Source: “Building Mash-ups by Example”, by Tuchinda, Szekely, and Knoblock

Data Feeds versus Web Services

Below is a diagram which highlights the principal differences between data feeds and on-demand web services, in the following domains:

- Business
- Product and Technology
- Operations

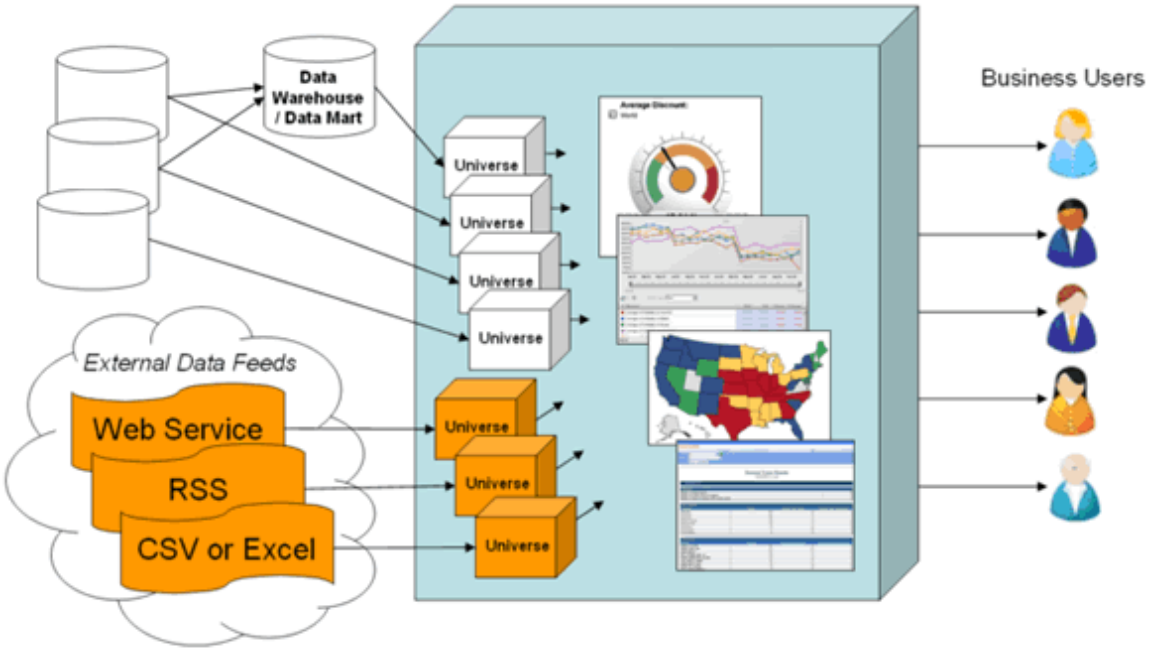
On-Demand Web Services		Data Feeds
pay-as-you-go subscription	**Business**	pay-up-front license
buy online or in-person		buy in-person
low investment risk		high investment risk
minimal vendor lock-in		significant vendor lock-in
reference data	**Product & Technology**	trading and reference data
latency < 1 sec		latency < 1 millisecond
specific data coverage per service		broad data coverage per feed
many service choices		small number of feed choices
individual data transactions		bulk data streams or files
ad-hoc service requests		continuous or scheduled updates
ad-hoc, configurable input variables		fixed, customizable input data formats
ad-hoc, configurable XML output		fixed, customizable output data formats
public, open APIs		private, documented APIs
platform independent		vendor supported platforms
high scalability, e.g., public websites		extreme scalability, e.g., trading floors
hosted online	**Operations**	delivered onsite
online vendor database		in-house database
web service configuration		integration code, middleware and tools
online mash-ups and developer community		In-house development and tools
online account administration		in-house upgrades and maintenance
solid understanding of Web standards		detailed knowledge of vendor product
modest programming skills		high-end programming skills

Source: http://xignite.web-services-blog.com/2009/01/market-data-feeds-vs-web-services-why-buy-the-cow/

The distinction here between a data feed and a web service is the level of dependability required by the system using the data. Data feeds are used by large systems, where low latency is critical and consistent data flow is critical. Data feeds usually require a dedicated team of programmers to write custom data parsers. An example of would be an air traffic control system that requires real-time data of all aircrafts around 100 miles of traffic towers. Web-services on the other hand, are more tailored to systems that use data on a need base demand. Web-services do not require custom parsers to be created, since they typically provide an API for attaining data. So web-services are even on-demand, offering services that reflect real-time data though the performance is typically not as good as data feeds.

Data Sources

Below is a diagram which shows how data source integrate into an application system, with the final data sources being delivered to the business users.



Source: https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/webcontent/uuid/50ea2aa5-9fa7-2b10-649c-a71fa01ab569

Depending on whether the source is a data feed or a web-service, a custom data parsers might have to be created to interact with the data. Any data feed will simply feed raw data into the system, in which then an interface will have to be created for the system to interact with the data. This would require large overhead of resources, developers and experts to understand the data and created real-time domain specific mechanism to convert the data feeds to meet system requirements.

Most web-services, on the other hand, provide an application programming interface (API). This is an interface provided from the vendor of the web-service that provides a mechanism for application programs to requests services. The typical outputs for web-services are XML files that application systems will parse the data from. However, there is a caveat to web-services, there latency is much higher than data feeds, and the provided APIs might not provide or behave according to an application system's demands.

An application called MashupAdvisor aids at assisting mash-up creation through the use of data sources in three easy steps:

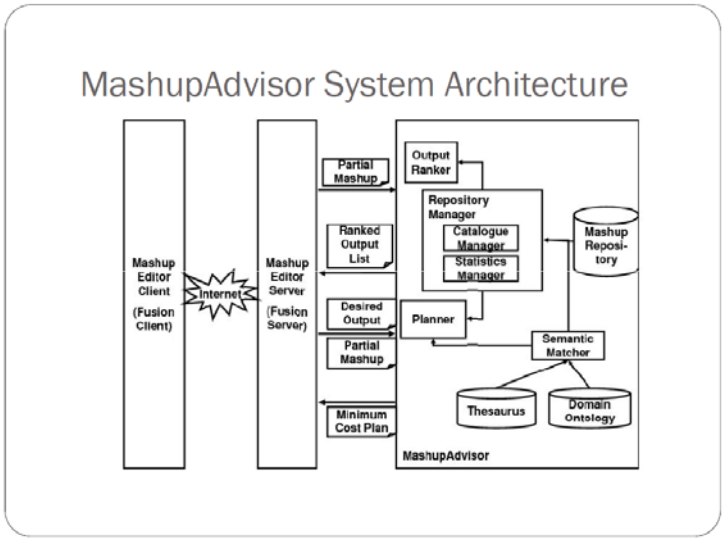
1. Exploits repositories of mash-ups, created by a community of users, to estimate the popularity of specific outputs, and makes suggestions using conditional probability that an output will be included, given the current state of the mash-up
    a. Ex: Given a business news feed

   i. Plot company stock prices
   ii. Map store locations
   iii. Display fonder information/bio
2. User accepts suggestions
3. MashupAdvisor automatically selects and configures the required inputs and adds the required filters, joins and services calls to the mash-up (AI metric planning techniques to find maximum plan)

Some of the benefits of this are the following:

- Shorter development time
- Higher quality plans
- Unthought-of-yet-interesting outputs

Below is a diagram which shows the architecture of the MashupAdvisor.



Source: http://www2.computer.org/portal/web/csdl/doi/10.1109/ICWS.2008.128

Data Representation

Each of these services will be described using OWL-S and WSDL, using the following files:

- WSService.owl – The overview service file
- WSProfile.owl – The OWL-S Profile file
- WSProcess.owl – The OWL-S Process file, describing the service composition
- WSGrounding.owl – The OWL-S Grounding file, describing how the inputs and outputs will be linked to the WSDL
- WSGrounding.wsdl – The web services WSDL description

Below is an example of a composite process encoded in OWL-S.

```
<process:CompositeProcess rdf:ID="Design">
        <rdfs:label> This is the top level process for CIBER-U Design </rdfs:label>
        <process:composedOf>
                <process:Split>
                        <process:components rdf:parseType="Collection">
                                <process:AtomicProcess rdf:about="#CAD_Format"/>
                                <process:AtomicProcess rdf:about="#CAD_Assembly_Format"/>
                                <process:AtomicProcess rdf:about="#Material_Model_Format"/>
                                <process:AtomicProcess rdf:about="#Vertex_Model_Data"/>
                                <process:AtomicProcess rdf:about="#Mesh_Format"/>
                        </process:components>
                </process:Sequence>
        </process:composedOf>
</process:CompositeProcess>
```

The data is represented for composition using the OWL-S. It describes the inputs and outputs, pre-conditions and post-conditions. The services will be represented with these models and interpreted by our editor to represent them in some way with visual programming, making it more intuitive than simply linking inputs to outputs.

Data Access and Service Discovery

The data used by the mash-up program will be directly accessed from its source, whether it is an existing web service or raw data feed of some sort. The program will have access to an existing repository of pre-defined, semantically modeled sources that can be used and mashed up to provide the dataset for a website component.

Of course since the web is always changing and more and more services and data feeds come into existence each day, it would be impossible to have an exhaustive repository of all of the web's information, all semantically modeled. Users will have the ability to model a data source by hand in OWL-S if need be, but this is extremely tedious and error-prone work, and defeats the purpose of a mash-up IDE for non-programmers.

The program will provide for automatic service identification and will be able to generate semantic models for many "unknown" data sources, using its knowledge of previously modeled data sources to identify newly discovered ones. We can use the many data sources that have been modeled by hand or otherwise as a reference, leveraging on the fact that different sources will very often provide similar or overlapping data.

For example, assume the program knows about three semantically modeled sources. Source 1 takes as input a zip code and returns the latitude and longitude of its centroid. Source 2 takes as input two pairs of latitude and longitude and returns the great circle distance between the two locations. Source 3 takes in a distance in kilometers and converts it to miles. Because the program knows what inputs these sources take, and it knows not only what outputs they have, but what these outputs represent, it can attempt to use them in identifying a newly discovered source.

Assume a user adds the URL for a Source 4 to the program. The program can determine that Source 4 takes as input two zip codes and returns some distance. By trying different combinations of its know sources, the program can come up with a composite that matches the inputs and outputs of Source 4. One such composite is using Source 1 to find the centroids of the two zip codes, and Source 2 to find the great circle distance between them. The program can then compare the output of the composite against the output of Source 4, and if they are "sufficiently similar," then Source 4 can be identified and modeled as a web service that finds the great circle distance in kilometers between the centroids of two zip codes. If not, the program can try other combinations (for example converting the distance to miles with Source 3), hopefully finding a match before exhausting all combinations.

This same method can be used not only for automatically identifying and modeling new sources, but also for automatically generating composite mash-up components given the desired inputs and outputs by the user.

Process Modeling

The processes will be modeled using the OWL-S process model, as shown in an example above. The model can represent processes that are both atomic and composite, which is a combination of atomic processes.

Below is a diagram which shows a sketch of a process model for the website Amazon.com.

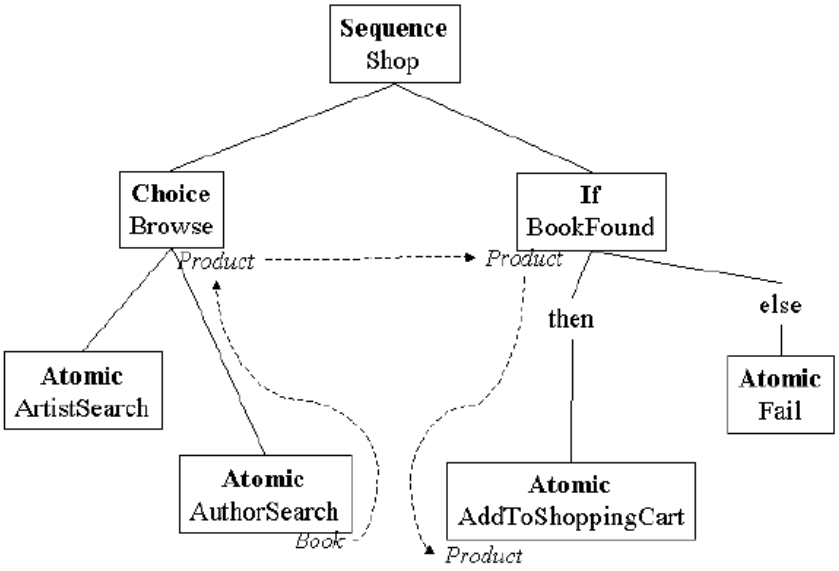
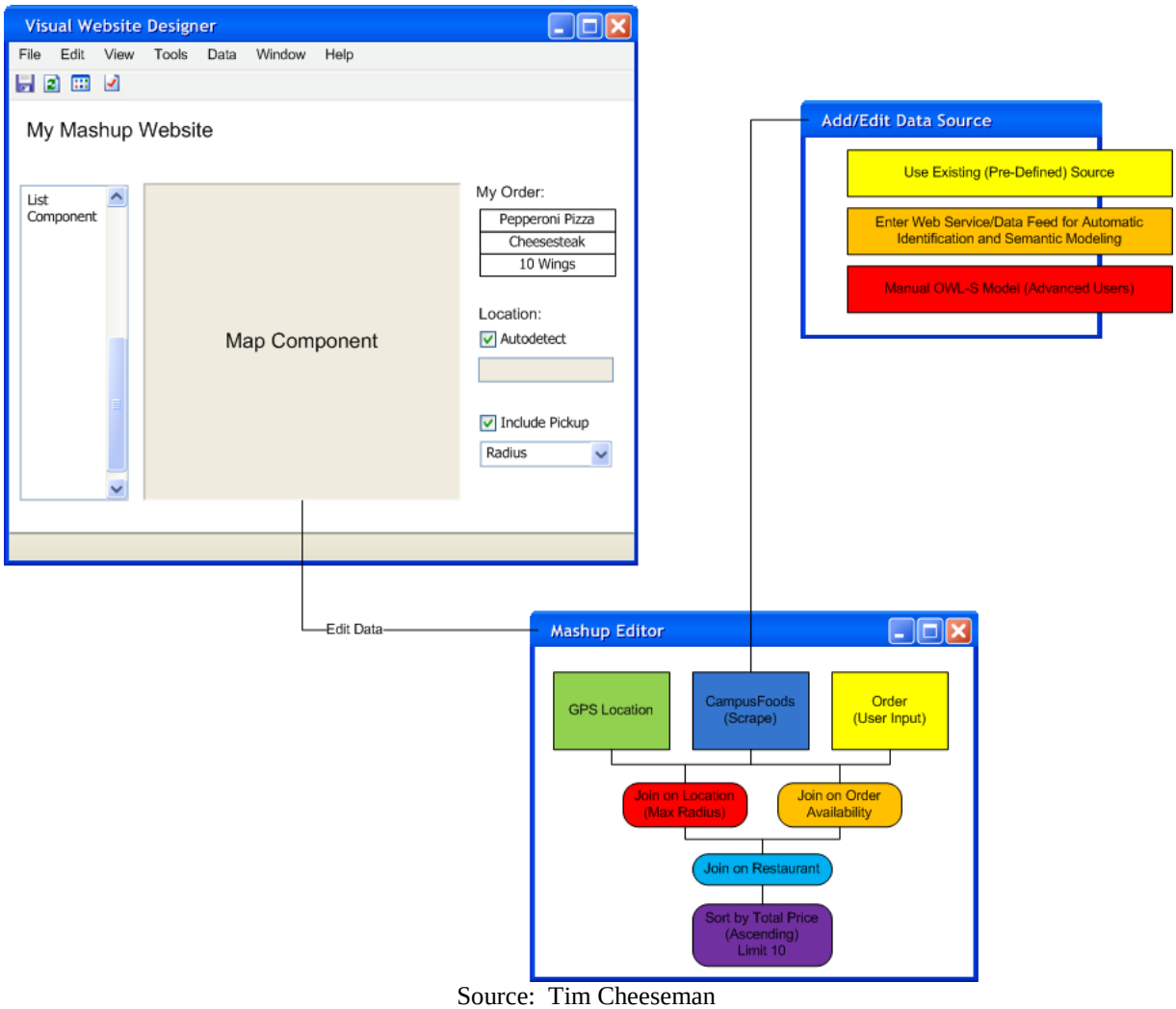


Fig. 1. The Process Model of Amazon.com's Web service

Source: http://www.ai.sri.com/SWS2004/final-versions/SWS2004-Ankolekar-Final.pdf

High-Level Design

The architecture of the system, from a high level, will be that of a WYSIWYG visual website designer, allowing the user the ability to drag and drop both static components (such as HTML components) and dynamic AJAX components that plug into the Data Masher back-end. The diagram below shows a mock user interface for the visual website designer.



Source: Tim Cheeseman

What separates this product from something like Adobe Dreamweaver, for example, is the Data Masher framework for plugging the data from mash-ups into the dynamic components. Users will be able to simply right-click a component (in this example, a dynamic map), and directly edit the data that feeds into it. The user can choose data sources (shown as rectangles) and mash them together based on certain constraints and restrictions (shown as rounded rectangles).

The user can also add or edit the data sources themselves, being able to choose from a pre-modeled repository of sources, automatic service identification as described in the Data Access and Service Discovery sections, or for the advanced user, hand-modeling the source themselves in OWL-S.

Case Study – Make Money Renting to Drexel Kids

Below is a case study that could be created in our generic web services mash-up editor on "Making Money by Renting to Drexel Students". See below for the web services involved and a narrative describing how they would be composed.

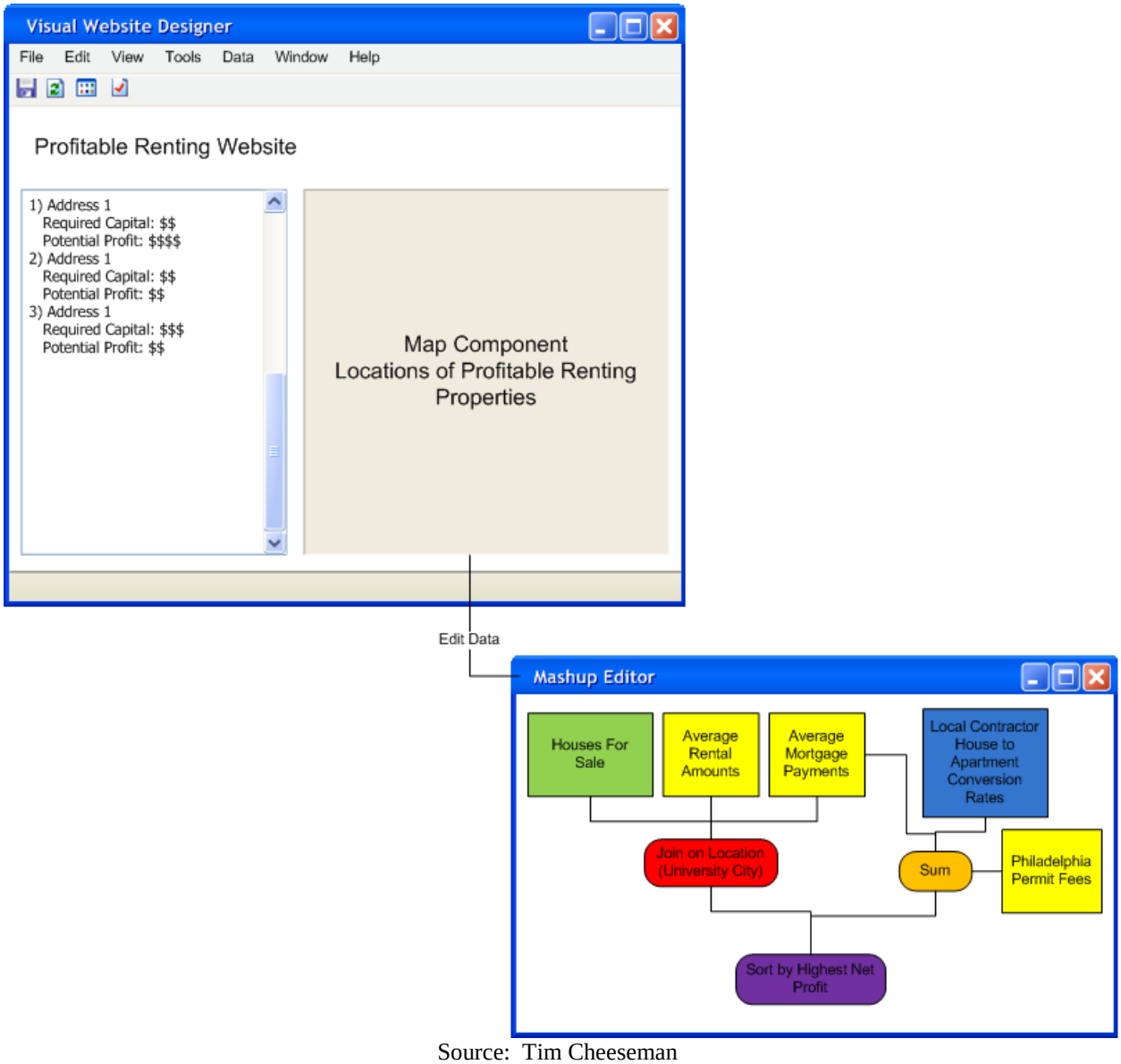
Web Services:

- Current Houses For Sale in University City
  - Example: http://www.zillow.com/homes/map/university-city,-philadelphia,-pa_rb/
- Area House-to-Apartment Conversion Rates by Contractor
  - Example: http://www.respond.com/handyman-services/PA/Philadelphia/yellowpages.html
- Fees Charged by the City of Philadelphia for Permits
  - Example: http://www.phila.gov/LI/ContentPage.asp?TopNode=services&level1=56&level2=&level3=
- Current Average Rental Amounts listed on Craigslist aimed at Drexel Students, by Location
  - Example: http://philadelphia.craigslist.org/search/apa?query=drexel&catAbbreviation=apa&minAsk=min&maxAsk=max&bedrooms=
- Current Mortgage Loan Payments in University City by Price
  - Example: http://realestate.yahoo.com/Pennsylvania/Philadelphia/loans/mortgage.html

Narrative:

- This service will use the current houses for sale near Drexel, the amount of money that may be necessary for permits and apartment conversion, the amount of rent that can be achieved by them based on current supply and their locations, and current average mortgage rates being given out, to determine which houses would be the most profitable to buy with a mortgage, convert if necessary, and rent to Drexel students

(Figure on the next page)



Source: Tim Cheeseman

Case Study – Happy Hour Crimes

Below is a case study that could be created in our generic web services mash-up editor on "Happy Hour Crimes". See below for the web services involved and a narrative describing how they would be composed.

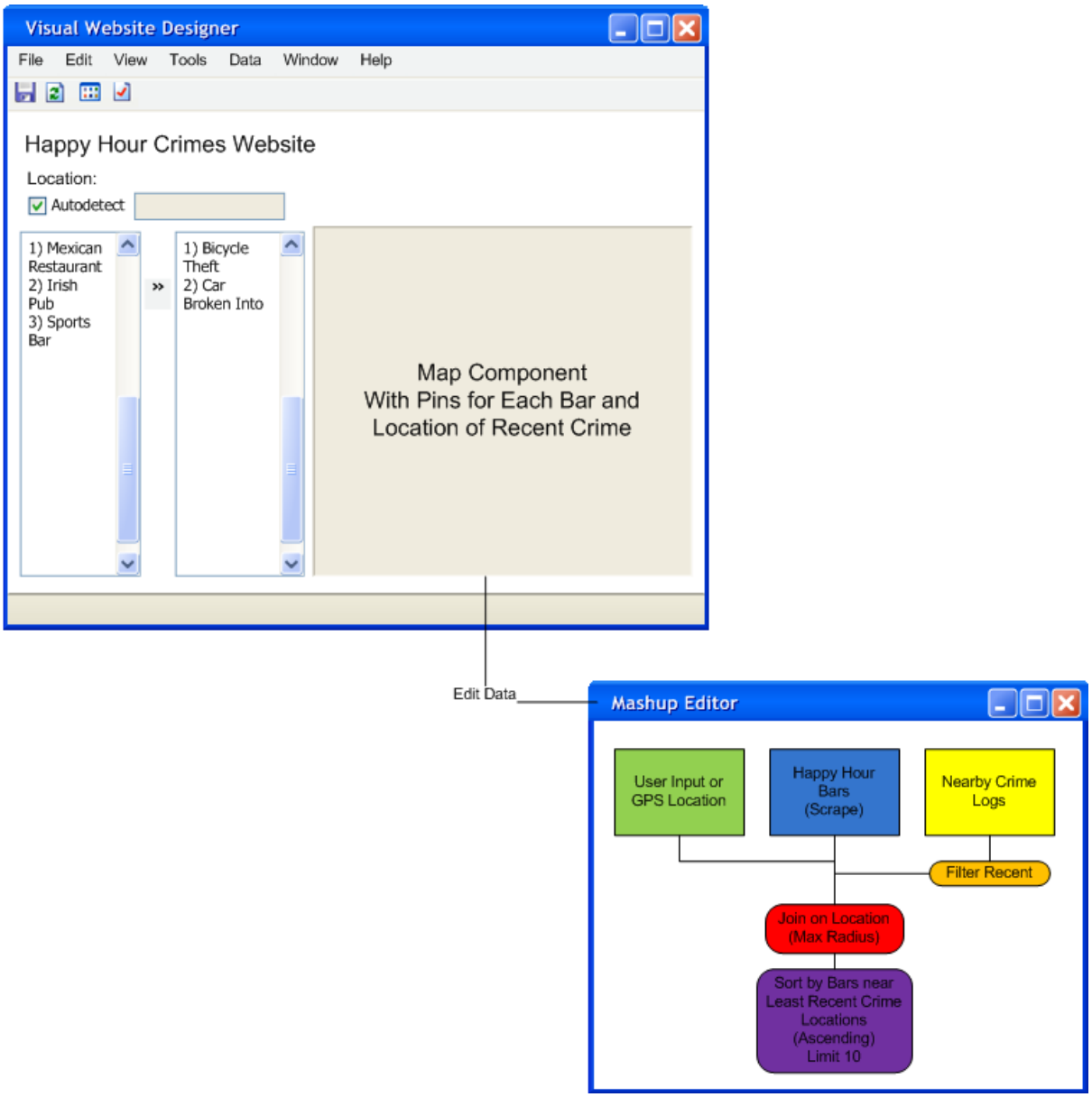
Web Services:

- List of bars with happen hours information (Scraper)
  - Example: - http://www.yelp.com/list/best-happy-hour-philadelphia-2
- Philadelphia crime logs (RSS)

  - Example: http://spotcrime.com/pa/philadelphia
- Google Map
  - Example: -http://code.google.com/apis/maps/

Narrative:

- Given all the bars within the city of Philadelphia and their known happy hours. Match this data up with minor crimes such as vandalism, public indecencies, gambling, speeding, etc. within 5 city blocks of bar. Plot this data on a map for users. This allows users to take clients, relatives, etc. to bars that are more likely to have fewer nearby crimes.



Source: Tim Cheeseman

Case Study – Fast, Cheap Food

Below is a case study that could be created in our generic web services mash-up editor on "Fast, Cheap Food". See below for the web services involved and a narrative describing how they would be composed.

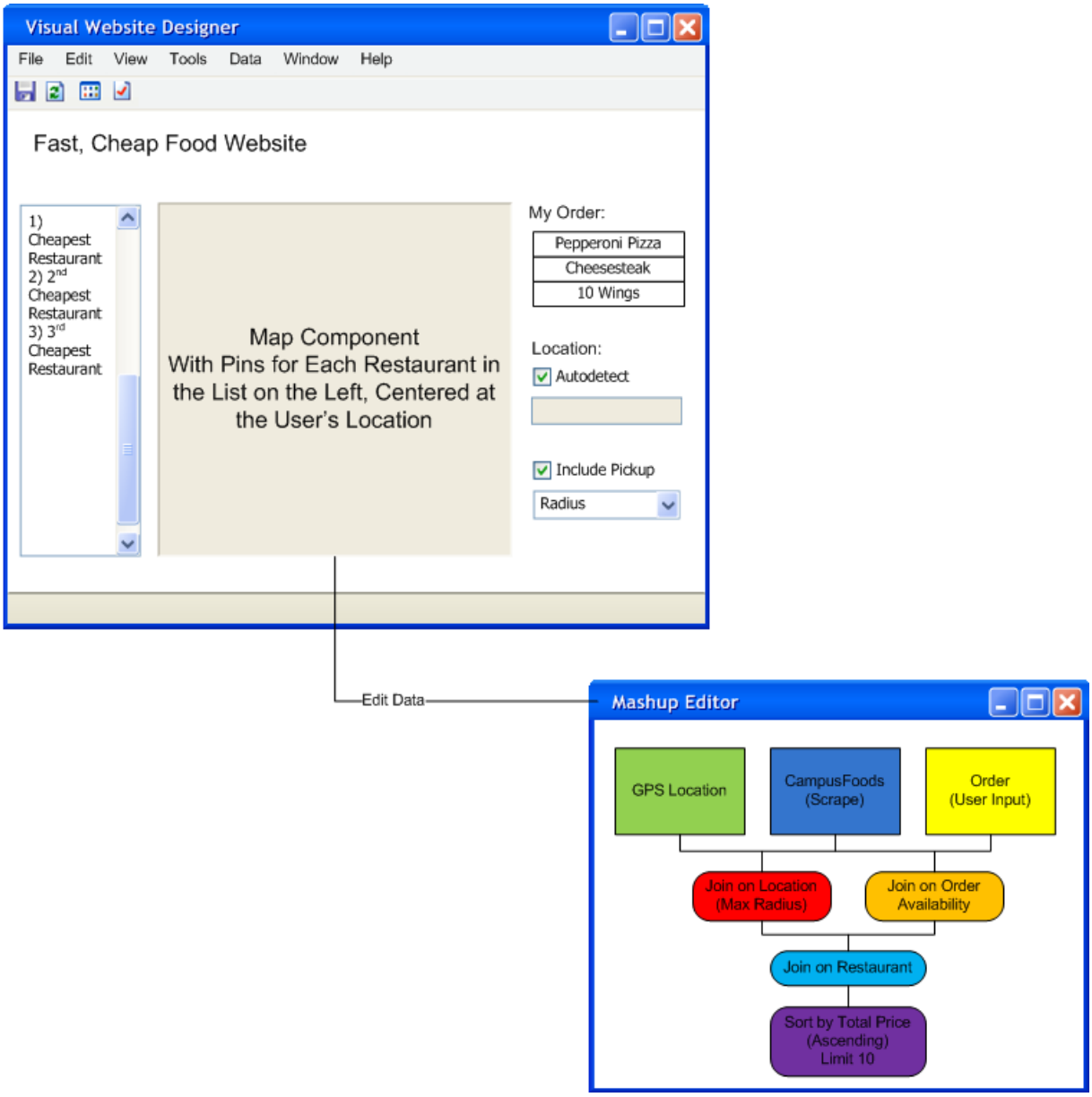
Web Services:

- Location
  - Example: - HTML 5 Location Framework
- Nearby Restaurants for Delivery and Take-Out
  - Example: - CampusFood.com
- Review Website
  - Example: - Yelp.com

Narrative:

The service will get the user's location from the browser (GPS on a smartphone, IP lookup on a laptop, etc.), and look up restaurants within a certain distance radius that deliver (or pickup if within a smaller radius). The user will also specify a desired order, and the service will find the restaurants that can fulfill that order for the lowest price (including specials, service and delivery charges, etc.) The service could also automatically up-vote the cheapest/closest restaurant on a review website, giving stock review about it being a great deal.

(Figure on the next page)



Source: Tim Cheeseman

Conclusion

The main obstacles currently facing mash-up editing are the limited components available for users to create their mash-ups. The burden falls heavily on the user and the mash-up communities of developers to create web-services. Therefore, there exists a need to automate the creation and quantification of new web-services to help speed up this development of mash-up creations. Also, since mash-up creation is a relatively new approach to software development,

there has not been a large following or a lot of support. There is still a need for more development to show the public at large the capabilities of mash-up editors.

The goal of this application is to create a portal which is a tool that will let programmers build an application of mashed-up services. The system will have a pallet of services that are clustered by their inputs and outputs. The framework will allow users to play with these services and have a Yahoo! Pipes-like interface to produce elaborate outputs. It would involve service connection and composition (like Yahoo! Pipes offers), data visualization, and writing and editing OWL-S descriptions.

The program result would be an attempt to create an environment for programmers (and non-programmers alike) which would make it easier to combine multiple services and have them interact than by scraping different websites and using web services manually, needing to code their integration every single time you are trying to make a new application. The result is an integrated development environment that anyone can use to mash together sources/services.

References

Hazem Elmeleegy, Anca Ivan, Rama Akkiraju, Richard Goodwin, "Mash-up Advisor: A Recommendation Tool for Mash-up Development," *Web Services, IEEE International Conference on*, pp. 337-344, 2008 IEEE International Conference on Web Services, 2008.

Mark James Carman and Craig A. Knoblock. Learning semantic descriptions of web information sources, In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI), 2007.

Tuchinda, R., Szekely, P., and Knoblock, C. A. (2008). Building mash-ups by example. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 139-148, New York, NY, USA. ACM.

* All graphics and their accompanied commentary are cited by the source tag below the graphic.

* Some of the content was derived from discussions during the AI-Planning Independent Study.