A framework for dynamic data source identification and orchestration on the Web

Alexander Berezovskiy
University of Southampton
University Road, Highfield
Southampton, SO17 1BJ, United Kingdom
+44 (0)7775 675360
letoosh@letoosh.com

Dr Leslie Carr
University of Southampton
University Road, Highfield
Southampton, SO17 1BJ, United Kingdom
+44 (0)23 8059 4479
lac@ecs.soton.ac.uk

ABSTRACT
The current Web offers a very large number of solutiofns and services, ranging from social networking and content delivery services to business applications and management systems. However, in a general case the solutions provided are largely disintegrated, with each product operating in it's own environment. Additionally, many of the products are unknown to the end user and finding the most suitable application is commonly a non-trivial task. The current project is an effort to provide a ubiquitous interface for Web application integration. The suggested approach allows for dynamic identification of the applications most suitable for a given task and access to their data using a unified interface in the REST architectural style. A novel algorithm for identification of the most appropriate data source is introduced within the study. Evaluation of the overall system and the obtained results is provided.

Categories and Subject Descriptors
H.3.3, H.3.5 [**Information Storage and Retrieval**]: Information Search and Retrieval – *retrieval models, search process, selection process;* Online Information Services – *data sharing, web-based services.*

General Terms: Algorithms, Management, Design.

Keywords: Information retrieval, web integration, unified data access, data source identification.

1. INTRODUCTION
It can be said that there has been an immense increase in the use of social applications in the past several years and people accept online communities as a part of their everyday lives. Additionally, businesses tend to pay more attention to the ways in which online social and management tools can be used [1]. Therefore, many operations involved in processing and manipulation of people's social, professional and personal information are moving onto Web platforms. Whilst this can be seen as a ubiquitous source for data retrieval and processing, it presents new challenges to the ways in which information can be retrieved, composed and manipulated. In a general case, web applications are not interconnected and every product operates in its own environment. With the absence of a flexible and scalable framework, which could be used to interact with any web resource, both developers and end users are forced to switch between applications and interfaces in order to fully benefit from their services. Therefore, the current project is an effort to provide such framework and ubiquitous environment for integration on the Web.

1.1 Definitions
For the purpose of the current paper and in order to avoid ambiguity the following definitions need to be introduced within the study:

• "*Application*" is defined as "a software product designed to help users perform a task"

• "*Service*" is defined as "a set of functionality provided by an application aimed at solving one or many related tasks"

Therefore, Web applications are accessible via a Web browser and the methods used and applied to such applications present a set of common technologies involved in the development, deployment and exploitation of the applications. In the context of the current paper, Web services represent a set of features provided by a Web application.

1.2 Project goals
With the rapid growth of the Web and the advent of associated technologies, it is now possible for Web applications to interact within the context of provided services. However, the methods currently available put a number of limitations on the scope of possible interaction. Therefore, it can be beneficial for the future development of the Web to construct a theoretical and practical framework, which can be used to uniformly interact with any Web application. The current project is an effort to create such framework and provide a ubiquitous environment for dynamic Web integration. More specifically the project intends to focus on:
- Dynamic identification of the most suitable service providers (applications),
- Provision of a universal interface to access and manipulate the providers' data.

The project aims to provide an interface for both developers and ordinary Web users to manipulate the data composed from disparate web sources on the basis of provided services.

2. ANALYSIS

The rapid development of the Web and related technologies have reflected the need for a scalable data integration platform in the new emerging products. Examples of such products are the ProgrammableWeb project, the OpenID technology and the OpenSocial platform. Although many of the products are dedicated to the mashup method and are unsuitable for large-scale data integration [2], some technologies such as OpenID and OpenSocial provide a more general approach to the problem.

The OpenID initiative suggests a technology to universally authenticate against a web application using existing credentials on some other application [3]. For example, if a user has an account on Google, this can be used as their credentials to log into ProgrammableWeb. Since OpenID specifies a protocol for authentication and is not tied to any specific product, it can be used as a way to enable communication between two applications in the means of identifying a user [4]. Although, the OpenID standard does not provide a framework for full data composition and is limited to authentication routines, it demonstrates the potential for large-scale integration on the Web and the requirement for a universally accessible data. It is stated that OpenID is gaining wide adoption and the number of users is estimated at 500 millions with more than 48,000 OpenID providers [5].

Another popular project aimed at web integration is the OpenSocial platform advocated for by Google. OpenSocial provides a set of APIs to access data of its partner applications. It aims at integration at both functional and data levels, and is focused on social networking platforms. For the purpose of the current study and in order to identify major design issues it is suggested to look at the details of the platform implementation.

2.1 Case Study: OpenSocial

The OpenSocial platform aims at universal integration of social networking systems using an approach similar to mashup creation. It utilizes the general concepts of social networking data in an attempt to abstract away from implementation and provide a generic interface to access and manipulate the data [6]. Operations are performed at the level of data entities existing within social networking systems and the relationships between them. The entities include user profiles, their activities, media items (photos, videos or other similar content), messages and more [7]. A unified JavaScript framework is provided to compose data into functional elements, which can be combined into a single page in a form similar to mashup construction and executed at client side. Each of such elements is treated as a separate “gadget”, providing some functionality and can be used separately. Thus, users are free to create their own gadgets and embed them into custom Web pages [8]. Apart from this, information can be gathered from multiple data providers, which can be dynamically integrated into the platform. Each of the providers must implement a specific interface in order for their data to be available within the platform.

In general, OpenSocial provides an extensive and rich framework for integration of social networking sites. The outside interface is independent of the underlying data provider and the platform is capable of manipulation on the data in a universal bi- directional manner. However, the system is targeted at social networking resources which limits the scope of application and significantly reduces the flexibility of the system [9]. Additionally, existing applications can not be immediately integrated into the system unless additional modifications are made in order to provide the required interface. Finally, it can be said that the platform lacks serious management and integration routines required for large-scale adoption, such as flexible and unique data addressing and interaction between gadgets [10, 11].

3. DESIGN

3.1 Overview

Due to the nature of the target solution, the overall project can be divided into two major tasks: data source identification and data retrieval. Since the system has to interact with numerous Web resources it is implemented and deployed online as a Web application by itself. The project takes a user-centric approach to the design and implementation. It is implied that users tend to use different Web applications for their needs depending on their location, age, background and other [12]. Therefore, data for different users may be spread across different applications. However, the project does not intend to limit the scope of available data to those attached to a given user. Indeed, any web page can be treated as a web application in the context of the system since the very existence of the page implies the existence of the data available within. The general functional sequence of the system can be defined in the following order. On receiving a data request, the system has to determine which applications to use. Next, a call can be made to the selected application in order to perform the requested action on the data (either retrieve or push information to the resource). Depending on the result of the call and nature of the data request, the system returns the data or reports on the results (Figure 1).

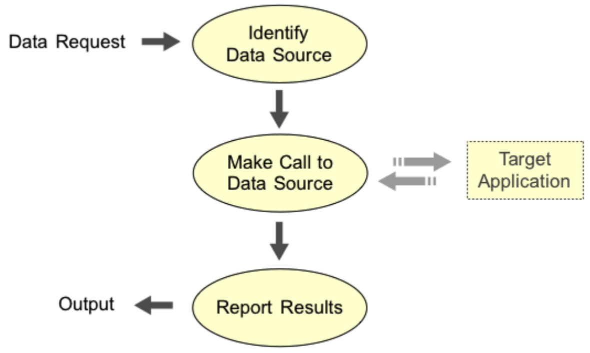

Figure 1: General operation

In order to uniquely address the type of data required, the project follows a service-oriented architecture, by devising common data patterns for similar applications. For instance, most social networks provide user profile data, which typically contains profile picture, name and associated activity or interests data. The devised patterns are then organized in hierarchical service structures. An example of a devised structure common for many applications on the Web is demonstrated in Figure 2.

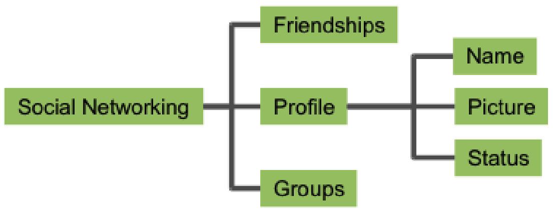

Figure 2: Data tree example

Thus, every application can be assigned one or many services, each corresponding to some data entity within the application (for example, Facebook provides "Social Networking / Profile / Status" service and related data). There is, however, no intention to predefine a fixed tree of services to be used within the system. Instead, the system aims to provide functionality to modify the set of available services at any time in order to enhance flexibility and accommodate new applications.

Dynamic application discovery within the vast amounts of web resources is commonly a non-trivial task and involves multiple data mining techniques, which are usually either computationally expensive or resource exhaustive or both. For the purpose of the current study, application discovery on the Web is not the target objective and outside of the scope of the project. Instead, following the general concept of social interaction on the Web, the project aims to provide features for end users to add new applications to the system. However, the project aims to provide capabilities to identify the most appropriate data resource for a given service within the set of registered applications.

The next two sections provide a detailed overview of the system architecture. It is assumed that all data requests received by the system are addressed to a particular service, so that the nature of the expected result is known.

3.2 Data source identification

As it has been shown in 3.1, people tend to use different applications depending on various parameters, such as their location, background and other. Therefore, in order to compute the most appropriate application for a given service, the system needs to keep track of such parameters and the applications people use for various tasks. Whilst it is apparent that a user's country of residence and language spoken affects their choice of application, the effect of the other parameters is uncertain and requires extensive testing and user survey. During the system design stage it has been chosen to dynamically build a set of user parameters in order to enhance flexibility and effectiveness of the identification process.

This way, users are free to provide their details based on the set of available parameters and corresponding options. Each set of details (parameter-option pairs) is then combined into a single data entity corresponding to a user's personal data environment (thereafter referred to as "environment"). Instead of directly assigning user selections to his or her profile, they are assigned to an environment instance, which in turn gets assigned to the user profile. It is expected that every unique set of selections corresponds to a single environment instance. Thus, for example, two users who share the same country and language, and have not provided any other information get the same environment instance assigned as a set of their personal details. This approach allows for more efficient computation of similar users and their preferred applications as it does not directly depend on the number of users, but utilizes only their details (environments) and preferences (usage records). A user may then explicitly choose the applications they use and, if desired, the services used within the applications. Due to hierarchical service model, it can be implied that if a user makes use of some service on a particular application, he or she also uses the child (contained) services within the given one, unless specified otherwise. For example, if the system knows the current user has their social networking profile on LinkedIn, it can be assumed that the user's profile picture, name and similar data is also contained within LinkedIn.

As a result, the system would need to consider all the usage records of all users to effectively compute the most appropriate application. However, this makes the computation dependent on the number of users, whilst the actual value is the number of users who use an application in a given environment. Therefore, a separate usage statistics for every application may be built in order to avoid this limitation. The general design of the identification part of the system is shown in Figure 3.

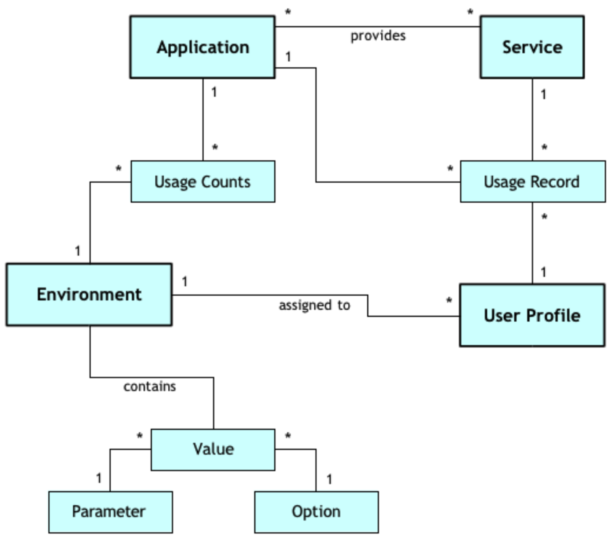

Figure 3: Architecture - Data source identification

3.3 Data retrieval

When a data request is received, the system initially tries to identify the most appropriate resource to handle the request. When the resource is identified, the system needs to contact the target application and perform the requested operation. However, every application provides a different Application Programming Interface (API) and no common operation can be suitable for all of them. There is usually no semantics provided with APIs, so automatic discovery of provided interfaces can not be achieved. Moreover, many applications do not even provide an API in any form, hence direct data access may not be possible. Therefore, the system has to be flexible enough in order to accommodate any web resource and allow for granular data retrieval without the need to communicate with an API. In order to achieve such level of flexibility, the system architecture provides a framework to dynamically define ways in which an application can be interacted with. Instead of communicating with an application directly, the system passes the request over to a small utility (thereafter referred to as "binding"), which interacts with the required resource and returns the result of the performed operation back to the system. Every binding is a small software and can be written by any user, in a manner similar to application registration. The system provides a number of standard utilities to simplify the process of binding creation and interact with third-party web resources, their APIs or in the form of web scraping. In order to ensure security of the underlying architecture, bindings are executed in a secure environment, where access is restricted to the routines required for request handling. Additionally, extra adjustments need to be made to ensure bindings do not contain any malicious code, which may be used to intercept users personal information, authentication details or similar information. To provide a functionality to prevent such situations the system implements an additional feature for bindings to be reviewed by administrators. By default, every new or modified binding is put into a "non-approved" state. The system administrators may then review the binding code and either mark it as "safe" or "non-safe". Thus, only the author of a non-approved binding can trigger its execution prior to review, which may be useful for development and testing purposes. However, some users may still wish to use bindings which have not been reviewed. The system, hence, allows to approve a binding for a user's personal use. However, bindings that have been marked as non-safe are never executed independently of personal approval, unless the author of the binding changes its code in which case the binding becomes a non-reviewed again.

Finally, target applications may require additional input in order to process certain requests. Examples of such requests include authorization into an application, user identification (username or user unique ID may be required), uploading a new profile picture (new file required), bookmarking a web page using social bookmarking systems (an URL is required) and other similar tasks. Therefore, the system provides a "Data Interface" entity to maintain sets of required and optional parameters available to pass over to the target application. Bindings, however, are not required to specify an interface in order to operate. Instead, the functionality is provided as a support feature to help enhance interaction between the end user and the target application (data source). Thus, users will be notified if there is any missing parameters that are required by the binding. Therefore, binding developers may ensure that all the required data is provided by a user at the moment of execution and does not cause unpredicted behavior of the target application.

The suggested approach provides very high level of flexibility as it allows to interact with any web resource in a unique way, specific to the resource. At the same time, it enhances the system functionality by allowing custom interfaces to be defined by binding developers. When combined with the identification subsystem, the overall architecture is capable of dynamically identifying user's data within the integrated applications.The overall system conceptual design is shown in Figure 4.

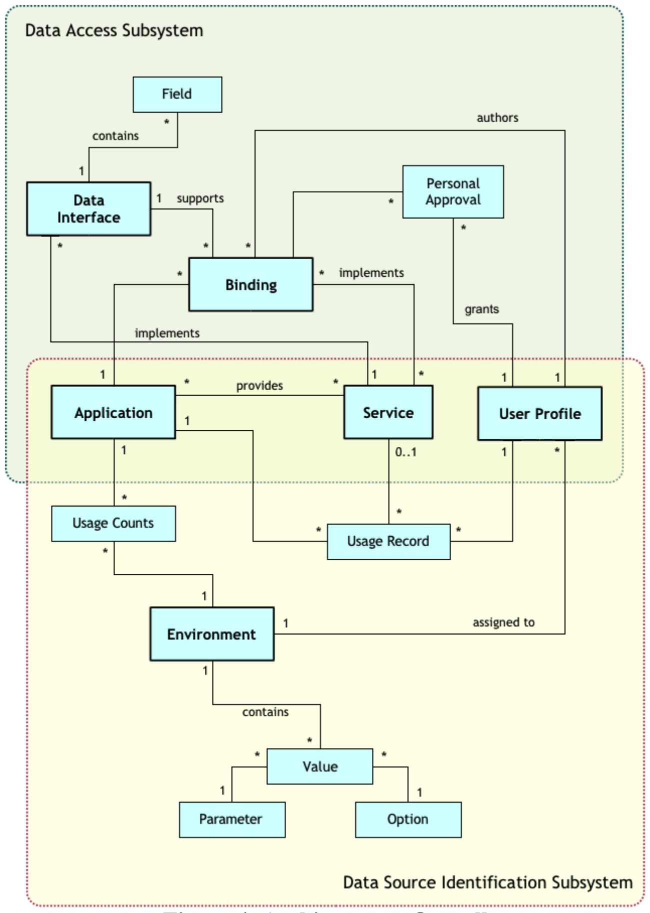

Figure 4: Architecture - Overall

3.4 System components

In general, due to the nature of the project, the system must be capable to operate at high loads and hence, allow for multiple bindings to be executed at the same time. Thus, the project implements a distributed structure of functional elements in order to achieve this goal. Initially, the elements described in 4.2 and 4.3 are contained within the main operating server (thereafter referred to as "Server") and the actual binding execution is handled by a separate execution controller (thereafter referred to as "Controller"). Server is capable of communication with multiple Controllers at the same time in the common distributed objects architectural style. Every Controller, in turn contains one or many secure execution environments (thereafter referred to as "Runtime"), which processes binding code for every request. Each Runtime provides a safe environment where untrusted code, such as bindings can be executed. Requests to third-party applications are performed from within the environment. Thus, should a binding fail to complete a request in a reasonable time interval, it will not affect the behavior of the rest of the system. Apart from this, bindings may need to store some data within the system (for example secure tokens to identify itself for the third-party application). Such data may only be accessed by binding developers. At the same time users may want to save their input within the system, to avoid entering the required parameters on every data request. Such user data saved within the system must only be available at the time of execution. Since Controller is in charge of handling binding execution, it is logical to construct persistent storage (thereafter referred to as “Storage”) within its own architecture instead of main Server. Finally, in order to further enhance multiprocessing capabilities of the whole system, each of the three major functional elements (Server, Controller, Runtime) can have multiple instances (Figure 5).

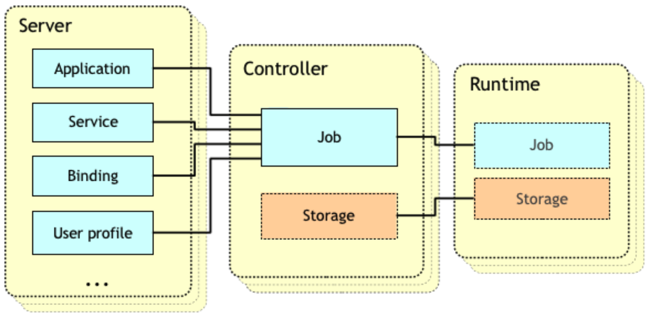

Figure 5: System components

4. IMPLEMENTATION

4.1 Identification algorithm

The data source identification subsystem is one of the major components of the project and was the first functional element to be delivered during the implementation phase. The algorithm for the task was developed incrementally based on the evaluation of achieved results. The final version of the algorithm is described in the current paper.

The starting point for the computation of suitability score of an application is its relevance to the requested service. Thus, an application that provides neither the service itself nor any of the parent services should receive a much lower score than an application that is more suitable for the task. Apart from this, the suitability computation can be seen as a general recommendation problem. Therefore a number of common techniques, such as collaborative filtering may be applied [13]. In user-based collaborative filtering items are recommended based on their similarity to the items currently used by a user. For example, if a user **A** is known to use application **1**, and there are other two users **B** and **C** who are known to use application **1** but also use application **2**, then **A** is likely to use **2**. Raw processing of all user preferences is usually computationally expensive with a large number of users, although the algorithm can be modified to precompute item similarity offline and use the resulted counts within the algorithm [14]. This method allows to efficiently predict the applications that a user may use, but is not sufficient by itself, since it requires some critical mass of users prior to effective operation [15] and initial intervention from the end user to mark used applications.

As demonstrated in 3.2, the major part of the algorithm needs to consider user personal details, such as the country of residence and language. It is possible then to compare users with matching details (same country, language or other parameters in their environments) and compute similarity based on the used applications. The system, however, implements a separate data entity (Usage Counts) for maintaining the usage numbers per application for a given environment. Therefore, the algorithm only needs these numbers and there is no need to analyze every user. Additionally, it is required to produce effective results prior to reaching a reasonable number of users. Thus, it is possible to predefine usage numbers based on the statistical data of freely available online surveys. Additionally, some applications provide information on their usage demographics. Such data was collected and aggregated in the beginning of the development phase in order to provide initial statistical counts for the search algorithm. At the same time, it can be said that the importance of the statistical data is lower of that of actual users of the system. Additionally, data obtained from registered users of the system is of higher value than the data gathered from those who have used the system either implicitly or directly, but have not actually registered for an account. Therefore, the difference between the value of each dataset is taken into account within the algorithm by defining a weight for each element.

The general form of the resulting algorithm:

Total Score for an Application a is defined as:

$TSA(a) = TRS(a) + ERS(a) + TRAA(a)$, where:

- $TRS(a)$ - Total Relevance Score for a
- $ERS(a)$ - Environment Relevance Score for a
- $TRAA(a)$ - Total Recommendation based on Application-to-Application score for a

Total Relevance Score $TRS(a)$ is expanded as:

$TRS(a) = AMS(a)\omega_{ams} + SMS(a)\omega_{sms}$, where:

- $AMS(a)$ - Application Match Score for a . This element is calculated differently depending on the request. When requesting a data operation on a given service AMS is always zero. When the user searches for an application using plain-text search this estimates to the number of matches within an Application title and description.
- $SMS(a)$ - Service Match Score for a , defined as:

  1 for every matching service on the upper part of the branch of the required service up to the top parent service multiplied by the distance from the required service, or:

  $$SMS(a) = \sum_{service}^{n} \frac{1}{distance+1} ,$$

Where $n = |ParentsOnBranch(required)|$ and

$service \in ParentsOnBranch(required) \cup ProvidedServices(a)$

- ω_{ams} and ω_{sms} are pre-defined weights for AMS and SMS respectively.

Environment Relevance Score $ERS(a)$ is expanded as:

$$ERS(a) = \sum_{e \in E(a)}^{n} \left[EMS(e, Env(u)) \frac{EAUC(a, e)}{ETUC(e)} \omega_{euc} \right]$$

where:

- $E(a)$ - set of environments where a is used
- $Env(u)$ - environment instance of the current user
- $EMS(e, Env(u))$ - Environment Match Score for e and $Env(u)$. This corresponds to the number of matching (identical) values between the two environment instances

- $EAUC(a,e)$ - Environment-Application Usage Count for a in e, or the number of users in the environment using the application, according to pre-defined statistical data
- $ETUC(e)$ - Environment Total Usage Count for e, or the total number of users in the environment, according to pre-defined statistical data
- ω_{euc} is a pre-defined wieight for Environment Relevance Score.

Total Recommendation based on Application-to-Application score $TRAA(a)$ is expanded as:

$$TRAA = \frac{\sum_{b=A(u)}^{n} RAA(a,b)}{\sum_{c=AR(a)}^{n} RAA(a,c)} \omega_{raa}$$, where:

- $A(u)$ - set of applications the user u is known to use
- $AR(a)$ - set of applications used along with a
- $RAA(a,b)$ - Application-to-Application score for a and b (precomputed)
- ω_{raa} - pre-defined weight for $TRAA$

Therefore, the full expanded form of the resulting algorithm can be written as:

$$TSA(a) = AMS(a)\omega_{ams} + SMS(a)\omega_{sms} +$$
$$+ \sum_{e \in E(a)}^{n} \left[EMS(e, Env(u)) \frac{EAC(a,e)}{ETC(e)} \omega_{esc} \right] +$$
$$+ \frac{\sum_{b=A(u)}^{n} RAA(a,b)}{\sum_{c=AR(a)}^{n} RAA(a,c)} \omega_{raa}$$

It can be said that the suggested algorithm provides a quite efficient method as its computational complexity does not depend on the number of users active within the system and the algorithm is capable of operation without initial mass of active users. Additionally, it allows to tune the importance of its components, hence providing a way to balance the results towards more dense datasets. Thus, for example, when the system reaches a reasonably high number of users, the statistical component may be completely withdrawn. Finally, it considers two dimensions of datasets by utilizing both user environment data and application usage data. It is stated that multidimensional algorithms are generally more effective than unidimensional and provide more accurate results [16].

Apart from the statistical counts that are predefined within the system, an automatic discovery of partial user data was implemented. On the first visit, the system attempts to detect the visitor's country based on their IP address. Apart from country, the system attempts to detect the user's language based on the Accept-Language header sent by the client's browser. The header is a part of HTTP standard and normally contains a number of preferred languages in a standardized format, which usually correspond to the language of the used browser software. In the meantime, in the context of the algorithm, all applications represent user preferences. Thus, both local and online applications may be treated equally, though differ by type. Therefore, the choice of web browser and operating system may reflect personal preferences of a user. The system attempts to automatically detect the operating system and browser in use based on the standard HTTP User-Agent header sent by the browser, which commonly includes information on the underlying operating system. Such applications are registered within the system as ordinary entities, but placed into a separate category, excluded from the set of recommendations devised by the algorithm.

As a result, two environmental parameters (country and language) and two preferred applications (browser and operating system) are detected automatically to create the required minimal dataset. The algorithm is, therefore able to provide initial results without any intervention required from the end user.

4.2 Data interface

The data access interface was implemented in the REST (Representational State Transfer) architectural style. Contrast to traditional integration technologies (such as WSDL and UDDI), REST relies on the standard HTTP protocol, does not require a separate resource discovery service, and is easier do implement, find and invoke [17]. Additionally, in the context of the current project, where it must be allowed for a large number of independent queries to be performed at the same time, REST represents a more suitable choice than traditional methods of Web integration [18]. Finally, following the general approach of the project geared towards high flexibility, REST represents a scalable architecture, which may be easily extended for use in a very large number of usage scenarios [19, 20].

In the general form the interface can be written as:

```
http://example.com/service/<service_path>/data[.extension]
  [/id=user_id][/app=application_id][[/bind=binding_id]
  [?binding_input_arguments]
```

Where all parameters in `<>` are required and all parameters in `[]` are optional, and:

- `service_path` – hierarchical path to the required service, for example: `social-networking/profile/name`
- `extension` – extension of the required data format (optional).

  For example, `html`, `xml`, `text` or `jpg`.
- `user_id` – unique ID number of a project user (optional).

  This may be used to make cross-user calls for data. If not specified the current user is used. If a foreign user profile is specified access must be granted by the foreign user to the current one.
- `application_id` – unique ID number of an application (optional).

  Can be used to specify a different application to the one suggested by the identification algorithm.
- `binding_id` – unique ID number of a binding (optional).

  Allows to specify a different binding to the default one.

- binding_input_arguments – a set of key-value pairs to pass to the binding.

  Uses the standard form of key=value[&key=value][...]

  If the request does not use the GET method, the set of input arguments is derived from a composition of arguments provided in the URL and the body of the request, where preference is given to the latter.

Each of the optional arguments may be specified independently of other arguments. The system will pass all the specified parameters to the target binding.

The system response to a data request depends on the result of the operation and the input parameters, such as expected format. The example of an XML response to a request to /service/social-networking/profile/name/data.xml is shown in Figure 6.

```
<data result="success" code="1">
  <title>Data / Name / Profile / Social networking</title>
    <info>
      <application id="1" name="Facebook"
                   uri="http://example.com/app/view/1/"/>
      <binding id="8" name="FacebookName"
        uri="http://example.com/binding/view/8/" />
      <interface id="53"
        name="Social networking / Profile / Name: FacebookName"
        uri="http://example.com/service/social-
             networking/profile/name/interface/app=1/bind=8" />
      <format type="data" name="XML" extension="xml"
        mimename="text/xml" />
      <service id="5" name="Name"
        treepath="social-networking/profile/name/"
        uri="/service/social-networking/profile/name/" />
    </info>
  <result>Alex Berezovsky</result>
</data>
```

Figure 6: XML data response example

4.3 Performance

Although the final system demonstrated good performance results during stress testing period, a large area for improvement was seen in the application of caching framework to the system.

Caching was implemented for a large proportion of internal functionality of the system. Apart from standard per user page cache, the major improvements were concerned with the identification algorithm. Thus, the environment match counts (EMS) and application-to-application similarity scores (AMS) were cached, followed by caching of the major algorithm components. The system was adjusted to drop individual cache elements on user actions which may affect the results of the algorithm (for example, on change of user environment). This allowed to dramatically decrease the number of queries to the database required in order to process a request. Generally speaking, due to the granular approach to the caching subsystem, most requests required no queries to be made to the database.

The implementation of the caching framework proved to significantly increase response rates of the system and overall stability. The final stress testing demonstrated a 30% increase in the amount of requests processed in a time interval and over 20% increase in stability for the price of only 9% increase in memory usage.

During the final stages of implementation, when the project matured to a state of fully functional system, it was decided to undertake a more intensive testing by opening the project to the public. In early March 2010 the project was deployed in the form of an public preview, set up and released. The users of the site were asked to provide feedback and comments on their experience.

This testing phase allowed to gather information about the suitability of the implemented system for wide public use. The feedback received was used to make further adjustments to the project code, which mainly involved usability and user interface improvements.

Additionally, with the increase in number of users, the implemented algorithm demonstrated improved precision due to the growing data set. This demonstrated the efficiency of the suggested approach for the problem domain.

Finally, the attention drawn to the project at the public preview stage demonstrated that the overall system presents a novel solution to web application integration and may have further implications on the common practice of web development.

5. CONCLUSION

The current project is an effort to provide a flexible and scalable solution to the problem of large-scale application integration on the Web. It can be said that the resulting system meets the devised set of requirements for such solution and presents a novel approach to the problem. Contrast to the existing products, it allows for granular and flexible access to application data, independently of the internal implementation details. In the meantime, the suggested approach does not limit the scope of data resources available for integration and does not require any adjustments to be made within the applications. In fact, any data available to a user on the Web can be integrated and accessed using the system. Broadly speaking, the domain of applicability of the system is not limited to Web resources, but can be extended to any data entity accessible via the Internet. Finally, the identification algorithm of the project allows to dynamically compute the most appropriate applications for a user. As a side effect the results produced by the algorithm can be used in many other areas, and the algorithm itself may be applicable in a different field of similar problem domain.

In conclusion, it can be said that the overall project presents a significant contribution to the research area of dynamic data integration and service orchestration on the Web. A further study of the area may reveal new intriguing challenges which may not yet be predicted.

6. DISCUSSION

Whilst the current project provides a way to identify the most appropriate data sources for a given service, it relies on the database of applications manually entered into the system. A rather interesting research challenge would be to achieve automatic discovery of such applications on the Web. An easy way to achieve this would be the use of emerging Semantic Web technologies. However, at the moment of writing there is very little number of applications on the Web that supply semantic data related to the provided services. Although, there is no common approach to achieve application discovery at present, a deeper study of the problem may substantially benefit the current project and the overall research area in general.

Apart from this, the capability to extract granular chunks of data from the Web provided by the system may be enhanced with the Semantic Web technologies. Thus, application of the technologies to the data extraction capabilities may allow for dynamic knowledge elicitation and construction of large-scale personal knowledge systems. The implications of such approach are difficult to predict, but it can be said that the resulted system may produce a substantial contribution to the problem of construction of personal agents, expert systems and other products.

Finally, despite the system demonstrated good results on the tests, the behavior of the system under high load in a real situation is yet to be seen. This may raise new challenges and open a vast area for improvement. In the meantime, the interface suggested in the current study was devised during the undertaken research on the topic. Introduction of a standardized interface, which may be used to address data entities on web resources may lead to a great degree of distributed integration systems and result in a truly dynamic and personal Web.

7. REFERENCES

[1] Kim, W., Jeong, O.R. 2009. On Leveraging Social Web Sites. In *Proceedings of the 4th International Conference on Innovative Computing, Information and Control* (Kaohsiung, Taiwan, December 07 - 09, 2009). 1273-1276.

[2] Abiteboul, S., Greenshpan, O., Milo T. and Polyzotis, N. 2009. MatchUp: Autocompletion for Mashups. In *Proceedings of the 25th IEEE International Conference on Data Engineering* (Shanghai, China, March 29 - 02 April, 2009). 1479-1482.

[3] Lerner, R.M. 2008. At the forge: OpenID. *Linux Journal* 169, 9 (2008).

[4] OpenID Authentication 2.0 – Final, 2007. Retrieved April 17, 2010, from OpenID Foundation: http://openid.net/specs/openid-authentication-2_0.html

[5] Jrstad, I., Johansen, T.A., Bakken, E., Eliasson, C., Fiedler, M. and Thanh, D. 2009. Releasing the potential of OpenID & SIM. In *Proceedings of the 13th International Conference on Intelligence in Next Generation Networks* (Bordeaux, France, October 26 - 29, 2009). 1-6.

[6] Mitchell-Wong, J., Kowalczyk, R., Roshelova, A., Joy, B. and Tsai, H. 2007. OpenSocial: From Social Networks to Social Ecosystem. In *Proceedings of the 2007 Digital EcoSystems and Technologies Conference* (Cairns, Australia, February 21 - 23, 2007). 361-366.

[7] OpenSocial API Server Specification 1.0, 2010. Retrieved April 21, 2010, from OpenSocial Foundation: http://opensocial-resources.googlecode.com/svn/spec/1.0/Social-API-Server.xml

[8] Wu, W., Uram, T. and Papka, M. 2009. Web 2.0-based social informatics data grid. In *Proceedings of the 5th Grid Computing Environments Workshop*, (Portland, OR, November 20, 2009). 6.

[9] Geambasu, R., Cheung, C., Moshchuk, A., Gribble, S. and Levy, H. 2008. Organizing and sharing distributed personal web-service data. In *Proceedings of the 17th International conference on World Wide Web* (Beijing, China, April 21-25, 2008). 755-764.

[10] Enterprise OpenSocial Whitepaper, 2010. Retrieved April 22, 2010, from OpenSocial Foundation: http://www.opensocial.org/page/enterprise-opensocial

[11] Shin, D. and Lopes, R. Enabling Interoperable and Selective Data Sharing among Social Networking Sites. In Bertino, E. and Joshi, J. *Collaborative Computing: Networking, Applications and Worksharing*, Springer, Berlin, 2009, 439-450.

[12] The State of Social Networks, 2010. Retrieved March 10, 2010, from comScore: http://www2.comscore.com/l/1552/APwithfocusonMalaysiaJan10-pdf/H16B5

[13] Brusilovsky, P., Kobsa, A. and Nejdl, W. 2007. *The Adaptive Web: Methods and Strategies of Web Personalization.* Springer, Berlin, Germany.

[14] Linden G., Smith, B. and York, J. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing* 7, 1 (2003). 76-80.

[15] Segaran, T. 2007. *Programming Collective Intelligence*. O'Reilly, Sebastopol, CA.

[16] Rosaci, D., Sarné, G.M.L. and Garruzzo, S. MUADDIB: A distributed recommender system supporting device adaptivity. *ACM Transactions on Information Systems* 27, 4 (2009). 24.

[17] Meng, J., Mei, S. and Yan, Z. 2009. RESTful Web Services: A Solution for Distributed Data Integration. In *Proceedings of the 2009 International Conference on Computational Intelligence and Software Engineering* (Wuhan, China, December 11-13, 2009). 1-4.

[18] Weerawarana, S., Curbera F., Leymann, F., Storey, T. and Ferguson, D.F. 2005. *Web Services Platform Architecture*. Pearson Education, Upper Saddle River, NJ.

[19] Christensen, J.H. 2009. Using RESTful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (Orlando, FL, October 25-29, 2009). 627-634.

[20] Annett, M. and Stroulia, E. 2008. Building highly-interactive, data-intensive, REST applications: the Invenio experience. In *Proceedings of the 2008 Conference for Advanced Studies on Collaborative Research* (Ontario, CA-ON, October 27-30, 2008). 15.