From Mashup Applications to Open Data Ecosystems

Timo Aaltonen, Tommi Mikkonen, Heikki Peltola, and Arto Salminen

Department of Pervasive Computing
Tampere University of Technology
Tampere, Finland

firstname.lastname@tut.fi

ABSTRACT
Web-based software is available all over the world instantly after the online release. Applications can be used and updated without need to install anything, with natural support for collaboration, which allows users to interact and share the same applications over the Web. In addition, numerous web services allowing users to upload, download, store and modify private and public resources have emerged. However, as the amount of web services and devices used to consume as well as generate data has exploded, it is difficult to access and manage relevant data. In this paper, we start from the principles of mashups, reflect their use to the concepts of software ecosystems, and finally extend the discussion to open data generated by users themselves. As a technical contribution, we also introduce our proof-of-concept implementation of a mashup system built on wellness data, and discuss the main lessons we have learned in the process.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**] Software Architectures – *Data abstraction*. C.0 [**Computer Systems Organization**]: General – *system architectures.*

General Terms
Design. Experimentation.

Keywords
Web apps; cloud service; ecosystem; open data.

1. INTRODUCTION
Despite its origins in sharing static documents, the Web has become a software platform. Today, majority of new applications intended for desktop computers are released as web-based software. This development has its disadvantages, but numerous benefits, as well. The web-based software is available all over the world instantly after the online release. It can be used and updated without need to install anything. Applications can effortlessly support user collaboration, i.e., allow users to interact and share the same applications over the Web. In addition, numerous web services allowing users to upload, download, store and modify private and public resources have emerged. These resources can include personal images, texts, videos, e-mails, etc. as well as public data such as stock quotes, weather data and news feeds.

As the amount of web services and devices used to consume data has exploded, it has become difficult to handle and gain access to the relevant information. To be able to manage the situation, searching has become one of the most important service of the Web. However, searching can be used only for data accessing, not for analyzing or parsing it, which are also commonly needed facilities.

Similarly to resources, communication has decentralized into different services such as e-mail, social media services, instant messaging services, chats, blogs, and so on. This use of silos for particular types of resources, messages, and so on is creating artificial boundaries between different data and services. Liberating users to access data in a fashion that is open yet private, new mechanisms are needed for managing services of the Web.

An important realization is that applications built on top of the Web do not have to live by the same constraints that have characterized the evolution of conventional desktop software. The ability to dynamically combine content from numerous web sites and local resources, and the ability to instantly publish services worldwide has opened up entirely new possibilities for software development. In general, such systems are referred to as mashups, which are content aggregates that leverage the power of the Web to support instant, worldwide sharing of content.

Since leveraging already existing data is essential for building mashups, sources of open and personal data are important when developing those. While there are numerous ways to collect data online – we will address this issue later on in this paper – the most obvious input for mashup development is data that is freely available (so-called open data) as well as data that is owned by the user (personal data). The former is increasingly made available for instance from public sources, whereas the latter can be easily generated with different types of gadgets that are owned by individual people. Moreover, it is also an option to share data between individuals, resulting in crowdsourcing of data.

OpenSym '14, August 27 - 29 2014, Berlin, Germany
Copyright 2014 ACM 978-1-4503-3016-9/14/08…$15.00.
http://dx.doi.org/10.1145/2641580.2641599

In this paper, we provide an insight to our work regarding the creation of an ecosystem where wellbeing data is collected and used as basis for mashups by combining it with facilities that are readily available on the web. As a concrete technical contribution, we introduce our proof-of-concept realization of Wellness Warehouse Engine (W2E), a NoSQL database system that is capable of hosting wellness data from various gadgets in a fashion where privacy issues are taken into account. Towards the end of the paper, we also discuss our future plans regarding the system as well as provide an insight to lessons learned during the process.

The rest of this paper is structured as follows. In Section 2, we provide an overview to mashup applications and their development process at a general level. In Section 3, we discuss ecosystems that commonly lie behind the mashups, and address their main characteristics and challenges. In Section 4, we link mashup development with open data as well as with data that can be produced by users themselves, provided that suitable gadgets are available. In Section 5, we introduce our proof-of-concept implementation where wellness related personal data can be linked with other data available online, and in Section 6, we address lessons learned during the design and implementation process. Finally, in Section 7 we draw some final conclusions.

2. OVERVIEW OF MASHUPS

Mashups can be characterized as applications that combine resources - data, code and other content - from different services in the Web into an integrated experience. They can combine the content in new, unforeseen ways, thus creating entirely new web services, or they can provide new visualizations for already existing services. For instance, a mashup can combine a map with images that can be attached to specific locations. In contrast, another mashup can visualize the images in novel fashion, for example on a timeline or as a collage.

Well-build mashups have functionality for filtering source data. By having adjustable filters a mashup can provide more relevant results. Filters can be based on much more relevant variables than manually entered limits such as the highest and the lowest price of a product. Such filters can be time of the day, location of the user, past activity of the user, activity of other users (trends), profile setting of user’s mobile device, etc. Heavy processing, e.g. filtering images with face detection algorithm, can be executed on the server, using MashReduce programming model [1], for instance.

There are four special characteristics that make mashup systems different from more traditional applications [23]:

1. In mashup development there is a lot more focus on reusing the content rather than the implementation of a web site. While standardized formats for various content formats (such as images and videos) exist, it is often surprisingly difficult to reuse the implementation of a web site in other contexts, e.g., because the current web technologies do not make it easy to specify which parts of the web site are intended to be reusable in other contexts and which are not. In the same fashion, many mashups reuse the visual representation of sites only (e.g., a map or the layout of a web site), while others reuse the content (substance) separately from its visual representation. No well-defined rules or interfaces exist (apart from HTML, CSS and the DOM) for keeping the content separate from its visual representation. We will discuss these topics in more detail in the next section.
2. Mashups are far more dynamic than conventional (binary) software components. Since mashups are all about combining content from multiple web sites in a highly dynamic fashion, they cannot be built easily with static programming languages that require advance compilation and static type checking[1]. This has created a trend towards more and more dynamic programming languages such as JavaScript, Python or Perl. Even though these languages were originally intended for relatively simple scripting tasks, many of them (especially JavaScript) are increasingly used as “real” programming languages. We have summarized our experiences in using JavaScript as a real programming language in another paper [1].
3. Due to the increased focus on content rather than on implementation, mashup developer base is different from conventional software development projects. A mashup developer does not necessarily have any formal training or background in software development. Rather, it is far more common for them to have some kind of a media background. Moreover, special tools or a particular development process are not a prerequisite for developing mashups, but a simple text editor and some example web pages are often enough for the composition of a simple system. Consequently, mashup developers are not always aware of the benefits of well-established software engineering principles such as separation of concerns, modularity or information hiding.
4. The distribution and sharing power of the Web makes it exceptionally easy to reuse content in unforeseen, unexpected ways. Basically, anything that is made available on the Web is instantly accessible to anybody anywhere in the world with a web browser. This increases the potential content user and re-user base by several orders of magnitude compared to conventional software components that are typically distributed in a far more controlled and limited fashion. Often, the developer of a web site may not be aware at all that content from his or her site is being used in other contexts as well. The same also applies to implementations, but due to the above complications their reuse is hardened in practice.

Jointly, these characteristics enable the design of compelling applications where user-generated data, produced in the crowdsourcing fashion, data readily available from online services, and personal content can all be used. In such systems, contracts, intellectual property rights, data and format dependencies, and other forms of relations bind the different stakeholders together to create mutual benefits. In general, these are referred to as mashup ecosystems.

3. MASHUP ECOSYSTEMS

Since mashups by definition combine data from multiple sources, the stakeholders that provide this data form an ecosystem, i.e. a set of entities that act as a single unit instead of each participating business acting separately.

Such ecosystem – formed by service providers, mashup authors, and users as visualized in Figure 1 – need not be controlled by a central authority. In contrast, even though mashup authors and service providers may have explicit service level agreements (SLA), it is common that mashups are developed without such contracts, and the ecosystem is formed implicitly. For instance, one can build a mashup on top of services freely available in the web with liberal enough licenses. In a broad sense, any web document author can be considered as a service provider, as it is common that content is gathered from web sites by technique called “screen scraping” or “web scraping”, where source data is parsed from HTML pages aimed at human readers.

In the following we will first provide some background information regarding mashup ecosystems, and then advance to some challenges associated with the establishment of new mashup ecosystems.

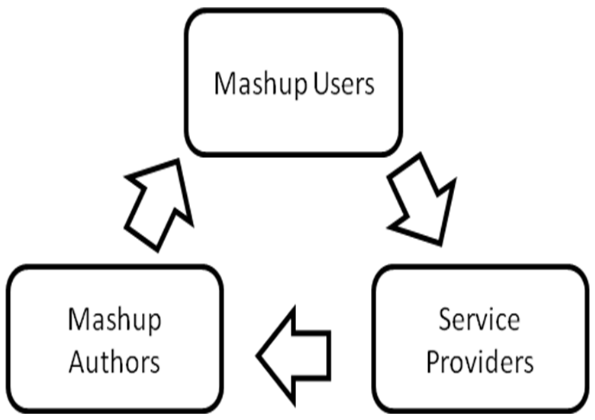

Figure 1. Mashup ecosystem key actors are mashup users, mashup authors, and service providers.

3.1 Background

Yu and Woodard [2] have described mashup ecosystems by using the ProgrammableWeb mashup indexing service (http://www.programmableweb.com/) data as source. They investigate the structure and dynamics of the Web 2.0 ecosystems by analyzing the data available about mashups and APIs. The first finding was that at the time of the study APIs were organized into three tiers, which were 1) the most popular API (Google Maps), 2) popular APIs (many APIs used for social services and searching) and 3) less popular APIs (APIs often used for blogging, online retail, music, videos and feeds). The second finding was that mashups are often composed by combining APIs across tiers. This highlights the central role of the most popular APIs, but also reveals the importance of less popular APIs in dilution of the ecosystem. Many of the third tier APIs bring together novel combinations of functionality.

Another interesting finding is that in contrast to what has been suggested [3], there is no long tail of services that would form a basis for a significant number of mashups. Instead, Yu and Woodart noticed that 95 % of mashups are build on 20 % of services, which is much more than in the famous Pareto Principle, or 80/20 rule as it is often called. Moreover, they noted that 51 % of services were not used by mashups at all. However, one should bear in mind that Yu's and Woodard's data source, ProgrammableWeb, lists only those services and mashups that have been added to it by developers. Therefore there are services and mashups that are not included in the source data.

Bosch has reviewed mashup ecosystem from end-user programming point of view [4]. Bosch also pointed out two success factors as well as two challenges that this ecosystem has. The two success factors are, first, the value that end-users gain by designing their own applications, and second, sharing of applications among users. The two challenges are enabling the end-user programming for inexperienced developers and minimizing ecosystem maintenance efforts. Furthermore, Bosch identifies so called “undirected developers’" that are able to use the platform in unforeseen ways and provide significant innovations for the overall ecosystem. Similarly to [4], our perception is that mashup ecosystems are very valuable for end-users and service providers.

3.2 Designing Services

Service providers are crucial stakeholders in mashup ecosystems, as they provide the necessary content that is reused in mashups. There are numerous motives to allow liberal access to the content of a service. One rationale is a desire for getting a wider audience for certain platform, product, or content accessed through the service. Moreover, opening a service can lead to numerous clients created by third party developers to emerge on different platforms and for different user requirements. Some services are designed so that spreading advertising messages along with the content is possible.

Service providers support mashup ecosystems in several clearly identifiable levels. These are described in more detail in the following.

3.2.1 No support for mashups

Some web content authors do not support mashups at all and provide their content solely as regular web documents. This kind of content is still accessible with ``screen scraping'', but such accessing is typically error prone, and it often is illegitimate. Some services even have implemented technical measures to prevent scraping. Furthermore, even if reusing the content in mashups would be allowed, the web content author does not have control on what parts of the content is reused, and it is difficult to build a business model around such approach towards mashups.

In addition, it is likely that accessing the content is very inefficient and cumbersome from mashup author's point of view. Furthermore, since even the smallest change in the web page can lead to a different interpretation of the content, mashups relying on such services are usually somewhat fragile.

3.2.2 Access through a web feed

It is common that regularly updated sites, such as blogs or news sites, provide their content through RSS, Atom, or other type of web feed. A web feed is easy to set up and maintain, particularly if some publishing system is used. The feed is intended mainly for users to subscribe with some feed reader application, but at the same time the data becomes accessible for mashups, too. While it is possible to establish some kind of licensing for reusing the content, the control over the content is still rather coarse.

Use cases of web feeds are limited to accessing the content as a whole, because, for instance, querying certain content item is not possible. Utilizing web feeds in mashups is typically straightforward as helpful libraries and tools for such task are available on most platforms. Some dedicated mashup tools, Yahoo! Pipes (http://pipes.yahoo.com/) for instance, support only web feeds if content from an arbitrary service is desired to be included into a mashup.

3.2.3 Access through a web interface

Providing a service with a web interface, typically following either REST or SOAP architecture style, enables using the service in mashups. Use cases of such interface allow not just data accessing but other types of services as well. For instance, a service can provide means for social communication, authentication, database accessing, or specialized functions such as reverse geocoding or music identifying.

Setting up a web service with REST or SOAP interface requires careful planning and implementation, especially if sensitive information is handled. However, such system allows fine-grained control over the content as well as applications using the interface, and it enables different kinds of business models. Service load can be handled as well by limiting requests made in a time period, even individually for each application.

Utilizing well-designed web interfaces in mashups is straightforward, and maintaining efforts that are needed when the service is updated are typically trivial. Conveniently, the content can be provided in different formats for the mashup developers to choose from, for instance both JSON and XML formats are often supported.

3.2.4 Access through a programmatic interface

Establishing a programmatic JavaScript API allows to integrate the sevice tightly with arbitrary web applications and mashup ecosystems. Such interface is used by including a JavaScript library into the application, which makes it possible to use the service with regular JavaScript function calls. Typically the JavaScript library is downloaded from the service provider's server instead of having a copy on the server hosting the mashup, which makes possible to always use the most recent version of the library.

Setting up a programmatic JavaScript interface requires careful engineering, but it enables superior control over the content and applications. Diverse business models are possible, and the content can be provided with different terms and licenses for individual clients. Program code of the JavaScript library is often protected against misuse by code obfuscation or by other technical means. Considerable downside of the programmatic interfaces is that updating the interface affects directly on the mashup implementation. Therefore, programmatic interfaces are often provided in numerous versions, and a new version is introduced whenever features are added. Consequently, bug fixes need to be performed on all the versions, which makes maintaining the interface more laborious.

Another downside is that if a programmatic interface is desired to be used on other runtime environments than a web browser, a parallel version needs to be provided. For instance, Google Maps API (https://developers.google.com/maps/) has separate native SDKs for Android and iOS mobile operating systems, and it also used to have another version for Adobe Flash Player (http://www.adobe.com/products/flashplayer.html). The Flash version was deprecated in September, 2011.

The proliferation of programmatic interfaces is a step towards software created from downloadable components. This is sometimes referred to as mashware, web software development technique described in [8]. The most successful example of this kind of interface is Google Maps JavaScript API, which is also the most popular interface used in mashups [2]. It can be argued that one reason behind the success of this API has been the implementation style, which is particularly convenient for application developers, as it is similar to DOM (Document Object Model) and other interfaces that can be found from web browsers. However, Google Maps is not the only example of programmatic interface approach, as there are numerous other examples including user authentication, social networking, HTML5 music and video players, and data visualization, among others.

Until recently most of the services have been provided for free with the exception of some very specialized ones such as image content recognition and sentiment analysis services. However, in October 2011 Google announced that Google Maps API will be provided in two different versions: free and non-free, with the latter called Google Maps API for Business. The one with a price tag provides more advantageous features such as higher request limitations and technical support. Even if this is the first remarkable example of this kind of development, it is an interesting change, particularly when bearing in mind that the Google Maps is the most popular service used in mashups, and it is widely utilized in other types of web applications, too. Therefore, this development may indicate a beginning of a new kind of emerging business model.

4. TOWARDS OPEN DATA MASHUPS

In addition to data that is available via online services, there is more and more data that is produced by users. Some of this data is uploaded to services such as Flickr.com or Youtube.com where numerous people can access the data. However, it is also possible to upload material that is meant to be for personal use to clouds. Then, the cloud provider will take care of numerous routines such as backing up the data in case of equipment malfunction. Moreover, different social media and networking utilities allow sharing of different items, which enables distribution and consumption of data at a rate that exceeds all traditional means.

This Data-as-a-Service approach is spearheaded by the Open Data (http://en.wikipedia.org/wiki/Open_data) movement, which advocates the transformation towards open – but still secure – services. The Open Data movement, reflecting the ideals of other open movements, such as open source, open content, and open access, is building on the idea that data should be made freely available for everyone to use, refine, and redistribute. The term Open Data itself is more recent, and has become popular with the launch of numerous open data government initiatives, like data.gov.

In general, combining open data with personal data and crowdsourcing requires services enables services that reflect several dimensions of citizens' lives. Today, such dimensions are only available for individual companies, and individual persons have little opportunities regarding the data associated with them; the data is kept proprietary for business reasons as it helps in creating e.g. better marketing strategies. In fact, it is becoming less and less clear what kind of data is being stored regarding individuals and their actions in proprietary web sites.

Finally, we do wish to acknowledge that while sometimes considered harmful, proprietary use of data has also helped with protecting privacy. When considering open data and its links with other sources of data, additional considerations are necessary.

This, together with e.g. terms of use in different services, requires that special attention is placed to privacy.

5. IMPLEMENTATION

In this section, we introduce our proof-of-concept implementation that enables developing applications that use data from various gadgets.

5.1 Wellness Warehouse Engine

Our proof-of-concept implementation is geared towards the well-being domain. A person's well-being is sum of numerous factors, including for instance sleep, activities during the day, and nutrition, to name some obvious examples. As these factors are so fundamental, sensors measuring them have recently become popular – for instance Fitbit, Beddit, Withings, Jawbone, Nike Fuel, OmegaWave, Endomondo, Sports Tracker, Polar, and Suunto are some of the most well-known brands in the field.

Unfortunately, all this data is in service providers' silos, and all wellness apps must connect to them. Consequently vendor lock effect is tight for users; changing from Endomondo to Sports Tracker simply does not happen, as in many cases having access to history data is important.

Our system, called Wellness Warehouse Engine (W2E, homepage at http://w2e.fi/) solves the problems associated with vendor-specific silos. The service connects to the silos and collects the wellness data to one service. The benefits of including all the data into one service are many:

- Data is unified during the process; all measurements same units and presentations.
- Multi-sensor analyses are possible.
- Wellness apps connect only to W2E, not to all possible silos.
- Wellness data can be accessed with a well-defined REST API, which makes it easy to compose client applications. While a JavaScript API would simplify the development of apps that run inside the browser, we found using REST API as a simpler alternative for apps that are run in mobile devices.

The W2E system has three main functions that we have been studying. These are gathering, unifying, and analyzing data. Each of these is addressed in more detail in the following.

- Gathering. W2E accesses numerous wellness services. The data from these source services is stored for later processing. Service authentication is centralized to W2E for most convenient user experience.
- Unifying. W2E service provides the data in varying formats. For instance, units of measurement and data frequency are service-provider dependent. By unifying the data we enable coherent input for analysis and other use.
- Analysing. Further analysis of the data enables you to provide quantified feedback for the users. Our automatic background processing calculates analysis as soon as the source data is available. Prefilled forms are available to use subjective data in analysis.

5.2 Clients

Originally the main idea of W2E was simply to gather and host data, but lately also some client applications have been implemented to demonstrate the options offered by W2E. These clients are addressed in the following.

- Activity Calendar. To demonstrate the capabilities of mashing up content from various devices, we created a wellness calendar (Figure 2). This application is a full-fledged calendar where user's well-being activities are visualized as daily events. More detailed view of an activity event can be seen by clicking it. The activity events can be browsed one month or one week at a time.
- Dashboard. This application visualizes all data of a W2E user. There are two views that are provided to the data (Figures 3 and 4). The single day view shows detailed activity during one day, and the longer view visualizes changes in one's well-being over one week to six months.

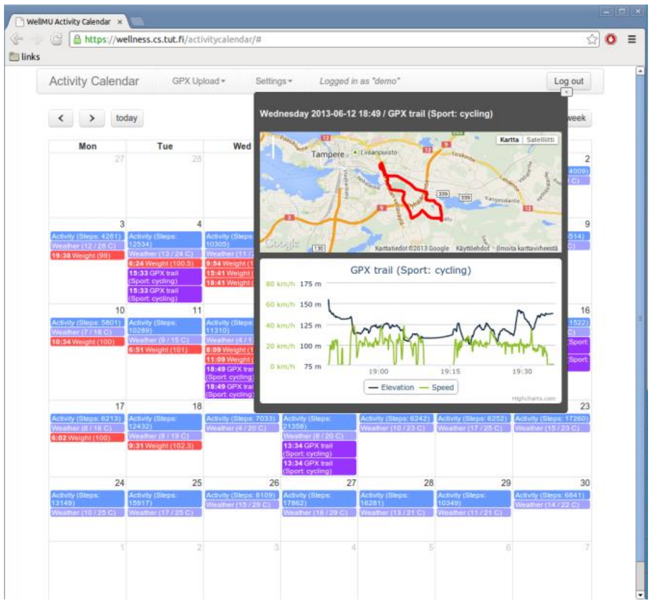
WellMU Activity Calendar
https://wellness.cs.tut.fi/activitycalendar/#
Activity Calendar
GPX Upload
Settings
Logged in as "demo"
Log out
today
week
Mon
Tue
Wed
Wednesday 2013-06-12 18:49 / GPX trail (Sport: cycling)
Tampere
GPX trail (Sport: cycling)
Elevation
Speed

Figure 2. Calendar application.

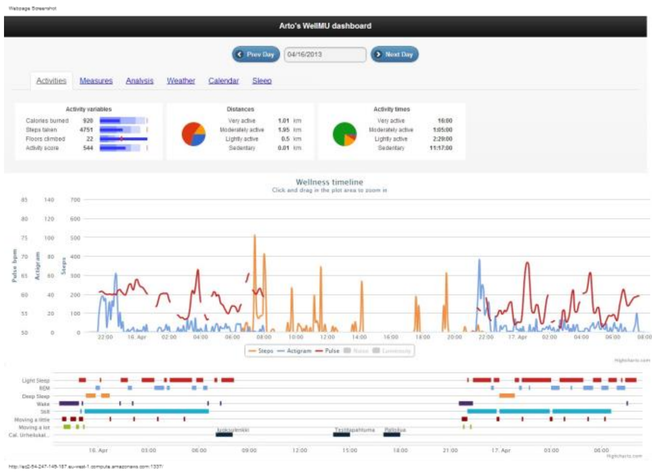

Figure 3. Dashboard application – single day view.

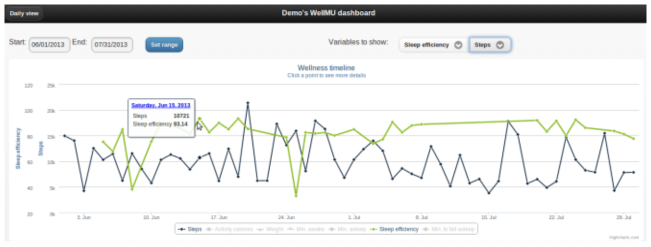

Figure 4. Dashboard application – longer view.

5.3 Implementation Details

Server-side has been implemented with Python using Django framework (https://www.djangoproject.com). Django emphasizes reusability and "pluggability" of code components. The server has a plugin for each of the data source services. Each plugin has the functionality for accessing and unifying the data.

Data is separated into two distinct databases. A relational database (MySQL, http://www.mysql.com) is used for storing user information and access tokens to source services. A NoSQL (MongoDB, http://www.mongodb.org) database is used for storing the data from the gadgets. Most of the data from gadgets is in JSON (JavaScript Object Notation) format, which is ideal for NoSQL.

Figure 5 shows the internal structure of W2E. Rest API handles all communication from the client programs. The Mashup server requests data from all of the data source services using the "Service access". Received data is stored without any modification to "RAW data". The data is also unified and stored. Client programs are offered both the raw and unified data. The data can be further processed with Analyser component. Data analyzing could also be done by client programs.

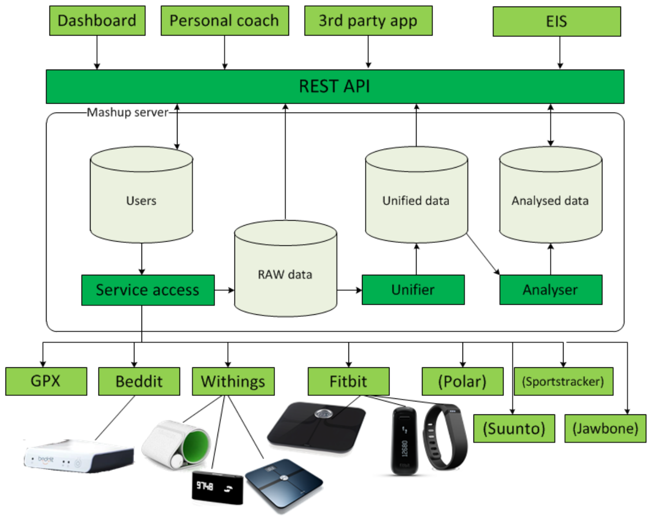

Figure 5. Internal structure of W2E.

Both client applications have been implemented as web applications using HTML, CSS, and JavaScript using the AngularJS framework (http://angularjs.org). No native client for mobile devices have been implemented yet.

5.4 Future Work

We are currently working on integrating the mashup service with Taltioni (http://www.taltioni.fi/en), a Finnish Personal Health Record (PHR) organization aiming at offering citizens, health care professionals, and well-being service providers a common health- and wellness information database.

Analyzing the actual data that we are hosting is still mostly future work. We have experimented simple calculations, such as moving averages and implemented questionnaire forms to calculate PSQI (Pittsburgh Sleep Quality Index) values. We are doing collaboration with the Department of Signal Processing and hoping to create more advanced analyzing in the future.

At the moment W2E supports only a handful of gadgets. Support for additional gadgets that are offering open developer APIs is planned. Adding new data sources is made as easy as possible with the plugin-architecture on the server.

Users have to grant W2E access to their data on data source services, in order for W2E to be able to access the data. Additionally, users can link services as data sinks that are using the collected data. It might get complicated for users to understand what each application is allowed to access and do. Therefore, we are planning to conduct large-scale usability tests related to account management, authorization and data flows.

At this point, we are planning to have co-operation with other research institutes to support research with the data that has been already gathered. As the data sources and amount of data are growing, we are getting more and more possibilities for research.

In the long run, our goal is to get as many users to W2E as possible in order to increase the amount of data available for research purposes, however keeping in mind the anonymity of our users. As for new users, continuity of a service is considered very important. Therefore, to ensure this project will not stay only an academic exercise that would eventually fade away, we are planning to establish a startup company to offer the service commercially.

6. LESSONS LEARNED

Device manufacturers have web services for storing data from their devices. Each vendor hopes to build the best services for their users, to avoid the users from changing to other vendor devices. In the past, this has led to storing all the data to vendor-specific silos. However, more recently vendors are starting to realize that by opening the data to other applications, users are getting more value from the devices, which in turn may increase device sales. Vendors are concentrating more on the devices, which is their main source of income and area of expertise. Applications that are based on the devices, on the other hand can be created either by the device manufacturers themselves or by some third party. Even today, original equipment manufacturers that provide the devices can require in their license terms that all user-related data is erased, whereas we would at very least like to preserve data in anonymized form.

Silos that are preventing the use of data form different sources, result also from different units and formats of the data. Therefore unification is a necessary step, because we need to be able to compare as well as use the same type of data that originates from different sources. This also helps when dealing with implementation changes in different versions of interfaces in the devices.

Data should only be unified, not simplified. Clients should be aware that data from different sources may be diverse. Even after unification, some data may be more detailed than other. API documentation must be comprehensive, so that client application developers know what kind of data to expect. Furthermore, two data sources may be describing the same phenomena. For example, sleep can be tracked with two devices at the same time. It is not the server's job to decide which of these is more accurate or relevant in each use case. It can be left for the client to decide which data it uses.

Some data sources offer time in user's local time while others use Coordinated Universal Time (UTC). Time should always be transformed to UTC on the server. Local time zone should also be stored and offered to client programs, since they most likely want to show local time to the users. In fact, the time may not even reflect the actual data, but for instance reflect the time and date the device is using as the default upon booting for the first time. Obviously, preparing to all such anomalies is in general impossible.

7. CONCLUSIONS

The number of different gadgets used to monitor wellbeing has been rapidly increasing. In order to avoid vendor-specific silos that prevent the use of data collected with such devices, we have created an open data mashup system, where data from different sources is automatically unified in order to enhance interoperability. In the present implementation, data is kept for personal use only, but in the future we envision social networking as well as business opportunities, to the extent of launching a startup company to host the data.

At present, we have launched the first official beta release open for public. The system has been up and running for a few months with the number of users steadily increasing. Furthermore, some organizations have been provided with API documentation and data access to databases for testing the interface within their own applications. Experiences on these issues will be reported separately as a part of future research.

ACKNOWLEDGMENTS

This research has been supported by TEKES (project RICTI) and SITRA. In addition, the authors would like to thank Pekka Turunen and Tuomas Teuri from Taltioni and Arto Leppisaari from Movendos for their time, knowledge, and connections.

8. REFERENCES

[1] Salo, J., Aaltonen, T., Mikkonen, T.: MashReduce: Server-side mashups for mobile devices. In Proceedings of the 6th international conference on Advances in grid and pervasive computing (GPC'11), Jukka Riekki, Mika Ylianttila, and Minyi Guo (eds.), pp. 168-177, Springer-Verlag, Berlin, Heidelberg (2011).

[2] Yu, S., Woodard, C. J.: Innovation in the Programmable Web: Characterizing the Mashup Ecosystem. In Service-Oriented Computing Workshops (ICSOC'2008), George Feuerlicht and Winfried Lamersdorf (Eds.). Lecture Notes In Computer Science, Vol. 5472, pp. 136-147, Springer-Verlag, Berlin, Heidelberg (2008).

[3] Hoyer, V., Stanoesvka-Slabeva, K., Janner, T., Schroth, C.; Enterprise Mashups: Design Principles towards the Long Tail of User Needs. IEEE International Conference on Services Computing (SCC '08), vol.2, pp. 601-602 (2008).

[4] Bosch, J.: From software product lines to software ecosystems. In Proceedings of the 13th International Software Product Line Conference (SPLC '09), pp. 111-119. Carnegie Mellon University, Pittsburgh, PA, USA (2009).

[5] Mikkonen, T., Salminen, A.: Towards a reference architecture for mashups. In Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems (OTM'11), Robert Meersman, Tharam Dillon, and Pilar Herrero (eds.), pp. 647-656. Springer-Verlag, Berlin, Heidelberg (2011).

[6] Salminen, A., Mikkonen, T., Nyrhinen, F., Taivalsaari, A.: Developing client-side mashups: experiences, guidelines and the road ahead. In Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek '10) pp. 161-168. ACM, New York, NY, USA (2010)

[7] Weiss, M., Sari, S.: Evolution of the mashup ecosystem by copying. In Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups (Mashups '09/'10). ACM, New York, NY, USA (2010).

[8] Taivalsaari, A.: Mashware: The Future of Web Applications. Technical Report. Sun Microsystems, Inc., Mountain View, CA, USA (2009).

[9] Salminen, A.: Mashups in Web 3.0. In Proceedings of 8th International Conference on Web Information Systems and Technologies (WebIST'2012, 18-21 April, 2012, Porto, Portugal) pp. 189-194. ACM, New York, NY, USA (2012).

[10] Hendler, J.: Web 3.0 Emerging. Computer 42, 1 (January 2009), pp. 111-113 (2009).

[11] Silva, J. M., Mahfujur Rahman, A. S., El Saddik, A.: Web 3.0: a vision for bridging the gap between real and virtual. In Proceedings of the 1st ACM international workshop on Communicability design and evaluation in cultural and ecological multimedia system (CommunicabilityMS'08) pp. 9-14. ACM, New York, NY, USA, (2008).

[12] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, Scientific Am., May 2001, pp. 34–43 (2001).

[13] Shadbolt, N., Berners-Lee, T., Hall, W.: The Semantic Web Revisited. IEEE Intelligent Systems 21, 3 (May 2006), pp. 96-101 (2006).

[14] Ed. Hickson, I. (2011) HTML5 Specification, working draft, W3C. http://www.w3.org/TR/html5/ Cited 15 Oct 2012.

[15] Ed. Marrin, C. (2011) WebGL Specification 1.0. Khronos Group. https://www.khronos.org/registry/webgl/specs/1.0/ Cited 15 Oct 2012.

[16] W3C (2010) Cross-Origin Resource Sharing. http://www.w3.org/TR/cors/ Cited 15 Oct 2012.

[17] MacManus, R., Schmidt, E. (2007) Defines Web 3.0. http://www.readwriteweb.com/archives/eric_schmidt_defines_web_30.php Cited 15 Oct 2012.

[18] Salminen, A., Mikkonen, T.: Mashups – Software Ecosystems for the Web Era. In Proceedings of ICSOB'2012 4th International Workshop on Software Ecosystems (2012).

[19] Mcilroy, D.: Mass-Produced software components. In Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany, pp. 88–98 (1968).

[20] Mikkonen, T., Taivalsaari, A.: The mashware challenge: bridging the gap between web development and software engineering. In Proceedings of the FSE/SDP workshop on Future of software engineering research, FoSER'10, pp. 245–250, ACM New York, NY, USA (2010).

[21] Mikkonen, T., Salminen, A.: Implementing Mobile Mashware Architecture: Downloadable Components as On-Demand Services. In Proceedings of The 9th International Conference on Mobile Web Information Systems (2012).

[22] Ed. Hickson, I. (2012) Web Workers. Candidate recommendation, W3C, May 2012. http://www.w3.org/TR/workers/. Cited 15 Oct 2012.

[23] Taivalsaari, A., Mikkonen, T.: Mashups and Modularity: Towards Secure and Reusable Web Applications. In Proceedings of the First Workshop on Social Software Engineering and Applications (2008).

[24] Brooks, R.: The next 50 years. Communications of the ACM, 51(1): pp. 63-64 (2008).

[25] Wang, G., Yang, S., Han, Y.: Mashroom: end-user mashup programming using nested tables. In Proceedings of the 18th international conference on World Wide Web, pp. 861-870 (2009).

[26] Xuanzhe, L., Zhao, Q., Huang, G., Jin, Z., Mei, H.: iMashup: assisting end-user programming for the service-oriented web. In Proceedings of the IEEE/ACM international conference on Automated software engineering. ACM, New York, NY, USA, 285-288 (2010).

[27] Crockford, D. Javascript: The Good Parts. O'Reilly Media, Inc. (2008).