Mashups – Software Ecosystems for the Web Era

Arto Salminen, Tommi Mikkonen

Department of Software Systems
Tampere University of Technology
P.O.Box 553, FI-33101 Tampere, Finland
arto.salminen@tut.fi, tommi.mikkonen@tut.fi

Abstract. Web-based software and services are available all over the world instantly after they are released online. They can be used and updated without need to install anything, and once in place, they can also be reused in other contexts. As the amount of web services and devices used to consume data has exploded, it is becoming difficult to handle and gain access to the relevant data. Mashups are a new breed of web applications that act as content aggregates that leverage the power of the Web to support instant, worldwide sharing of content. Another dimension of mashups is that since they build on services that are readily available, they are also implicitly creating software ecosystems between service providers and application developers. In this paper, we address the role of mashups in the creation of software ecosystems for the web era. In addition, we identify four levels of support that service providers can offer for mashups. Furthermore, we will also discuss the different flavors of mashups as well as implementation considerations that are relevant from the ecosystem perspective.

Key words: Mashups, web applications, software ecosystems

1 Introduction

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 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 is difficult to handle and gain access to the relevant data. To be able to handle 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. Similarly to resources, communication has decentralized into different services such as e-mail, different social media services, instant messaging services, chats, blogs, etc. Therefore, new mechanisms are needed for resource handling and communication 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.

In this paper, we address the role of mashups in the creation of implicit and explicit software ecosystems [1] for the web era. We perform this in the following fashion. Section 2 provides an overview to mashups and the potential associated with mashup development. Section 3 discusses mashup ecosystems from the viewpoint of already existing research as well as challenges we have encountered in practice. In addition, this section identifies the four levels of support that service providers can offer for mashups, and discusses the associated consequences of each approach. Section 4 discusses the different breeds and types of mashup ecosystems that have been introduced, and Section 5 discusses implementation considerations. Finally, Section 6 draws some final conclusions.

2 Mashups: An Overview

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

Mashups have potential for great user experiences, as they include more functions than just composition. Mashups can be used to filter, combine and modify data retrieved from multiple sources over the Web. Combining web resources into mashups is an efficient way to create new services or extract relevant information from a complex mixture of source data. Even unexpected innovations are possible as mashups can combine resources in unforeseen fashion. Furthermore, mashups are even more usable when non-technical users can create them with special purpose tools and have their own views for data. This is very inspiring part of mashups as it allows creative users to design their own applications that are capable of doing unexpected things. Allowing "do-it-yourself" mashups serve the long tail of users having diverse needs that are not fulfilled by existing applications or services. On embedded devices, mobile devices being at the forefront, mashups can benefit from accessing the user's context to combine resources, potentially automatically.

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 users mobile device, etc. Heavy processing, e.g. filtering images with face detection algorithm, can be executed on the server, using MashReduce programming model [2], for instance.

Different kinds of dependability mechanisms play an important role in a mashup. At least the mashup should be implemented so that it checks whether the input data is correct. More sophisticated mashups can have fall-back mechanisms that, instead just giving up on error, try to use next best strategy to ensure even partial functionality. Furthermore, mashups can have controlling mechanisms that supervise the functionality and replace failing parts with other ones. In addition, mashups can have capabilities to extract the result mashup to some external viewing device and change the user interface of the mashup accordingly. For instance, this enables the creation of a mashup in a mobile device whereas the resulting output can be shown on a bigger screen if one is available.

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. This ecosystem – formed by service providers, mashup authors, and users as visualized in Fig. 1 – need not be controlled by a central authority. In contrast, even though mashup authors and service providers may have an 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.

3.1 Background

Yu and Woodard [3] 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

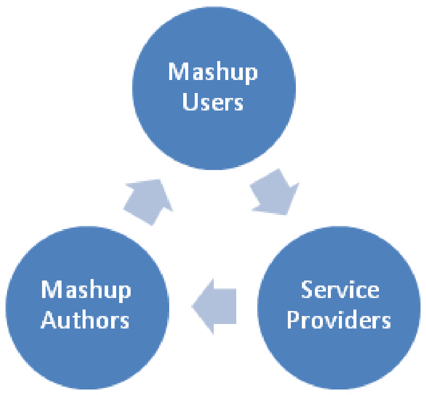

Fig. 1. Mashup Ecosystem

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 [4], 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 [5]. 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 [5], our perception is that mashup ecosystems are very valuable for end-users and service providers. However, despite popularity that mashups have gained, polished end-user programming solutions for mashups have not been very successful. In contrast, some promising efforts by major players on the field, such as Google Mashup Editor and Microsoft Popfly, have been discontinued, and mashup composing still lacks tool support. On the other hand, mashup development has focused on building applications with traditional web development tools and architectures. Consequently, our previous research has been focused on applying good software engineering practises on mashups, with the most practical tools at hand [6, 7].

Another interesting study is concerning the way a mashup ecosystem grows. For instance, hypothesis in [8] is that mashup developers create new mashups by copying existing ones. Simulations suggest that this would be true, as it is in line with the reports about mashup ecosystem growing [3]. However, the hypothesis of [8] has not been tested empirically.

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 four identifiable levels, which are described in the following:

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.
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, as, 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. *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 allows 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.
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 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, which is sometimes referred to as *mash-*

ware, web software development technique described in [9]. The most successful example of this kind of interface is Google Maps JavaScript API, which is also the most popular interface used in mashups [3]. 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 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 prize 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.

3.3 Designing Mashups

Typically, mashups are build with combination of server- and client-side parts. Functionality between these two parts is divided according to what is suitable for the current design. In the early days, dynamic web sites were created on server-side with combination of C programs, Perl, and shell scripts using Common Gateway Interface (CGI). Today, server-side web applications are often developed with Java, server-side JavaScript, Perl, PHP, Python or other suitable language. Applications of this kind work especially well if the client device has low processing resources as heavy processing takes place at the server-end and the client just shows the result. As client-end terminals have become more capable, it has become possible to compose mashups where the business logic resides completely on the client-end.

While mashups can be constructed in numerous different ways with a plethora of tools, there still are major practical problems related to mashup composing and security. For instance, the web browser security model is too restricting for mashups, tools introduced are lacking behind, and using dynamic languages for large applications is an unknown territory for many developers. The field of web programming is constantly changing as new interfaces, technologies and frameworks build upon novel technologies emerge constantly. The amount of different, constantly evolving APIs with different licenses is overwhelming. When developing large-scale mashups, situation may be even more problematic. Mashup authors build their applications on web services, and mashup users can add content to these services and consume it with mashups. Such ecosystem has commercial potential, which is, however, limited because of technical, legal and other reasons. Some of the issues are general for both mobile and desktop environment, but naturally mobile mashups have their own specific things to handle as well.

While mobility restricts applications and application development, at the same time it is a great enabler from the mashup development point of view. The dynamic nature of mashups suits well for different ways mobile terminals can be used. Often, the information needed on the fly is related to user's context, which can be available for applications to access automatically [10]. This opens up opportunities to provide advantageous user experiences, as mashups can dynamically present eligible information, possibly even automatically without requiring specific user action. However, as mobile devices capabilities are limited, extending mashups to the mobile domain is not trivial, and special solutions are sometimes necessary. Mobile mashup ecosystem challenges, especially from utilizing multimedia in mashups point of view, have been described in detail in our previous paper [11].

There are situations when the composition of a mashup is not possible using only dynamic code. For example, applications that require a lot of computation power or access to interfaces that are not available for dynamic code, have to be constructed with both dynamic and native code. Therefore, offering an interface for mixing web technologies with the capabilities of native software components is sometimes necessary. On the other hand, utilizing hybrid technology allows one to combine the best of both worlds: performance and eye candy of traditional, installed binary applications and pervasiveness and seemingly infinite resources of the web.

3.4 Legal Considerations

In general, web interface legal terms and conditions are diverse. Commonly service providers set restrictions for those uploading content to the service, as well as those utilizing content of the sevice through an API, including mashup developers. In the following, some typical requirements and terms that affect mashup development are described.

- Service Level Agreements (SLA) are used to provide uptime guarantee or to state that the API has no liability for downtime or unexpected changes. Sometimes the latter is available for those who use the interface for free, and the former for paying customers.
- If the interface allows accessing user created content under different licenses, terms of service (TOS) require developers to strictly follow those licenses. If the application uses a cache, also the cache needs to reflect changes in content's licenses and availability. Sometimes service terms determine time limits for the cache reflecting these changes. Moreover, caching may be forbidden completely.
- If the interface enables accessing user's private data, TOS usually include restrictions about how this data can be used and stored. The service provider's logo or other branding needs to be explicitly available in the mashup. Other

services require adding acknowledgements to application source code. Detailed terms on how the branding is presented may be represented. For instance, when using Google Maps, the terms of the Google Maps API require that the Google logo is the largest logo in the final implementation (`https://developers.google.com/maps/terms`).

– Interface access rate can be limited to a certain amount of requests in a time period. For instance, Twitter limits unauthenticated calls to 150 requests per hour, whereas authorized calls are limited to 350 requests per hour (`https://dev.twitter.com/docs/rate-limiting`).
– Certain types of applications may be prohibited by the service provider. For instance, Flickr TOS deny using Flickr API for any application that replicates or attempts to replace the essential user experience of Flickr.com (`http://www.flickr.com/services/api/tos/`).
– Repeated violations of interface terms, for instance exceeding use rates or using the API in a forbidden type of an application, may make the service provider to terminate certain application from accessing the interface. Technically this can be achieved by restricting application IP addresses or application specific API key from accessing the service. In practice, TOS often contain a clause for such situation, although we have no data how commonly the clause is exercised.

The above issues are further complicated by the fact that in many cases, mashup developers have not signed a formal contract with service providers, but rely on licenses. Consequently, as copyright owners and service providers can change licenses more liberally than signed contracts, developers may end up accidentally violating license rights overnight when the original service provider updates license terms.

4 Sample Mashup Breeds and Ecosystems

Mashups can be classified based on numerous criteria, which in many ways affects the fashion the associated ecosystem can be established. One can classify mashups into breeds, such as server- and client-side mashups, and multiple and single API mashups. Moreover, mashup ecosystems can be classified into explicit and implicit ecosystems. The former includes commercial and enterprise mashups, and the latter includes situational mashups, as well as most mashups, that can be classified according to the most essential API used, establishing an ecosystem that is led by the provider of this API. In the following, mashup breeds and ecosystems are described in more detail and examples of different types of mashup ecosystems are presented.

4.1 Mashup Breeds

Server- and client-side mashups. One way to classify mashups is division between server-side and client-side mashups, based on where downloading, processing and generating of the web content takes place. Server-side mashups application logic as well as accessing different web resources is implemented at the server-end. Client-side mashups are implemented completely on the client-end so that processing takes place at the user's web browser. Because of historical reasons, server-side approach has been more popular in the past, but as the processing power of web browser at the client-end has increased, client-side approach has become common as well. These two types of mashups have their advantages as well as disadvantages and suit for different situations, for instance a server-side mashup is not limited by browser's security model, the same origin policy, that isolates documents loaded from distinct origins from each other. Naturally hybrid approach combining server- and client-side mashup techniques is possible as well, and mashup developer can decide how to divide the functionality between the server and the client. If a mashup ecosystem consists of client-side mashups, it is necessary to pay more attention on how mashups can interact with services located at different origins. In addition, accessing specific mashup clients may be difficult because of addressing issues in IPv4-based networks.

Multiple and single API mashups. Instead of combining content from multiple APIs, which is usually the case, some mashups are using only one single API to create new visualization for existing web services. Often the user interface of this kind of mashups is simplified and added with attractive properties of some kind. Another kind of single API mashups provide more advanced ways for searching than the original service. For instance, there are numerous mashups that show images retrieved from the popular image service Flickr. Another example of a single API mashup is WikiMindMap (`http://www.wikimindmap.org/`), which generates a mindmap about a keyword based on Wikipedia articles. Mashup ecosystems that consist of numerous single API type of mashups are usually build around the few most popular services of the web. Such ecosystems have emerged, for instance, around Google Maps, Flickr, Wikipedia and Twitter.

4.2 Explicit Mashup Ecosystems

Commercial mashups. Commercial mashups are created to show a profit for the mashup publisher where as non-commercial mashups are provided non-profit. In commercial mashup ecosystems, mashup authors and service providers coordinate explicitly and use either specific contracts or common TOS agreements. Commercial mashup ecosystem is required to implement reliable and secure methods to in order to transfer sensitive data. In addition, availability of services in the ecosystem in a commercial setting is naturally vital. A typical example of a commercial mashup combines information about the product being sold with user reviews from multiple sources. Another type of commercial mashups is those including advertisements. Commercial mashups are targeted at consumers in contrast to enterprise mashups that are targeted at business users, even though both are often created by a company. It is common that a commercial mashup is provided for mobile device users as an alternative user interface for an electronic commerce. Further examples of commercial mashups are price comparison and product search mashups. For instance, there are numerous mashups offering this kind of service based on Amazon's and EBay's price data. Another kinds of commercial mashups help to locate a certain dealer on a map. An example of a commercial mashup combining social network services is Scupal (`http://www.scupal.com/`), a social buying website launched in India. Scupal allows users to select a product they would be willing to purchase, and then gather other interested buyers of the same product within their social networking contacts. The more there are buyers the less is the price.

Enterprise mashups. Enterprise mashups are developed to solve some particular business-related problem. In contrast to consumer mashups, that utilize only open web services, they can use closed enterprise data sources and combine the information with data from the web. Forming more closed ecosystem than the commercial mashup ecosystem, enterprise ecosystem is controlled by organization's internal interface specifications and descriptions, in addition to usage of public web services available under common TOSs. Security features of an enterprise mashup ecosystem are crucial as sensitive data of an organization is often handled. For instance, storing the data should be done in controlled fashion within the organization's own storage facilities. Enterprise mashups can be created solely by the company's IT department or a sand-box environment may be provided for non-experts to create mashups. However, the more degree of freedom is allowed, the greater are the skills needed for mashup development. Typical to enterprise mashups is that they focus is on a single presentation and target at providing a tool to help collaboration with different people working with the same objective.

Reusing of existing mashup solutions is often in a key role in an enterprise mashup ecosystem. One activity that targets at such reuse is Enterprise Mashup Markup Language (EMML), which is a XML-based domain specific language for developing enterprise mashups developed by the Open Mashup Alliance (OMA). With EMML, OMA aims at introducing a standardized, consistent and interoperable way to develop enterprise mashups. In addition to defining the language, OMA provides a reference implementation of a runtime that processes mashup scripts written in EMML. EMML can be used to declaratively describe the data processing flow, i.e. data composing, of a mashup.

4.3 Implicit Mashup Ecosystems

Situational mashups. Term *situational application* is used about an application that is created for a narrow group of users with unique needs, and some mashups are developed as situational applications. In Clay Shirky's essay *Situated Software* [12] this type of applications are described to be "designed for use by a specific social group, rather than for a generic set of 'users' ", and therefore, ecosystems build around situational mashups are implicit. Typically situational applications have short life span and the quality of engineering may not be first class. In addition, scaling up is often difficult with situational applications. However, Shirky remarked that as the group of users is relatively small, it is often unnecessary to implement mechanisms for user supervision.

Furthermore, situational applications are typically more personalized, and they can contain pre-entered information that is relevant only for the small group of intended users. As simple mashups that utilize readily available interfaces can be composed together rather quickly, the cost of implementation is relatively low, and the ecosystem containing situational mashups may have rather lightweight security, moderation and authentication features. Therefore, mashups can be targeted at small, specific groups of users and be very personalized, as well. The architecture and other engineering aspects of this kind of mashups may not be the most polished, but with the specific target group and purpose, it does not have resonance. One should bear in mind, however, that when mashups are used to address non-trivial, more complicated issues, this approach should not be used as it quickly leads to difficulties. Situational mashup ecosystems can emerge swiftly, but typically lifespans of such ecosystems are shorter as well.

The most essential API. One way to do the classification is to use the type of most essential API to determine the mashup type. For instance, a mashup can be classified as social, news, map, image, video, audio or search mashup based on the main service utilized. In consequence the ecosystem is build around this central service, and it can contain both implicit and explicit interactions. Often these mashups are targeted at consumers and provided for free, and therefore the implicit model is more common. Mashup statistics divided into categories based on the essential API used in a mashup can be collected from ProgrammableWeb site. The site provides statistics about mashups as well as service interfaces used to create new mashups. Only those mashups that are submitted to the website are listed, but the site can be used as a source for suggestive information about consumer mashups. However, the site does not list enterprise mashups at all. As can be seen in Fig. 2, mapping mashups are the most popular type of mashups. Social, search, photo, shopping and video mashups are roughly equally popular. In addition, remarkable number of mashups have been discontinued (tagged "deadpool").

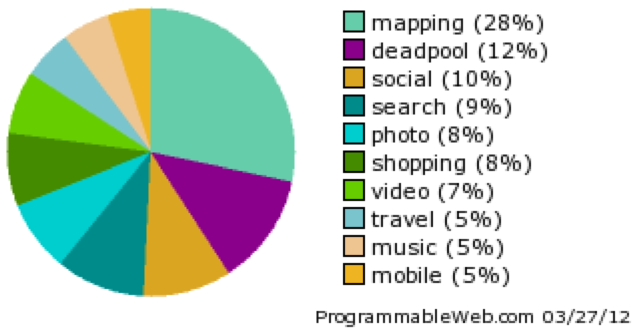

Fig. 2. Mashup types according to ProgrammableWeb (`http://www.programmableweb.com/mashups`).

5 Implementation Considerations

In our research (see e.g. [6, 7]), we have identified challenges that mashup ecosystem confronts in the areas of cloud infrastructure, web services, legal issues, and tool support. In the following, these will be briefly addressed.

Cloud Infrastructure. Cloud infrastructure related issues refer to addressability of mashup ecosystem endpoints as well as transparency and protocol support of network. Before IPv6 gains ground it might be necessary to use higher level methods to address network endpoints. Addressing mashup clients and services at too high level can derive scalability and performance issues. Another problem is caused by non-transparent network nodes that may cause some parts of the ecosystem become unattainable. Furthermore, lack of protocol support for other than HTTP can cause for instance video and audio streams fail to operate.

Web services. Web service reliability and complexity of integrating a high number of services are another type of challenges. Web service reliability can be addressed by adding fallback mechanisms, but this strategy will make the implementation more complex. While adding more services to the mashup can be attractive for users it makes the implementation more complex and increases vulnerability to service breakouts and incompatible version upgrades, which plague especially mashups that reuse services anonymously without explicit contracts. Furthermore, client-end device capabilities may be limited and operational expenses can be an issue, especially with mobile devices.

Legal issues. Moreover, legal issues related to mashup ecosystem are numerous. Service terms are often incompatible and hard to follow in complex mashups. The situation is even more complex when a mashup is hosted on third party platform, such as mashup tool providers servers. Mashups can be required to follow some content related limitations as well. For instance, some content can be freely available in U.S. but restricted from accessing in U.K. Some service providers may restrict their interfaces to be used only on desktop and prohibit using them on mobile devices. In addition, libraries and frameworks used in mashups may have conflicting licenses. Furthermore, protecting a client-side mashup from copying is often difficult as the executable code needs to be transferred to the client-end terminal.

Tools. Mashups can be developed with conventional web programming techniques using text editor and environment with debugging capabilities [13]. This requires considerable experience, as sometimes it is necessary to crawl the content from web pages. This is error-prone and can lead to hard-to-trace errors when subtle changes happen in the web sites from which content is downloaded. In contrast, dedicated mashup development tools can be helpful, especially when end-users are creating mashups. Typically, the target environment for dedicated tools is a web browser. Moreover, also the number of web sites from which content can be accessed is limited, and only few services are supported by the tools.

6 Conclusions

Web-based software and services have become commonplace. As virtually all imaginable content and services are becoming available online, there will be new, optimized ways to consume content and access services. In this paper, we have argued that mashups – special kinds of web applications that combine data and services from numerous sites – enables the development of new, improved applications that enrich the basic online facilities. A dimension that has been commonly overlooked with mashups is that they are not only about the technology, but their development and use is governed by other factors as well. Consequently the elements of opportunistic design – hacking, mashing and gluing, as pointed out in [14] – must be associated with the creation of sustainable software ecosystems of the web era.

Service interfaces are integral part of mashup ecosystems, and we indentified four levels of support that service providers can offer for mashups. We believe that the success of programmatic JavaScript interfaces is one indicator of the trend towards mashware ecosystems – software ecosystems that leverage source code and software components that are downloaded dynamically from all over the world. As pointed out in [9, 15], mashware ecosystems can dramatically improve productivity of web application development and allow global reuse of software components. However, research is needed in numerous areas including security, modularity and legal aspects, as well as software engineering methodologies to support the development of such ecosystems.

In the future, we expect that the multifaceted nature of mashups will lead to increasing interest also on the research side. So far, such applications, as well as associated ecosystems, have gained relatively little attention from researchers. Consequently, there are numerous directions for future work, where the main development principles of mashups in general as well as associated business impacts are analyzed in more detail.

References

1. Messerschmitt, D.G., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press, Cambridge, MA, USA (2003)
2. 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, Berlin, Heidelberg, Springer-Verlag (2011) 168–177
3. Yu, S., Woodard, C.J.: Service-oriented computing — icsoc 2008 workshops. Springer-Verlag, Berlin, Heidelberg (2009) 136–147
4. Hoyer, V., Stanoesvka-Slabeva, K., Janner, T., Schroth, C.: Enterprise mashups: Design principles towards the long tail of user needs. In: Services Computing, 2008. SCC '08. IEEE International Conference on. Volume 2. (july 2008) 601 –602
5. Bosch, J.: From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference. SPLC '09, Pittsburgh, PA, USA, Carnegie Mellon University (2009) 111–119

6. 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, Berlin, Heidelberg, Springer-Verlag (2011) 647–656
7. 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, New York, NY, USA, ACM (2010) 161–168
8. 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, New York, NY, USA, ACM (2010) 11:1–11:7
9. Taivalsaari, A.: Mashware: the future of web applications. Technical report, Mountain View, CA, USA (2009)
10. Mikkonen, T., Salminen, A.: Towards pervasive mashups in embedded devices. In: Proceedings of the 2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications. RTCSA '10, Washington, DC, USA, IEEE Computer Society (2010) 35–42
11. Salminen, A., Kallio, J., Mikkonen, T.: Towards Mobile Multimedia Mashup Ecosystem. In: IEEE International Conference on Communications Workshops, ICC Workshops. (2011)
12. Shirky, C.: Situated software. First published March 30, 2004 on the "Networks, Economics, and Culture" mailing list (2004)
13. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development. Internet Computing, IEEE **12**(5) (sept.-oct. 2008) 44 –52
14. Hartmann, B., Doorley, S., Klemmer, S.R.: Hacking, mashing, gluing: Understanding opportunistic design. IEEE Pervasive Computing **7**(3) (July 2008) 46–54
15. 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, New York, NY, USA, ACM (2010) 245–250