# Mashups and Modularity: Towards Secure and Reusable Web Applications

Antero Taivalsaari and Tommi Mikkonen
*Sun Microsystems Laboratories*
*P.O. Box 553 (TUT)*
*FIN-33101 Tampere, Finland*
*{antero.taivalsaari, tommi.mikkonen}@sun.com*

## Abstract

*The software industry is currently experiencing a paradigm shift towards web-based software. In this paper we argue that web technologies should allow developers to easily create application mashups that leverage components and other content from all over the world. In order to accomplish this, improvements are needed especially in two areas: security and modularity. In this paper we summarize our vision for the future of web applications, focusing especially on these two important areas.*

## 1. Introduction

The software industry is currently experiencing a *paradigm shift* towards web-based software. In the past few years, the Web has become a popular deployment environment for new software systems and applications such as word processors, spreadsheets, calendars and games. We believe that in the near future the vast majority of end-user software applications will be written for the Web, instead of conventional target platforms such as specific operating systems, CPU architectures or devices. In the new era of web-based software, applications live on the Web as services. They consist of data, code and other resources that can be located anywhere in the world. Furthermore, they require no installation or manual upgrades. Ideally, applications should also support user collaboration, i.e., allow multiple users to interact and share the same applications and data over the Internet.

Systems, tools and other facilities that enable web application development are often referred to collectively as "Web 2.0" technologies. Web 2.0 is mostly a marketing term, surrounded by a lot of hype, but there is real substance behind this somewhat nebulous term. Specifically, Web 2.0 technologies combine two important characteristics: *collaboration* and *interaction*. By *collaboration*, we refer to the "social" aspects that allow a large number of people to collaborate and share the same data, applications and services over the Web. An equally important, but publicly less often noted aspect of Web 2.0 technologies is *interaction*. Web 2.0 technologies make it possible to build web sites that behave much like desktop applications, for example, by supporting direct manipulation and allowing web pages to be updated one user interface element at a time, rather than requiring the entire page to be updated each time something on the page changes.

Even though Web 2.0 systems bring back some of the best qualities of desktop applications, such as direct manipulation and instant responsiveness, these systems are not simply about making the Web a better place for desktop applications such as word processors or spreadsheets. Rather, the best Web 2.0 applications typically leverage the potential of the users to produce their own content and share it with a large number of other users. Also, the most interesting applications leverage the possibility to combine content from multiple web sites. In web terminology, such content aggregation sites are commonly referred to as *mashups*.

In this paper we summarize our vision for the future of web applications. We argue that web technologies should move in a direction that allows developers to collaboratively create application mashups that leverage components and content generated by other developers around the planet. In general, just like the Web facilitates social interaction among its users, we believe that application development for the Web should occur in a collaborative, social fashion as well. However, there are still several obstacles to this. In this paper we focus on two areas – security and modularity – that currently make the Web an "anti-social" environment for developers.

The lack of a proper security model for applications makes it difficult to share and combine ("mash up")

content flexibly and securely. Even more importantly, proper modularity and information hiding mechanisms are needed on the Web to support the development of reusable content and components that can be utilized without reintroducing "spaghetti code" problems that have been solved elsewhere in the software industry already decades ago.

We argue that with some relatively straightforward improvements in these two areas, security and modularity, the software industry and the Web could be transformed to support truly social software systems and applications – not only in terms of the usage of these systems but also in terms of their development.

This paper draws upon our experiences in developing the *Sun Labs Lively Kernel* – an exceptionally flexible and interactive browser-based web programming environment written entirely in JavaScript (http://research.sun.com/projects/lively). We have summarized our experiences with JavaScript, and more generally, the use of the web browser as an application platform, in our earlier papers [1, 2].

The structure of this paper is as follows. In Section 2, we provide a general introduction to mashups, and provide some background on the tools and techniques used for mashup development today. In Section 3, we summarize the main problems in mashup development, focusing specifically on the two areas mentioned above. In Section 4, we define our suggested architecture and provide some recommendations for the future of web applications. Section 5 discusses related work, and finally, Section 6 concludes the paper.

## 2. Mashups and Mashup Development

In web terminology, a *mashup* is a web site that combines content from more than one source (from multiple web sites) into an integrated experience. Mashups are content aggregates that leverage the power of the Web to support worldwide sharing of content that conventionally would not have been easily accessible or reusable in different contexts or from different locations. Even though mashups are not necessarily full-fledged applications in the conventional sense, in this paper we use the terms 'mashup' and 'web application' interchangeably. Basically, we assume a mashup to be a web application that leverages the possibility to combine components from multiple web sites. Web applications are commonly also referred to as *Rich Internet Applications* (RIAs). However, we prefer the shorter term 'web application' instead of the longer (and often over-hyped) term.

Typical examples of mashups today are web sites that combine photographs or maps taken from one site with other data that is overlaid on top of the map or photo. Some of the archetypical mashups include the following:

- *Chicago Police Department crime statistics mashup* (http://chicago.everyblock.com/crime/). This site displays Chicago area crime statistics in various formats based on ZIP codes, dates, type of crime, etc.
- *Parking availability mashups* (e.g., http://www.parkingcarma.com/). This service displays the availability of parking spaces in various U.S. cities visually, and allows pre-reservation of parking spaces.
- *Traffic tracking and congestion mashups* (e.g., http://dartmaps.mackers.com/). Traffic tracking and congestion services are available for various cities throughout the world already. The web site mentioned above displays the location of commuter trains in the city of Dublin, Ireland.
- *Real estate sales and rental mashups* (e.g., http://www.housingmaps.com/). These mashups display houses or apartments for sale or rental, or provide information about recent real estate sales.

Mashups are by no means limited to maps or photos with overlays. In principle, the content can be anything as long as it can be meaningfully combined with other information available on the Web, e.g., price comparison information combined with product specifications, latest product news and user reviews or blogs. The key aspect is that the content must be available in a format that can be reused easily in other contexts. Textual representations such as HTML, XML, CSV (Comma-Separated Value format) or JavaScript source code, and standardized image and video formats such as GIF, JPEG, PNG and MPEG-4 play a crucial role in enabling the reuse of content in different contexts.

Mashups are often generated manually using normal web development technologies such as HTML, CSS (Cascading Style Sheets) and JavaScript. However, a number of automated tools are also available. These include (in alphabetical order):

- *Google Mashup Editor* (http://code.google.com/gme/)
- *IBM Project Zero* (http://www.projectzero.org/)
- *IBM QEDWiki* (http://services.alphaworks.ibm.com/qedwiki/)

- *Intel Mash Maker* (http://mashmaker.intel.com/)
- *LiquidApps* (http://www.liquidappsworld.com/)
- *Microsoft Popfly* (http://www.popfly.com/)
- *Open Mashups Studio* (http://www.open-mashups.org/)
- *Yahoo pipes* (http://pipes.yahoo.com/)

Many of these tools are still under development, reflecting the rapidly evolving state of the art in mashup development.

Despite the recent emergence of tools and the general interest and hype around web technologies, mashups are not really anything new. In software development, it has been a common practice or desire to build more advanced software systems out of prefabricated, reusable components developed by other software developers. The desire for reusable software components was first expressed by McIlroy and other participants of the NATO Software Engineering conference back in 1968 [3], and techniques for software reuse have been investigated for decades. However, mashup development differs from conventional software reuse in several important ways, addressed in the following.

First, 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.

Second, 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, static type checking and binary files[1]. This has created a trend towards more and more dynamic programming languages such as JavaScript, Perl, PHP and Python. 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].

Third, because of the increased focus on content rather than on implementation techniques, the 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. Consequently, they often are not aware of the benefits of well-established software engineering principles such as separation of concerns, modularity or information hiding.

Fourth, 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 exponentially 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.

In general, mashup development is still very much an *ad hoc* activity. In many ways, the techniques used for mashup development reflect the historical evolution of the Web from a document sharing environment (the original design center of the Web) towards a medium for increasingly interactive content. In the next section, we will take a more detailed look at the various trends and problems in mashup development today.

## 3. Problems in Mashup Development

The power of the World Wide Web stems largely from the absence of static bindings. For instance, when a web site refers to another site or a resource such as a bitmap image, or when a JavaScript program accesses a certain function or DOM attribute, the references are resolved at runtime without static checking. Apart from binary media files such as images and videos, the content on the Web is generally stored in textual form (e.g., in HTML or XML) instead of binary files. It is this dynamic nature that makes it possible to flexibly link and combine content from multiple web sites and, more generally, for the web to be "alive" and evolve constantly with no central planning or control.

---

[1] We often half-jokingly use the expression "*You cannot create mashups out of concrete bricks*" to highlight the difference to junior engineers and students.

However, there are a number of problems that encumber the development of reusable web sites today. In this paper, we focus specifically on two areas – modularity and security – because we feel that these areas most fundamentally constrain the evolution of the Web towards a full-fledged platform for web applications and mashups.

## 3.1. Lack of modularity, well-defined interfaces and information hiding

*Modularity* is a form of abstraction that allows systems to be composed of distinct parts, independently of implementation details. A *module* is generally defined to be a self-contained component of a system, which has a well-defined interface to the other components. Something is *modular* if it includes or uses modules which can be interchanged as units without disassembly of the module. The internal design of a module may be complex, but this is not relevant; once the module exists, it can easily be connected to or disconnected from the system.

A central aspect of a modular system is the presence of *well-defined interfaces*. A well-defined interface separate the specification of a component from its implementation details, allowing the implementation to be changed as necessary without impacting the external use of the component.

*Information hiding* is a principle that goes hand in hand with modularity and well-defined interfaces. Basically, in order to isolate design decisions and implementation-level issues from the external use of a component, the internals of the component must be hidden and preferably represented separately from the interface. This allows the implementation details to be changed later without impacting the external use of the component.

Modularity principles were introduced in the 1970s and 1980s by Parnas, Clements, Hoare, Liskov, Zilles and many others [4, 5, 6, 7, 8, 9, 10, 11, 12, 13].

**The importance of modularity principles from the "social" viewpoint**. Modularity principles have an important role in enabling software engineers to work with each other effectively in a "social", well-behaved fashion[2]. In the presence of well-defined interfaces, each engineer in a project can focus on the implementation details of those components that he or she is responsible for, without interfering with the work of the other engineers working on other components. In

this sense, modularity is a critical, enabling factor that allows large software systems to be developed collaboratively. Large systems that require dozens or hundreds of engineers, or projects that are built by a large number of teams in different geographic locations, simply cannot be built reliably without paying at least some attention to modularity. In fact, the less known the developers are to each other – a situation that is especially common in geographically distributed projects – the more important it is to have well-defined interfaces and commonly agreed specifications in place.

**Comments on modularity on the Web**. In principle, the absence of static bindings or binaries on the Web would not necessarily have to imply the lack of modularity, well-defined interfaces or information hiding. Unfortunately, today's web sites are rather unmodular "white boxes", or worse yet, "glass boxes", with their implementation details openly exposed to the outside world. The glass box approach is beneficial in the sense that it opens up the content of a web site to be leveraged by thousands of other sites and developers. However, it also makes such leveraged web sites extremely brittle, for there is no widely accepted way for a web site to publish a summary of its intended external interface separately from its implementation. Because of such limitations, web site developers today often resort to techniques such as obfuscation to prevent people from reverse engineering the implementation details (obfuscation is commonly used also for security reasons or to improve the download speeds of a web site.)

**The fundamental contradiction**. We have provided a more detailed analysis of the modularity problems on the Web in our previous article [14]. Here it suffices to say that there is a *fundamental contradiction* on the Web today with respect to modularity: Mashup developers (or more generally: web application developers) are usually unknown to each other. When a mashup developer reuses content from other sites, he or she often does not know the developers of those other sites in person. In such a setting, modularity would be especially critical, for it is nearly impossible to build well-behaved, maintainable systems unless all the elements of reuse have well-defined interfaces with proper information hiding. This simply isn't the case in web development today.

## 3.2. Security problems and challenges

The history of the web browser as a document viewing environment (as opposed to an application execution environment) is apparent when analyzing the

---

[2] One could argue that modularity is needed *the least* when there is only a single developer building and maintaining a system. The absence of modularity is an "anti-social" feature that discourages other developers from reusing a system.

restrictions and limitations that web browsers have in the area of networking and security. Many of these limitations date back to the conventions that were established early on in the design and historical evolution of the web browser. Some of the restrictions are "folklore" and have never been fully documented or standardized. In the following we provide a summary of the key problems based on our earlier analysis [2].

**1) The "Same Origin" networking policy is problematic**. A central security-related limitation in the web browser is the "same origin policy" that was introduced originally in Netscape Navigator version 2.0 (http://www.mozilla.org/projects/security/ components/same-origin.html). The philosophy behind the same origin policy is simple: it is not safe to trust content loaded from arbitrary web sites. When a document containing a script is downloaded from a certain web site, the script is allowed to access resources only from the same web site but not from other sites. In other words, the same origin policy prevents a document or script loaded from one web site ("origin") from getting or setting properties of a document from a different origin.

The same origin policy makes it difficult to build (and deploy) mashups or other web applications that combine content (e.g., news, weather data, stock quotes, traffic statistics) from multiple web sites. Special proxy arrangements are usually needed on the server side to allow networking requests to be passed on to external sites. Consequently, when deploying web applications, the application developer must be closely affiliated with the owner of the web server in order to make the arrangements for accessing the necessary sites from the application.

**2) No namespace isolation is available to safeguard applications from cross-site scripting issues**. A common security problem in web application development today is *cross-site scripting* (XSS). Cross-site scripting is a type of security vulnerability that arises from the fact that today's browsers and scripting engines do not provide proper namespace isolation to safeguard scripts or other active content loaded from one site from the scripts or active content loaded from other sites. Since the scripts share the same JavaScript namespace (and the DOM tree) at runtime, it would be possible – in the absence of the same origin policy – for scripts from one site to interfere with content loaded from other sites, collect private data that should not be visible to external users, or inject malicious scripts into content loaded from other sites. Vulnerabilities of this kind have been exploited to craft powerful phishing attacks and other browser exploits. The possibility of cross-site scripting vulnerabilities is

the reason why the same origin policy restrictions discussed above were originally introduced. An excellent summary of cross-site scripting problems is available at the following web site: http://www.cgisecurity.com/articles/xss-faq.shtml. A good summary of the issues is also provided in [15].

**3) Limited access to host platform capabilities or APIs**. Web documents and scripts are usually run in a sandbox that places various restrictions on what resources and host platform capabilities the web browser can access. For instance, access to local files on the machine in which the web browser is being run is not allowed, apart from reading and writing a limited amount of data in cookies. Among other things, the sandbox security limitations prevent a malicious web site from altering the local files on the user's local disk, or from uploading files from the user's machine to another location.

The sandbox security limitations of the web browser make it difficult to build web applications that utilize local resources (e.g., the local file system) or other host platform capabilities. Consequently, it has been nearly impossible to write web applications that would, e.g., be usable also in offline mode without an active network connection. These problems are gradually being solved with the introduction of libraries such as WebDAV [16] and Google Gears (http://gears.google.com/). For instance, for our own Lively Kernel system, we have implemented a storage solution using WebDAV.

**Towards a more fine-grained security model**. The key point in all the limitations related to networking and security is that there is a need for a more fine-grained security model for web applications. On the Web today, applications are second-class citizens that are on the mercy of the classic, "one-size-fits-all" sandbox security model of the web browser. Decisions about security are determined primarily by the site (origin) from which the application is loaded, and not by the specific needs of the application itself. Even though some interesting proposals have been made [15, 17, 18, 19, 20], currently there is no commonly accepted finer-grained security model for web applications or for the Web more generally.

## 4. Towards Secure and Reusable Web Applications

It is not difficult to define the "ideal" architecture for web applications. Our ideal environment would support highly interactive, visually rich desktop-style applications that can be built out of pre-existing components that can be loaded from anywhere on the

Web on an on-demand basis. The components would be published with well-defined interfaces (i.e., each site would publish a well-defined "contract" to the outside world), and would be delivered in a platform-independent format that does not require any static linking or binding (e.g., in source code format or in some portable intermediate format). Application execution would take place in the web browser (preferably without any plug-in components), enhanced with a security model that allows application components to be downloaded from anywhere on the Web. Application communication with the web server would take place asynchronously, without blocking the user interface. No installation or upgrades would be required, since the applications and all the necessary components would be loaded dynamically from the Web.

This kind of an environment would make it possible for application developers and software engineers to collaborate on an immensely large scale, allowing unparalleled sharing and reuse of software, data, layout and visualization information, or any other content across the planet. Applications would be truly *web* applications, consisting of components that are loaded dynamically from those web sites that provide the most applicable components for each purpose. If such massive-scale reuse were possible, the productivity of software development could potentially be improved by an order of magnitude or more. The Web could be the enabling factor that would finally make large-scale software reuse a reality rather than just a perpetual dream.

What is missing from this vision? As we have summarized in our earlier papers [1, 2, 14], there are still numerous problems related to web application development. Many problem areas (e.g., usability and browser compatibility issues) are outside the scope of this paper. Architecturally, the following features would be needed:

- Modularity support and proper interfaces with information hiding.
- A mechanism to document and publish application interfaces (more generally: the public interfaces of a web site) in a standardized format.
- A more fine-grained browser security model that provides controlled access to security-critical APIs and host platform resources, as well as allows applications to download components flexibly from various sites.
- An execution engine inside the web browser that supports namespace isolation and modularity to allow content from different sites to run securely.

In principle, technologies for all these areas are already available. For instance, modularity mechanisms and interface description languages have been investigated for decades. In the context of the Web, technologies and protocols such as REST [21, 22], WSDL (http://www.w3.org/TR/wsdl) and SOAP (http://www.w3.org/TR/soap) are gradually making it possible to specify and use the interfaces of web sites in a portable and reusable fashion. Fine-grained security models and namespace isolation have been studied extensively, e.g., in the context of the Java Platform, Standard Edition (Java SE) [23] or in the Java Platform, Micro Edition (Java ME). The latter platform has a lightweight, permission-based, certificate-based security model [24] that could potentially be applicable also to web application development.

In general, the main problems in the areas discussed above are not really technological. The key problem is related to retrofitting proper modularity and security mechanisms into an architecture that was not really intended to be a full-fledged software platform in the first place. Standardization is another major problem, since it is difficult to define a security solution that would be satisfactory to everybody while retaining backwards compatibility with the existing solutions. Also, many security models depends on application signing and/or security certificates; such solutions usually involve complicated business issues, e.g., related to who has the authority to issue security certificates. Therefore, it is likely that any resolutions in this area will still take years. Meanwhile, a large number of security groups and communities – such as the Web Application Security Consortium (WASC), the Open Web Application Security Project (OWASP), and the W3C Web Security Context Working Group – are working on the problem.

## 5. Related Work

Mashup development has become a very popular topic in the past few years. However, there are still surprisingly few academic research articles on mashups, especially in areas related to mashup security and modularity or reusability. Here we highlight some of the most interesting proposals.

**The '<module>' tag for secure mashup communication**. Doug Crockford has proposed a new HTML tag with the following syntax [17]:

```
<module id="NAME" src="URL" style="STYLE" />
```

The '`<module>`' tag allows the web browser to bypass the limitations of the same origin policy by allowing scripts from different sources to communicate with each other in a secure and controlled fashion. Communication is allowed between scripts from different origin in the form of JSON (JavaScript Object Notation) objects, i.e., by passing along serialized objects or other data in textual form. Such messages are exempt from the same origin policy.

**Microsoft's MashupOS proposal for fine-grained mashup security**. Microsoft's MashupOS proposal [19] introduces a more fine-grained approach to mashup security, providing different types of access for four types of content or "trust relationships": isolated content, access-controlled content, open content and unauthorized content. Special abstractions are proposed for the currently unsupported trust relationship types.

**Google Caja.** Caja (http://code.google.com/p/ google-caja) allows web applications to run scripts safely in third-party content, while relying on existing web standards. Caja defines a subset of JavaScript that can be used as an object-capability language. Normal (unsafe) JavaScript programs are translated into this safe subset before deployment and execution. When downloaded and run, the resulting code will enforce its object-capability security model.

**Other proposals for web application security**. In their WWW'2007 Conference paper, Jim, Swamy and Hicks [15] proposed the use of a special "security hook": a piece of code embedded in web applications that will be executed in a suitably-modified web browser before executing any other script. When instantiated with a server-provided whitelist or sandbox policy, the hook function can remove malicious scripts before execution.

Jackson and Wang's Subspace proposal [20] introduces a cross-domain communication mechanism that allows efficient communication across domains while sandboxing untrusted content. The Subspace proposal is somewhat similar to Doug Crockford's proposal, except that the Subspace concept is intended to be implementable in existing web browsers without any browser modifications or enhancements to HTML.

Another interesting proposal is the Tahoma web browsing system by Cox, Hansen, Gribble and Levy [25]. That system introduces a new trusted software layer on which web browsers should execute. In the Tahoma architecture, each web application is isolated and virtualized within its own virtual machine sandbox, removing the need to trust web browsers and the services they access. However, like many other proposals, the Tahoma architecture requires changes (in this case major changes) to the web browser, as well

as restricts the set of sites with which each web application can communicate.

For a good overall summary on mashup security, the reader is referred to the white paper at http://www.openajax.org/whitepapers/Ajax%20and%2 0Mashup%20Security.php.

**Proposals for web site interface description**. There does not seem to be a lot of academic research in the area of interface description languages for web applications. However, standardization activities and industrial efforts abound. Most of the work revolves around XML, with the general idea that web sites can publish a summary of the services that they offer in the form of XML descriptions that can be parsed by client systems with an XML parser. Perhaps the best known language used for such descriptions is the *Web Services Description Language* (WSDL) (http://www.w3.org/TR/wsdl). WSDL is an XML format for describing network services as a set of messages containing either document-oriented or procedure-oriented information. The clients will then typically use other XML-based protocols such as SOAP to communicate with the services. However, because of the same origin policy restrictions in web browsers today, it is difficult to use such technologies in the creation of mashups that could flexibly combine content and functionality from multiple services.

# 6. Conclusions

For better or worse, the World Wide Web is rapidly becoming the platform of choice for end-user software applications. Web-based applications have major benefits: in particular, they require no installation or manual upgrades, and they can be deployed instantly worldwide. The transition towards web-based applications and mashups will turn the web browser into a very important target platform for software applications. As a consequence, software developers will increasingly write software for the Web rather than for a specific operating system or hardware architecture.

In this paper we have summarized our vision for the future of web applications. We argued that web technologies should move in a direction that allows developers to collaboratively create application mashups that leverage components and content generated by other developers around the planet. In order to support such collaborative development, the security model of the Web needs to evolve in a direction that allows content to be shared and combined flexibly and securely. Furthermore, we argued that proper modularity and information hiding

mechanisms are needed on the Web in order to support the development of truly reusable content and components. In the absence of modularity and a more flexible security model, the Web is still an "anti-social" environment for application developers. We suggested that with some relatively straightforward improvements in these two areas – security and modularity – the software industry and the Web could be transformed to support truly social software systems and applications – not only in terms of the usage of these systems but also in terms of their development.

# References

[1] Mikkonen, T., Taivalsaari, A., Using JavaScript as a Real Programming Language. Sun Microsystems Laboratories Technical Report TR-2007-168, October 2007.

[2] Taivalsaari, A., Mikkonen, T., Ingalls, D., Palacz, K., *Web Browser as an Application Platform: The Lively Kernel Experience*. Sun Microsystems Laboratories Technical Report TR-2008-175, January 2008. To appear in the SEAA'2008 Conference Proceedings (34th Euromicro Conference on Software Engineering and Advanced Applications, Parma, Italy, September 3-5), 2008.

[3] McIlroy, M.D., Mass produced software components. In Naur, P., Randell, B. (eds): *1968 NATO Working Conference on Software Engineering* (Garmisch, Germany, October 7-11), 1968, pp. 88-98.

[4] Parnas, D.L., A technique for software module specification with examples. *Communications of the ACM,* Vol. 15, No. 5, May 1972, pp. 330-336.

[5] Parnas, D.L., On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, Vol. 15, No. 12, Dec. 1972, pp. 1053-1058.

[6] Dahl, O-J., Dijkstra, E.W., Hoare, C.A.R., *Structured Programming*. Academic Press, 1972.

[7] Zilles, S.N., Procedural encapsulation: a linguistic protection technique. *ACM SIGPLAN Notices*, Vol. 8, No. 9, Sep. 1973, pp. 142-146.

[8] Liskov, B.H., Zilles, S.N., Programming with abstract data types. *In Proceedings of ACM SIGPLAN Conference on Very High Level Languages, ACM SIGPLAN Notices*, Vol. 9, No. 4, Apr. 1974, pp. 50-59.

[9] Liskov, B.H., Zilles, S.N., Specification techniques for data abstractions. *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, Mar. 1975, pp. 7-19.

[10] Parnas, D.L., On the design and development of program families. *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, Mar. 1976, pp. 1-9.

[11] Guttag, J., Abstract data types and the development of data structures. *Communications of the ACM*. Vol. 20, No. 6, Jun 1977, pp. 396-404.

[12] Parnas, D.L., Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 2, Mar. 1979, pp. 128-137.

[13] Parnas, D.L., Clements, P.C., Weiss, D.M., Enhancing reusability with information hiding. *In Proceedings of the ITT Workshop on Reusability in Programming* (Newport, Rhode Island, September 7-9), 1983, pp. 240-247.

[14] Mikkonen, T., Taivalsaari, A., Web Applications: Spaghetti code for the 21st century. Sun Microsystems Laboratories Technical Report TR-2007-166, June 2007. To appear in the SERA'2008 Conference Proceedings (6th International Conference on Software Engineering Research, Management and Applications, Prague, Czech Republic, August 20-22), 2008.

[15] Jim, T., Swamy, N., Hicks, M., Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. *In Proceedings of the 16th International Conference on World Wide Web* (Banff, Canada, May 8-12), 2007, ACM, pp. 601-610.

[16] Dussealt, L., *WebDAV: Next-Generation Collaborative Web Authoring*. Prentice-Hall Series in Computer Networking and Security, 2003.

[17] Crockford, D., The <module> Tag: A Proposed Solution to the Mashup Security Problem. http://www.json.org/module.html, October 30, 2006.

[18] Yoshihama, S., Uramoto, N., Makino, S., Ishida, A., Kawanaka, S., De Keukelaere, F., Security Model for the Client-Side Web Application Environments. *IBM Tokyo Research Laboratory presentation*, May 24, 2007.

[19] Wang, H.J., Fan, X., Howell, J., Jackson, C., Protection and Communication Abstractions for Web Browsers in MashupOS. *In Proceedings of the 21st ACM Symposium on Operating Systems Principles* (Stevenson, WA, USA, October 14-17), 2007, pp. 1-16.

[20] Jackson, C., Wang, H., Subspace: Secure Cross-Domain Communication for Web Mashups. *In Proceedings of the 16th International Conference on World Wide Web*, (Banff, Canada, May 8-12), 2007, ACM, pp. 611-619.

[21] Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California at Irvine, CA, USA, 2000.

[22] Fielding, R.T. and Taylor, R.N. Principles design of the modern web architecture. *ACM Transactions on Internet Technology*, Vol. 2, No. 2, May 2002, pp. 115-150.

[23] Gong, L., Ellison, G., Dageforde, M., *Inside Java™ 2 Platform Security: Architecture, API Design, and Implementation* (2nd Edition). Addison-Wesley (Java Series), 2003.

[24] Riggs, R., Taivalsaari, A., Van Peursem, J., Huopaniemi, J., Patel, M., Uotila, A., *Programming Wireless Devices with the Java™ 2 Platform, Micro Edition* (2nd Edition). Addison-Wesley (Java Series), 2003.

[25] Cox, R.S., Hansen, J.G., Gribble, S.D., Levy, H.M., A Safety-Oriented Platform for Web Applications. *In Proceedings of the IEEE Symposium on Security and Privacy*, IEEE Computer Society, 2006, pp. 350-364.