Next in Mashup Development: User-Created Apps on the Web

Florian Daniel, *University of Trento*
Maristella Matera, *Politecnico di Milano*
Michael Weiss, *Carleton University*

Mashups have relatively simple, component-based development paradigms, yet few end users develop their own applications. To help turn end users into developers and innovators, the authors present two mashup platforms for lightweight Web development practices and discuss open challenges.

Since the beginning of the millennium, the Web has evolved from a mere one-way communication medium dominated by developers and information providers (Web 1.0) into a fully distributed and democratic communication platform that equally involves developers, information providers, and consumers (Web 2.0). Many factors contributed to this evolution, yet two clearly stand out: the emergence of the service-oriented architecture (SOA) and the success of social applications. The former enabled unprecedented interoperability among applications, and the latter, unprecedented interoperability among people. As such, SOA represents a technological dimension, while social applications represent a societal dimension. Both dimensions will affect future Web applications.

Emerging technologies are already reshaping the landscape of today's Web development practices. Cloud computing, with its elastic hardware resources, is radically changing how developers architect Web applications to cope with varying workloads. Software as a service (SaaS) is changing how applications are distributed and consumed, and HTML 5 (with its Web sockets) is turning traditional client-server Web architectures into full-fledged, distributed programming environments. Furthermore, increasingly sophisticated mobile devices—smartphones and tablet PCs—are making ubiquitous access commonplace.

Similarly, societal changes are reshaping how consumers act on the Web. Ten years ago, the average Web user could barely navigate through complex Web applications. Today, users actively contribute to the Web's success through user-contributed content such as reviews, opinions and ratings, tags, and status updates.

 Published by the IEEE Computer Society

Here, we discuss a particular set of technological and societal trends—Web mashups[1] and user innovation.[2,3] Together, these lead to a novel development paradigm in which end users and developers co-develop applications. We expect such support for end-user development to be prominent in future Web development practices. Mashup tools, or platforms that simplify mashup development, are already common; they just typically fall short of adequate end-user support.

Toolkits for User Innovation

In a traditional design-build-evaluate product life cycle, developers don't collect user feedback until after they've developed a prototype, at which point changes are costly. In user-driven product innovation, a company offers users an *innovation toolkit* that lets them build their own products.[2,3] The toolkit provides a constrained interface to the capabilities of the company's product platform. In particular, it ensures that new products are properly constructed.

An innovation toolkit aims to let the user carry out the iterative experimentation needed to develop a new product. Many users can work in parallel on solving a problem, focusing on their own need for a solution. They can create a solution that closely meets their needs and can obtain feedback quickly through their development experiments. The company providing the toolkit doesn't carry the cost of failed experiments, but if an experiment ends up adding significant value for users, the company can integrate the user innovation back into its core products. On the Web, this is similar to Google monitoring use of its public APIs (such as Google Maps and Google Search) and incorporating the best innovations to fine-tune the APIs.[4]

Opening services for integration in mashups is thus a strategic choice that revolutionizes the business model that for years has characterized the Web and its applications. Rather than being passive receivers of innovation, Web users can become actively involved in the innovation process. Their desire and ability to extend the functionality of products they own—to realize their ideas and express their creativity—can help drive future mashups.

The Mashup Development Scenario

How mashups are developed depends on their type. Current *consumer mashups*—for example, mashups based on Google Maps—are mainly clever hacks by expert developers. *Enterprise mashups*, on the other hand, highlight much more diverse development application scenarios and are more interesting in terms of revealing underlying development practices and how traditional development processes should evolve to cope with the new paradigm.

We reviewed recent studies of such enterprise mashups,[5,6] noting the contributions of different actors and their skill levels. We identified two main scenarios, which differ in the heterogeneity of services to be combined, the diversity of user needs, and the sophistication of either the involved actors or the tools supporting their work.

Mashup tools are already common; they just typically fall short of adequate end-user support.

Figure 1a shows the first scenario, in which expert developers (such as IT programmers, service providers, or sophisticated users) create mashups centrally, exploiting ready-to-use internal or external resources to deliver applications quickly. End users aren't directly involved in constructing such mashups, but they benefit from the shorter turnaround time for new applications.

Figure 1b shows the second scenario, in which the users create the mashups in a "distributed" fashion, starting from a set of ready services. Such services can be developed internally—purposely created according to the final users' needs—or achieved by wrapping public services. In this scenario, users close to the application domain construct the mashups to fulfill a specific short-term, situational need.[7] For example, an enterprise manager might compose his or her own dashboard. There's a wide range of corporate services (such as those that provide access to enterprise information sources), Web resources, and open services that, if integrated together, would simplify the construction of applications for process and data analysis. These mashup applications constitute the "long tail" of applications and usually aren't implemented in the central development scenario,[8] which means that many users' needs, though modest, aren't being met.

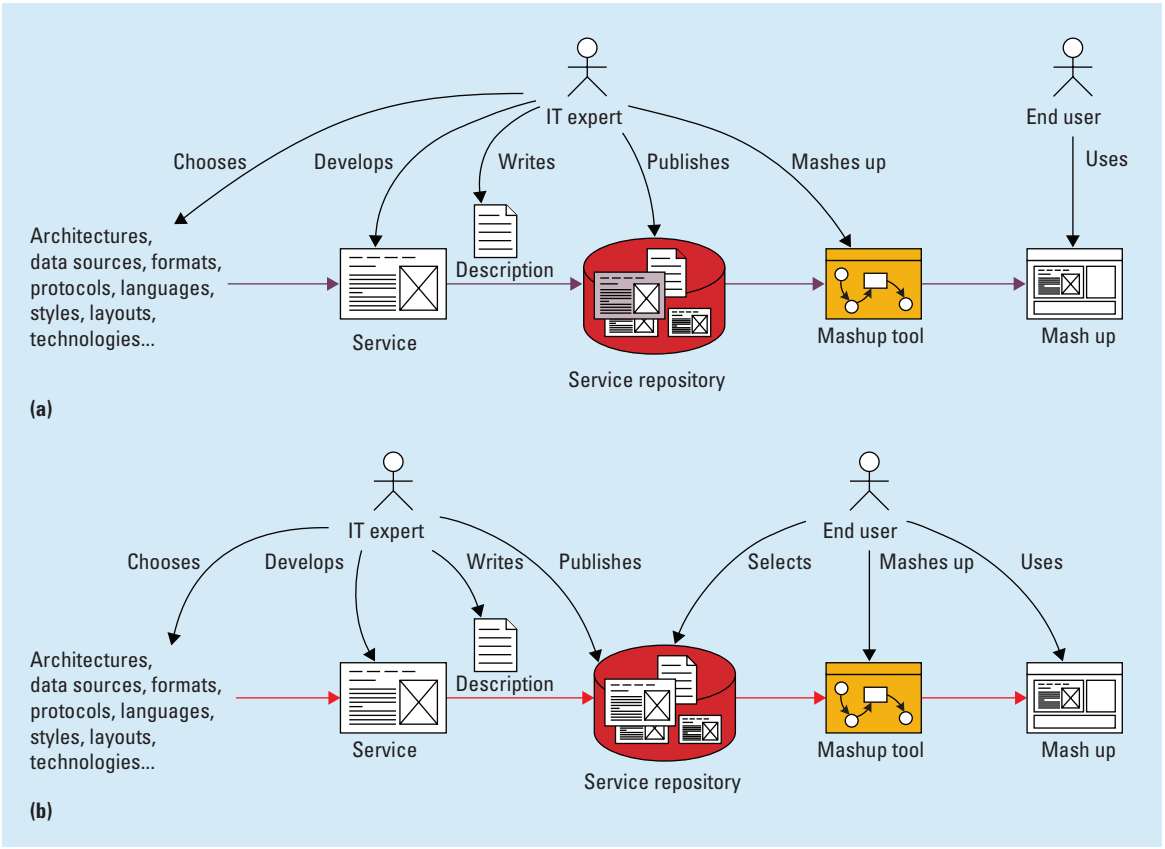

Figure 1. The two main mashup development scenarios. (a) Expert developers exploit mashup tools "centrally" to deliver applications quickly. (b) Users exploit such tools to create mashups in a "distributed" fashion, starting from a set of ready services. (The red arrows indicate when the artifacts come into play during mashup development.)

Creating a tool for this second scenario would be challenging but could pay off significantly by helping users combine services and data to create their own mashups. An experiment conducted to assess user experiences during a project that let enterprise analysts and managers flexibly construct dashboards revealed that this development paradigm is effective and increases end-user satisfaction.[9]

The two scenarios also differ in the degree of control over the mashup's quality. In the first scenario, the IT department fully controls what kind of mashup is being developed, ensuring the quality of those mashups. However, not all end users need applications with stringent security, performance, or reliability requirements; they might want an application only for a specific purpose, so a complex solution developed by the IT department would be too costly. Although the second scenario doesn't guarantee the quality of the final applications, it allows for a greater flexibility with respect to the user needs and promotes innovation.

Other researchers have similarly classified the various roles in mashup development,[8,10] distinguishing between three types of users: professional developers, consultants and sophisticated users, and end users. These users work at different levels of complexity, using tools appropriate for their level. Developers expose existing enterprise applications and data sources through APIs that provide the basic mashup components. Consultants and sophisticated users combine APIs into user interface widgets or API combinations that can be reused as building blocks for end-user mashups. Finally, end users configure and use mashups and also create simple mashups.

Lightweight Development Processes

The life cycle of Web applications is typically more dynamic than that of other classes of software, because prototypes and a final application must be developed in Internet time—that is, in days or weeks instead of months or years. Additionally, the possibility of logging usage data for hundreds to millions of users leads to more

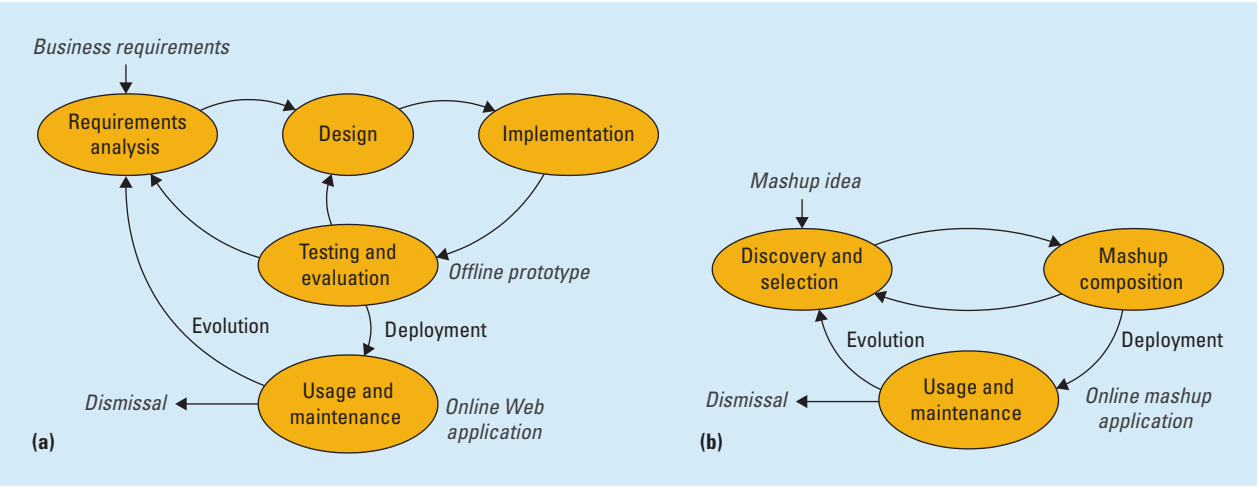

Figure 2. Life-cycle models of (a) current Web applications and (b) mashups. The mashup model assumes availability of a dedicated mashup platform and toolkit, along with a set of open Web services that support features and available data.

advanced testing and usability analyses. Finally, once an application has been deployed, evolutions and improvements are applied while the application is actually online and in use. In other words, the application undergoes continuous online evolutions.

Development for the Web thus naturally spans two main stages: the incremental *development* of the application's base version and its post-deployment, incremental *evolution*. Such a development process is oriented toward professional programmers and big software projects and thus goes well beyond the skills of average mashup composers (see Figure 2a).

The ideal mashup development process should reflect the innovation potential of mashups: to compose an application, starting from given content and functionality that addresses personal needs, and run it without worrying about what happens behind the scenes. The prototype-centric and iterative approach is accentuated: the composer mashes up services and runs the result to check whether it works. In case of unsatisfactory results, the composer fixes the problems and runs the mashup again. Given the situational nature of mashup applications, the role of application stakeholders must be put into perspective: requirements indeed correspond to the (short-lived) needs of the mashup composer. We summarize these considerations in a *lightweight development process* model that comprises three main activities (see Figure 2b).

Discovery and Selection

The mashup composer starts with an idea that addresses personal needs and preferences and then selects source services that can provide the necessary data, application logic, or user interfaces. In most cases, these are open services available on the Web.

Discovery and selection is a new life-cycle activity for mashup applications. It precedes mashup composition and implicitly incorporates requirements analysis and specification, because the idea itself is an informal expression of the application requirements. The selected mashup components represent these requirements in terms of enabling services, proving the idea's feasibility and providing a draft of the final mashup's organization.

Mashup Composition

Dedicated mashup platforms can help less-skilled Web users visually compose the selected components and set up the composite application's integration logic and layout. The platforms must base the integration logic on intuitive formalisms and models, expressed in domain-specific languages, which in most cases will be hidden behind graphical modeling notations. The platforms can also help with composition by recommending compatible services for improved mashup quality,[11] presenting composition patterns that have been successful in the past,[12] or compiling or automatically connecting services on the user's behalf.[9]

Mashup composition simplifies traditional design and implementation activities by eliminating the need for cornerstone activities (such as hypertext design) that have long characterized the development of document-centric Web applications. Deployment just requires saving the

mashup application on a server for the hosted execution (a one-click activity).

Usage and Maintenance

Once composed, the mashup must be immediately executable online. Note that to eliminate the deployment task, which would be beyond most end users' capabilities, the mashup platforms must support hosted solutions for both development and execution (which is already partly in practice). Consequently, we view mashups as applications whose life cycle naturally starts from the deployment point in Figure 2a and whose development occurs via incremental evolutions. Indeed, once saved, mashups are immediately online, so there aren't any incremental development cycles.

After deployment, the mashup composer shares application maintenance with the platform provider: the composer fixes problems in the composition logic, while the provider fixes problems in components and the hosted execution environment.

Enabling a larger class of users to compose their own mashups and innovate requires intuitive development tools.

The mashup usage and maintenance phase incorporates the traditional test-and-evaluation tasks. By running the mashup, the composer can easily check whether the application works and satisfies his or her needs, while at the same time collecting feedback from other users. Application evolution then requires simply starting the mashup process anew (from service discovery and selection).

Mashup Tools

So how do we enable even less-skilled Web users to develop their own mashups? A mashup composer can always use a conventional programming language to mash up the components of his or her choice. Given the heterogeneity of components, programming languages, and interaction protocols, and the complexity of the necessary integration logic, only highly skilled programmers can manually develop mashups—and even they might have a hard time mastering all the development challenges. Service composition approaches (such as those using BPEL, the Business Process Execution Language) can't cope with the heterogeneity of technologies and are still rather complex.

In line with the end-user development vision, enabling a larger class of users (not just skilled developers) to compose their own mashups and innovate requires the availability of intuitive development tools and a high level of assistance.[13] There's a considerable body of research on mashup tools (mashup makers), typically featuring easy-to-use GUIs and drag-and-drop paradigms for combining mashup components. However, such tools are suited only for certain development tasks and often don't provide the integrated development paradigms, instruments, and languages necessary for helping nonprogrammers integrate heterogeneous components. For example, Yahoo Pipes (http://pipes.yahoo.com) focuses on data integration via RSS or Atom feeds and offers a data-flow composition language, but it doesn't support the integration of user interfaces. Furthermore, very few tools support integration at all three layers characterizing Web applications: the data, application, and presentation (user interface) logics.

Defining environments based on lightweight development processes is the object of our research on the agile, mashup-based development of Web applications.[14] Our work concentrates on identifying abstractions and composition paradigms that can hide the technical details of the composition logics, thus easing mashup development. We've developed two mashup platforms, accommodating different development scenarios.

MashArt

MashArt offers a universal integration approach for handling components as varied as simple RSS feeds, SOAP or RESTful Web services, and user interface components.[15] It addresses development scenarios in which IT experts need easy-to-use tools to quickly produce mashups. As Figure 3 shows, mashup development uses a hosted, Ajax-based visual editor for graph-based composition. Mashup composers "draw" their mashup logic by specifying event and data flows among mashup

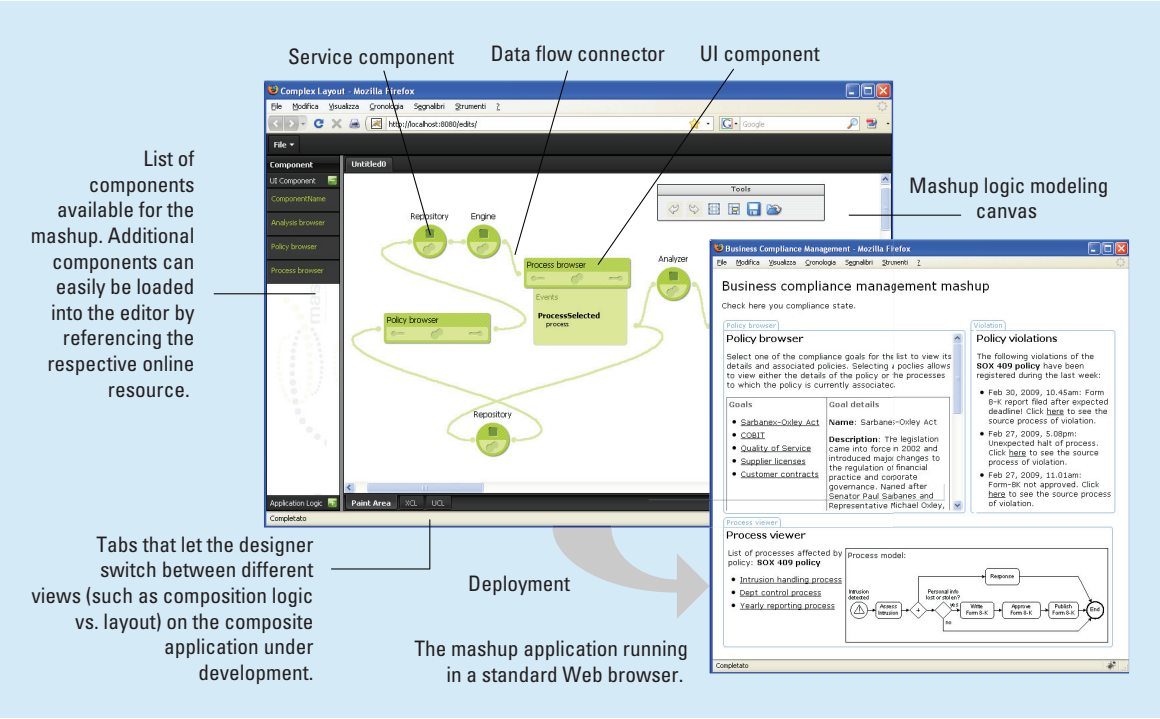

Figure 3. MashArt fosters universal compositions to address development scenarios in which IT experts need easy-to-use tools to quickly produce mashups. The two screenshots show the design and resulting mashup of a business compliance management application.

components: event-operation couplings synchronize user interfaces, while data flows enable service orchestration. The mashArt integration platform hosts mashup specifications and interprets them during mashup execution.

Given its modeling approach, which lets users fine-tune components and services, mashArt is probably more suited to assist the IT department in the first development scenario we presented (Figure 1a).

DashMash

Based on the same event-driven mashup paradigm for composing user interfaces,[14] the DashMash tool provides a sandbox environment,[9] where inexperienced users can easily define mashups through an intuitive drag-and-drop development paradigm. As Figure 4 shows, users select components from a visual menu and move them into a composition canvas. DashMash instantly interprets composition actions and executes the resulting mashup in a WYSIWYG (what you see is what you get) style. The tool guides the composition task in multiple ways. The composition engine creates default bindings between components, using compatibility rules automatically inferred from component descriptors. The same compatibility rules also suggest additional bindings, which users can define through a form-based mechanism that abstracts from technical details (see Figure 4).

DashMash steps into our second scenario (from Figure 1b), hiding the underlying model and proposing direct visual feedback to its users.

Over the last few years, research on mashups has concentrated on enabling technologies, languages, and protocols. However, to effectively turn end users into developers and enable user innovation, we need to devote more effort to less technology-specific research challenges. For example, we need the following:

- *intelligible composition paradigms:* mashup tools must abstract from technical details, leveraging models that hide the complexity of the technology heterogeneity characterizing the plethora of resources available for mashup and for composition logics;
- *domain-specific platforms:* for users to fully understand the possibilities of a mashup platform,

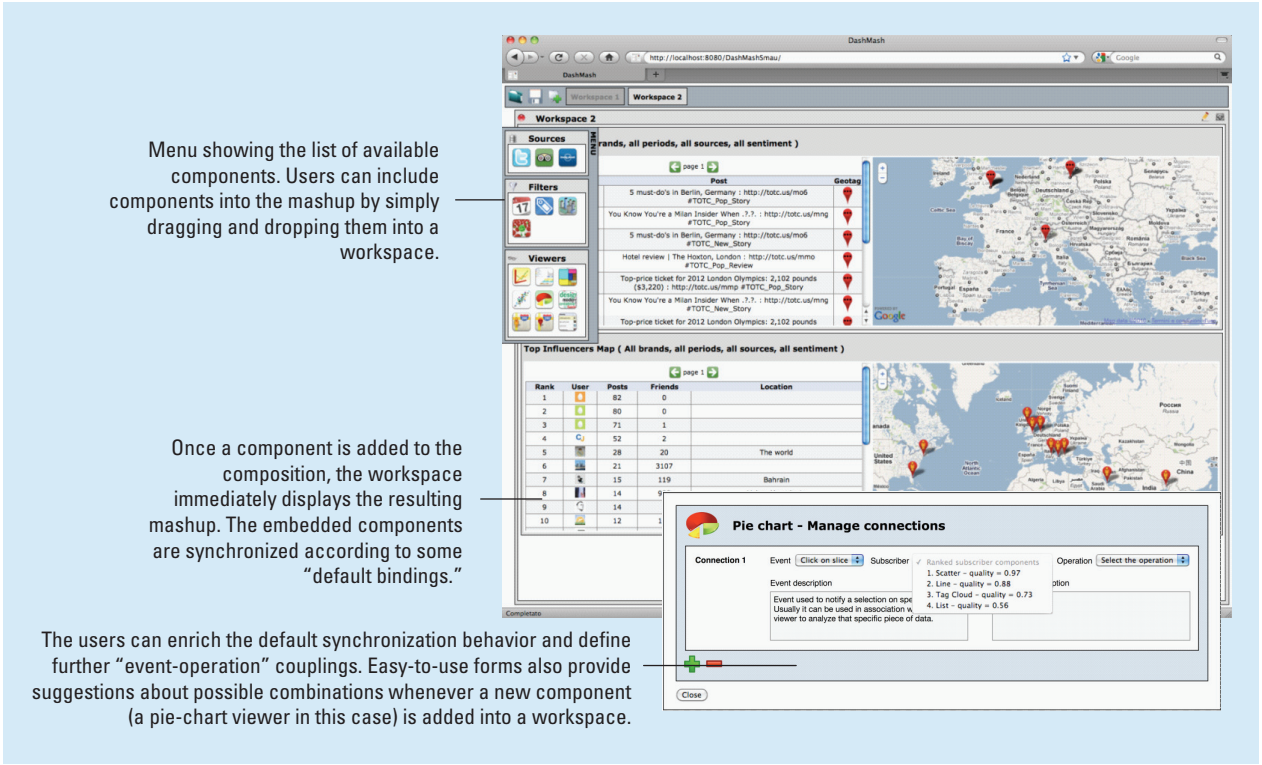

Figure 4. DashMash supports inexperienced end users by offering a sandbox environment characterized by default bindings and visual mechanisms for component synchronization. The screenshot shows the mashup of sentiment-analysis dashboards, which provide company analysts with indicators that summarize user opinions about entities of interest extracted from the Web.

we must tailor the platforms to well-defined domains familiar to the users—so the tools "speak the language of the user";

- *assisted composition:* we need automatic recommendations on mashup quality[11] and knowledge reuse[12] to teach users how to "speak the tool's language" and develop applications.

Only the right coupling of technical solutions with effective end-user development paradigms will turn user innovation into practice, yielding novel, lightweight Web development practices.

References

1. J. Yu et al., "Understanding Mashup Development," *IEEE Internet Computing*, vol. 12, no. 5, 2008, pp. 44–52.
2. S. Thomke and E. von Hippel, "Customers as Innovators: A New Way to Create Value," *Harvard Business Rev.*, vol. 80, no. 4, 2002, pp. 74–81.
3. M. Weiss and G.R. Gangadharan, "Modeling the Mashup Ecosystem: Structure and Growth," *R&D Management*, vol. 40, no. 1, 2010, pp. 40–49.
4. B. Iyer and T.H. Davenport, "Reverse Engineering Google's Innovation Machine," *Harvard Business Rev.*, vol. 86, no. 4, 2008, pp. 58–69.
5. A. Jhingran, "Enterprise Information Mashups: Integrating Information, Simply," *Proc. 32nd Int'l Conf. Very Large Databases* (VLDB 06), ACM Press, 2006, pp. 3–4.
6. M. Ogrinz, *Mashup Patterns: Designs and Examples for the Modern Enterprise*, Addison-Wesley, 2009.
7. S. Balasubramaniam et al., "Situated Software: Concepts, Motivation, Technology, and the Future," *IEEE Software*, vol. 25, no. 6, 2008, pp. 50–55.
8. T. Janner et al., "Patterns for Enterprise Mashups in B2B Collaborations to Foster Lightweight Composition and End User Development," *Proc. IEEE Int'l Conf. Web Services* (ICWS 09), IEEE CS Press, 2009, pp. 976–983.
9. C. Cappiello et al., "Enabling End User Development through Mashups: Requirements, Abstractions and Innovation Toolkits," *Proc. 3rd Int'l Symp. End-User Development* (IS-EUD 11), LNCS 6654, Springer, 2011, pp. 9–24.
10. T. Gamble and R. Gamble, "Monoliths to Mashups: Increasing Opportunistic Assets," *IEEE Software*, Nov./Dec. 2008, pp. 71–79.

11. M. Picozzi et al., "Quality-Based Recommendations for Mashup Composition," *ComposableWeb 2010*, LNCS 6385, Springer, 2010, pp. 360–371.
12. S. Roy Chowdhury et al., "Wisdom-Aware Computing: On the Interactive Recommendation of Composition Knowledge," *Proc. 6th Int'l Conf. Eng. Service-Oriented Applications* (WESOA 10), LNCS 6568, Springer 2010, pp. 144–155.
13. M. Burnett, C. Cook, and G. Rothermel, "End-User Software Engineering," *Comm. ACM*, vol. 47, no. 9, 2004, pp. 53–58.
14. J. Yu et al., "A Framework for Rapid Integration of Presentation Components," *Proc. 16th Int'l World Wide Web Conf.* (WWW 07), 2007, pp. 923–932; www2007.org/papers/paper468.pdf.
15. F. Daniel et al., "Hosted Universal Composition: Models, Languages and Infrastructure in mashArt," *Conceptual Modeling—ER 2009*, LNCS 5829, Springer, 2009, pp. 428–443.

Florian Daniel *is a postdoctoral researcher at the University of Trento, Italy. His main research interests are mashups and user interface composition approaches for the Web, Web engineering, and quality and privacy in business intelligence applications. Daniel received his PhD in information technology from Politecnico di Milano, Italy. Contact him at daniel@disi.unitn.it or www.floriandaniel.it.*

Maristella Matera *is an associate professor at Politecnico di Milano. Her current research interests span Web mashups, Web engineering models and design methods, quality in Web engineering, Web adaptivity, and context awareness. Matera received her PhD in information technology from Politecnico di Milano, Italy. Contact her at matera@elet.polimi.it or http://home.dei.polimi.it/matera.*

Michael Weiss *is an associate professor in the Department of Systems and Computer Engineering at Carleton University in Ottawa and teaches in the Technology Innovation Management program. His research interests include open source, ecosystems, mashups, patterns, and social network analysis. Weiss received his PhD in computer science from the University of Mannheim, Germany. Contact him at weiss@sce.carleton.ca or www.sce.carleton.ca/faculty/weiss.*