Mashups have emerged as an innovative software trend that re-interprets existing Web building blocks and leverages the composition of individual components in novel, value-adding ways. Additional appeal also derives from their potential to turn non-programmers into developers.

Daniel and Matera have written the first comprehensive reference work for mashups. They systematically cover the main concepts and techniques underlying mashup design and development, the synergies among the models involved at different levels of abstraction, and the way models materialize into composition paradigms and architectures of corresponding development tools.

The book deliberately takes a balanced approach, combining a scientific perspective on the topic with an in-depth view on relevant technologies. To this end, the first part of the book introduces the theoretical and technological foundations for designing and developing mashups, as well as for designing tools that can aid mashup development. The second part then focuses more specifically on various aspects of mashups. It discusses a set of core component technologies, core approaches, and architectural patterns, with a particular emphasis on tool-aided mashup development exploiting model-driven architectures. Development processes for mashups are also discussed, and special attention is paid to composition paradigms for the end-user development of mashups and quality issues.

Overall, the book is of interest to a wide range of readers. Students, lecturers, and researchers will find a comprehensive overview of core concepts and technological foundations for mashup implementation and composition. Even without low-level coding details, practitioners like software architects will find guidance on key implementation concepts, architectural patterns, and development tools and approaches. A related website provides additional teaching material which can be used either as part of a course or for self study.

This book is timely, provides a thorough scientific investigation and also has practical relevance in the general area of composition and mashups. It is of particular interest to researchers and professionals wishing to learn about relevant concepts and techniques in service mashups, composition, and end-user programming.

From the Preface by Boualem Benatallah, University of New South Wales, Sydney
Fig. 7.1 A simplified model of concurrent mashup components with components managing own users.
7.2 Multi-User Mashups

A notation format that allows developers to natively extend their applications with collaboration support.

7.2.2 Concurrent mashups

A way to solve the isolation problem of concurrent mashup components is to make the mashup itself concurrent: A concurrent mashup is a multi-user mashup that enables its users to operate a same instance of the mashup via a same view (e.g., a set of pages) in parallel, i.e., concurrently. That is, in a concurrent mashup it is the mashup that takes over the management of users and component synchronization.

As illustrated in Figure 7.2, this poses new requirements to the mashup itself: now it is the mashup that must be able to keep track of its user basis, to manage possible access rights (e.g., remember which user has access to which mashup or page thereof), and – more importantly – to synchronize at runtime the views of the different users participating in a same mashup instance.

Storing user profiles and managing user registrations and access rights is relatively simple. What is complicated is the synchronization at runtime of the Web pages, i.e., of the components running inside the Web browsers of the different users. The effect we would like to achieve with concurrent mashups is, e.g., that if one user performs a selection of an item in one component, also all other components connected to the same mashup instance see the selection in their own browser. This requires UI events to be propagated from the user performing the selection to all other users whose mashups are in “listening mode.”

Extending what we have seen for UI mashups in the last chapter (and in line with the infrastructure support proposed in [142]), concurrent mashups must therefore implement:

- A distributed eventing infrastructure, which allows a mashup instance in one Web browser to notify other instances running in other Web browsers about selected user interactions. It is generally neither necessary...
7.2 Multi-User Mashups

Business process is enacted by a single organization, but it may interact with business processes performed by other organizations.

The definition introduces three new concepts: activities, which represent individual, atomic pieces of work that can be assigned to and performed by participants in the business process (e.g., select a book or pay an order); coordination, which structures activities so that their joint execution achieves a pre-defined effect (e.g., before paying the order, it is necessary to select the book); and the business goal, which is the effect the business process wants to achieve (e.g., online book sales). Usually, business processes are expressed via business process models, e.g., using the Business Process Modeling Notation (BPMN), which can be parsed and enacted by a business process engine, causing the engine to interact with the people involved in the process and to automatically orchestrate their work.

Figure 7.3 provides an example of a business process model expressed in BPMN. The model illustrates a naive business trip approval and reimbursement process. The process is initiated by an employee, who creates a business trip request, which is inspected by the head of the employee's business unit and approved or rejected; alternatively, the head may also ask for additional details or changes to the request. If the request is rejected, the process ends (represented by the bold circle). If the request is accepted, the employee goes on the trip (note that this activity is not represented in the model as it is out of the control of the business process engine) and, once back, submits his expenses for reimbursement. Again, the head has to sign off declared costs and, once approved, the secretary takes care of reimbursing the employee. Given this process model, a process engine can coordinate the tasks of the involved actors and automatically trigger them when their intervention is needed.

In process mashups, we do not necessarily have the same kind of separation of process model and process engine, and it is the mashup itself that acts as both process model and process engine. A process mashup is therefore a

![Diagram of a business trip approval and reimbursement process involving three actors.](image)

**Fig. 7.3** A simple BPMN diagram representing a business trip approval and reimbursement process involving three actors.
Fig. 7.4 A simplified model of process mashups involving different actors performing activities.

Fig. 7.5 Simplified model of mobile mashups. (similar to the ones used in MashArt \[90\]), managing the rendering of UI components and the propagation of local, browser-internal UI events. This is only one example of how process mashups can be implemented. In \[95\], the authors review different tools and their suitability for the development of process mashups. They specifically highlight the three core dimensions introduced by process mashups, i.e., the multiple users, multiple pages, and business process logics, and argue that developing good process mashups is a complex task that only skilled developers are able to master.
7.3 Mobile Mashups

Next to being used by multiple users, mashups are also increasingly becoming more personal and pervasive, in that they may also run on smart phones or tablets, i.e., on mobile, carry-on devices. In this case, we speak about **mobile mashups**, i.e., mashups that run on a mobile device (e.g., a smart phone or tablet) and that, besides integrating remote Web APIs, may make use of device-specific components, such as device APIs, and/or device-specific implementation frameworks.

![Diagram](image)

**Fig. 7.5** Simplified model of mobile mashups.
UI views (e.g., the selection of a data item) with the invocation of operations in UI components (e.g., a search on Flickr, based on a search key selected in the core data view). Differently from Web-based mobile mashups, native mobile mashups are not sandboxed and therefore allow easy access to different remote data sources and services, which makes them less dependent on a Web server. Both types of mobile mashups are typically hybrid mashups, where the integration of data sources produces data views that are visualized through UI views that in turn are able to synchronize with UI components and the device-native services through events—which is proper of UI mashups. Mobile mashups are also usually distributed over client and server, while native mobile mashups may also run on the client device only.

Figure 7.6 highlights the main architectural components in the runtime environment of a native mobile mashup [59]. The client-side logic consists of a UI manager handling the dynamic creation of UI views. It makes use of the device technology to generate the different “screens”, both the ones displaying the integrated data view and the ones for wrapped UI components. For example, in the Android operating system, the UI manager can be achieved through different activities, each one managing the generation at runtime of the code handling a specific screen.

The management of the data sets (service querying and result set parsing and manipulation) is then operated by a data manager. Typical choices need to be made regarding the integration of data:

- If a hosted solution is adopted (that is, if there is also a server-side part of the mashup), data can be integrated at the server side, based on a mashup...
7.4 Telco Mashups

A network gateway is one that allows one to send SMS messages from the Web to mobile phones. This is the most common way to cross the boundaries of the Internet for Web applications in general, yet, an application or mashup may also implement internally a communications manager, which is able to directly talk to the target operator network without the need for a network gateway. While this is theoretically possible, in practice the costs for this are prohibitive, and this is therefore not an option for individual mashups (the idea expressed in the figure is that a dedicated telco mashup platform could make this kind of service available to its mashups, that is, to a multitude of mashups).

Assuming a suitable communications manager is available, it is possible to implement, for instance, a mashup that connects users with the three client devices shown in Figure 7.7: a conventional desktop Web client, a mobile smartphone accessing the Web via the Internet, and a conventional dial-in client (a phone) without visualization capabilities. In order to correlate the three devices, the communications manager would use an internal channel table, which allows it (i) to accept incoming phone calls and (ii) to correlate them with the respective mashup instances running in the platform. In practice, this may happen as with today's phone conferencing systems: the initiator of the mashup notifies all participants about the dial-in phone number (to reach the communications manager) and the access code to be used (to correlate users with mashup instances). Next, the communications manager uses the internal channel table to correlate the incoming phone call with the correct mashup instance.

Fig. 7.7 Reference telco mashup architecture as defined by Gebhardt et al. [121].
Fig. 7.8 A typical enterprise mashup architecture with logic layer, data layer including semantic data repositories, and security layer (based on [136]).