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. 1.1 The housingmaps.com mashup providing for the synchronized exploration of housing offers from craigslist.com and maps by Google Maps.
1.2 Mashup Types

A good instrument to understand today's mashup ecosystem is [http://www.programmableweb.com](http://www.programmableweb.com), an online registry of mashup components and ready mashups that can be inspected, also hosting a variety of news regarding mashups and a vivid mashup developer community.

In Figure 1.2, we illustrate a snapshot of the content of programmableweb.com at the time of writing this text. Actually, we copied the pictures this morning, while we are writing this text in the evening; in the meantime, 10 new APIs have been added to the API registry. We show the ranking of the most used APIs (the components) in Figure 1.2(a); Figure 1.2(b) illustrates the most used tags to describe developed mashups. As perhaps expected by most readers, Google Maps turns out to be the most used component by the community behind programmableweb.com, followed by Twitter, YouTube, Flicker, etc. As for the categories of mashups developed by the programmableweb.com community, in line with the popularity of the components, we find mapping mashups at the first place, followed by “deadpool” (by now dismissed mashups), search, social, and photo mashups, etc.

A closer look at the mashups listed in programmableweb.com however reveals that there are also many applications that would not qualify as mashups according to our definition above, e.g., because they make use of only one single component and do not set up any component inter-communication. This may be compliant with some interpretations \[207\], but since in this book we specifically aim to study the composite nature of mashups, we do not further investigate the applications that make use of only one component.

Also, the tags used by developers to describe the mashups they developed are certainly a good instrument to search for mashups in a given domain. However, we do not think the resulting classification of mashups into mapping, search, social and similar mashups helps clarify the mashup ecosystem from a research and engineering perspective, which is typically more inter-

**Fig. 1.2** A snapshot of the programmableweb.com mashup ecosystem (as of October 23, 2013).
Mashup classes like Web mashups, mobile mashups, telco mashups, data mashups and similar bear some more semantics from this perspective. Yet, there still seems to be an arbitrary proliferation of prefix-mashup combinations, without any evident connection among them and, more interestingly, these types of mashups are not mutually exclusive (for example, there may be mobile telco mashups or Web data mashups). We therefore analyzed the most important mashup types we found in literature and found that the prefixes used by these classifications can almost all be fit into one of three perspectives. We graphically illustrate the resulting ecosystem of mashups and mashup definitions as a cube in Figure 1.3. The perspectives are:

- **Composition**: This perspective emphasizes the internals of a mashup, i.e., its components and how these are composed into a new application. This perspective is the one driving our own definition of mashup in Definition 1.1. It stems from the traditional separation of concerns in software development, which separates an application into three layers, a data layer, an application logic layer and a presentation layer. This separation of concerns did not influence only how applications are structured internally, but it also fostered the growth of suitable API and component types at the three layers, so as to ease interoperability and integration of the layers. Looking at which layer of the application stack a mashup is composed, the composition perspective therefore groups mashups into data mashups, logic mashups, UI mashups, and hybrid mashups (any combination of the former three types).

- **Domain**: This perspective emphasizes the purpose of a mashup, i.e., the functionality it aims to provide. Partly, the tags used by developers to describe their mashups fall into this perspective, as they too describe delimited domains. The domains in this context may be essentially of two types: technological domains, such as for telco mashups or mobile mashups, and application domains, such as for social mashups or mapping mashups.

**Fig. 1.3** The *mashup cube* with three different perspectives on the mashup ecosystem.
The observation that mashups are typically still limited in their scope and simple is not ours only. In fact, mashup development – inside the larger context of software development – has been associated relatively early with the long tail of the software market. The long tail market model observes that traditional markets commonly target only 20% of the possible products/solutions in a given domain, which however guarantee huge sales numbers and large customer bases; the other 80% of the products are not considered bestsellers and, therefore, not even sold. Novel companies, most notably the online book store Amazon, instead make most of their money by addressing exactly these latter products and, therefore, selling products in the long tail of their market (e.g., books that are only very hard to find in regular, physical book stores).

Figure 1.4 applies the long tail model to the software market and illustrates the 80/20 rule (the so-called Pareto principle) in terms of applications and development effort. The mainstream software market focuses on the 20% of applications that guarantee high revenues, while the other 80% of applications, which would however bring added value to some users, are not even developed. In terms of invested development effort, the rule is typically inverted: 80% of the effort by the market is invested into the 20% of bestselling applications, and only 20% of the effort goes into the other 80% of applications.

As the figure also highlights, it is this 80% of applications that are not in the scope of the mainstream software market that represent the “market opportunity” for mashups. We intentionally use quotes, as we do not want to imply that mashups are mandatorily sold like any other product, although they might. As we will see, mashups may also serve very personal, situational needs that apply to an individual user only. However, if the development of mashups is adequately supported, e.g., via suitable mashup tools and easily accessible mashup paradigms, this is the final goal of this book – also these very limited needs can be satisfied.

**Fig. 1.4** The long tail of the software market and its opportunities for mashups.