Chapter 7
Evolution of Web Systems

Holger M. Kienle and Damiano Distante

Summary. The World Wide Web has led to a new kind of software, web systems, which are based on web technologies. Just like software in other domains, web systems have evolution challenges. This chapter discusses evolution of web systems on three dimensions: architecture, (conceptual) design, and technology. For each of these dimensions we introduce the state-of-the-art in the techniques and tools that are currently available. In order to place current evolution techniques into context, we also provide a survey of the different kinds of web systems as they have emerged, tracing the most important achievements of web systems evolution research from static web sites over dynamic web applications and web services to Ajax-based Rich Internet Applications.

Parts of this Chapter have been taken and adapted from other publications of the first author [469] [460] [467] and the second author [98] [97] [315] [316] [96].

T. Mens et al. (eds.), *Evolving Software Systems*,
DOI 10.1007/978-3-642-45398-4_7,

7.1 Introduction

The emergence of the World Wide Web (WWW), or the *web* for short, has led to a new kind of software that is based on web technologies: web sites, web applications, web services, and possibly others. In the following, if we do not make a distinction among these, we speak of them as *web systems*. We have chosen this term to convey that software that is based on the web platform can be of significant complexity, employing a wide range of technologies—starting from complex client-side code based on HTML5, Flash and JavaScript, to server-side code involving high-performance, cloud-based web servers and database back-ends. Web systems also have to deal with demanding non-functional requirements, which are cross-cutting the client and server side, such as scalability and security.

Research on the evolution of web systems is a relatively young research branch within the research on the evolution of software systems. Early research into web systems evolution started towards the end of the last millennium. The first International Workshop on Web Site Evolution (WSE), held in October 1999, evolved into the annual Web Systems Evolution workshop and symposia series sponsored by the IEEE. Since that time, this research branch has become more prominent, reflecting the web's increasing significance, and has broadened its scope, reflecting the web's increasing diversity.

Even though web systems have their own characteristics and idiosyncracies (cf. Section 7.1.2), almost all of the evolution techniques that have been proposed for software systems in general (e.g., refactoring [596] and clone detection [592, Chapter 2]) can be suitably adopted and applied to web systems (e.g., refactoring of PHP code [856] and detection of cloned web pages [243]). In addition, dedicated techniques have been developed to account for the specific characteristics of web systems (e.g., testing for browser-safeness, discussed in Section 7.4, and refactoring of web design models, discussed in Section 7.3.2). In this chapter we concentrate on techniques and tools that target web systems evolution that are meant to change one or more of the following aspects of a web system: its architecture, its (conceptual) design, and its technology.

7.1.1 Reengineering

One of the most involved methods to realize software evolution is *reengineering*, which conceptually can be defined as the composition of *reverse engineering* followed by *forward engineering*. The system to evolve is first reverse engineered to obtain higher-levels of meaning/abstractions from it, such as a model of its current design. Based on the information obtained in the reverse engineering step and the evolution objectives, a forward engineering step produces a new version of the system with a new architecture, design, and/or technology. The approaches discussed in Section 7.3 are mostly reengineering techniques.

Reengineering activities are performed with certain goals in mind, of which prominent ones are: (1) to "come to a new software system that is more evolvable [...] than the original one" [592, Chapter 1], or (2) to "improve the quality of the software" [781]. Regarding the first goal, making a system more evolvable often means adapting it so that it remains usable in reaction to changes in the real world (i.e., changing requirements or assumptions, cf. Chapter 1) in which the software operates (e.g., a new execution context, or interfacing it to another system). Regarding the second goal, quality improvements of a system can refer to internal quality, external quality, or quality in use (cf. ISO/IEC 9126 [12]).

7.1.2 Evolution Challenges and Drivers

Just like software in other domains, web systems have evolution challenges—perhaps even more so compared to other domains because standards, technologies, and platforms in the web domain are changing rapidly.[1] Web systems development is often equated with rapid development cycles coupled with ad-hoc development techniques, potentially resulting in systems of lower quality and reliability. An empirical study that tracked six web applications over five years for anomalies and failures found that "only 40% of the randomly selected web applications exhibit no anomalies/failures" and comes to the conclusion that "the common idea of researchers and practitioners is correct, i.e., process, tools and in general methodologies for web development and testing should be improved" [719].

Web systems have distinct characteristics from other domains, say desktop applications, that present unique evolution challenges. One of these challenges is their architecture: a web system is split between a client side and a server side, with possibly complex interactions between the two.[2] Another one is the use of the web browser as client platform. A web system needs to support several different web browsers consistently, whereas each of these browsers is evolving at a rapid pace. Also, the scope and complexity of standards that affect browsers is increasing with each iteration.

A related issue for modern web systems is the challenge to accommodate a wide range of devices. Previously, web development could assume a "classical" desktop browser as its target and design for this target in terms of technologies, user interface, user interactions, etc. With the emergence of smartphones and other mobile devices, web systems should be equally appealing across all of them, regardless of

[1] A graphical illustration of the evolution of browser versions and (web) technologies can be accessed at http://evolutionofweb.appspot.com/?hl=en.

[2] Conceptually, web systems adhere to the client-server model. This model is useful to understand the high-level architecture and the split in functionality between the web browser (client) and the back-end (server). However, it should be noted that the concrete architecture of a web system can differ in the sense that it may utilize multiple servers, such as for load balancing or for realizing a three-tier architecture that separates functionality into web browser, application server and database server. Also, a web system may be composed (or "mashed-up") of several services (accessible via web APIs) and each service may be hosted on a different server.

the devices' form factors. Approaches such as *responsive web design* address this challenge [562].

Lastly, modern web systems provide diverse content, often multimedia, along with sophisticated functionality for navigating and manipulating that content by the user. Access and manipulation of content are governed by complex business rules and processes. Due to the nature of web applications, both content manipulation and business rules are tightly interwoven.

To summarize, evolution challenges that are often pronounced in the web domain are:

- rapid churn of standards, technologies and platforms;
- rapid change of requirements and (domain) assumptions;
- ad-hoc development practices, which lack well-defined processes;
- complex interactions between client and server side that is difficult to comprehend, analyze and trace;
- use of multiple (web) technologies with complex interactions among them;
- support of multiple browsers and assurance of browser-safeness; and
- support for multiple devices with a wide spectrum of form and performance factors, including processing speed and connection bandwidth.

The evolution of web systems is caused by different drivers. While this chapter has a strong focus on tools and techniques in the context of changing web technologies (i.e., technological evolution), one should keep in mind that there are other drivers as well that are interacting with technological aspects and that also have a strong impact on the web's evolution. Examples of such drivers are consumers' satisfactions and demands, market competition among web-based *business models*[3] and e-commerce platforms, and laws and regulations (e.g., in the public administration domains). Depending on the web system's domain, the key drivers can differ, but regardless of the domains, technology serves as an enabling factor for evolution. In the following, we briefly reflect on important drivers and how they interact with technology.

Originally, the web's purpose was centered on the dissemination of (scientific) information and consequently the early web mostly had brochure-ware sites (cf. Section 7.2.1). Over the years, the web has seen an increasing commercialization driven by online shops with novel business models as well as traditional "bricks and mortar" businesses that started utilizing the web as a new sales channel ("bricks and clicks") [702]. As a result, the web presence of a company can represent an important (intangible) asset that may significantly affect its revenue and goodwill.

The concept of web applications along with improved technological capabilities (e.g., HTTPS, CSS, JavaScript, and plug-ins such as Flash) enabled organizations to establish and innovate on virtual stores and to offer increasingly sophisticated e-commerce capabilities. In this evolution, technology and business models are cross-fertilizing each other. User-generated content (UGC) is an example of a concept that was enabled by an interplay of both technology and business drivers. Blogs are

[3] While business models are typically associated with commercial gain, they can be defined as describing how an organization captures value, which can be economic, social and cultural.

an early example of UGC on the web, which was also commercially exploited by companies such as Open Diary (launched in 1998) and Blogger (launched in 1999); later examples of UGC are Wikipedia, YouTube and Facebook.

UGC enjoys high popularity with users, which has prompted many web systems to develop business models that entice users to provide diverse content, including personal information. UGC is also often highly interactive and real-time. By necessity, UGC is stored on the server, not the client. In effect, such web systems are now described as *hosted services* accessible through a cloud-based web application (cf. Section 7.2.5). These kinds of applications are often tightly coupled with service models on different levels: software (SaaS), platform (PaaS) and infrastructure (IaaS). Hosted services can utilize convenient payment functionality, ranging from more traditional credit-card services over online payments systems (e.g., PayPal and Google Wallet), to micro-payments (e.g., Flattr). As a result, desktop applications are increasingly replaced by, or alternatively offered as, hosted applications on a subscription bases (e.g., Microsoft's Office Web Apps and Adobe Creative Cloud).

The above developments, among others, have driven technological innovations in the areas of server architectures, browser features, caching, virtualization, (agile) software engineering methodologies, programming/scripting languages and their efficient compilation/interpretation, web-development platforms and frameworks (e.g., Ruby on Rails), and API design.

Both the web's reach and commercialization have contributed to the fact that legal issues are now an important concern [463] [466]. Legal issues are a driver in the sense that it restricts features and innovation in business models and technology. For example, copyright law has been at the center of many disputes around innovations [513]; examples on the web are deep and inline linking to other sites, framing of other sites, reverse engineering of client-side code, time-shifting (MP3.com) and space-shifting (Cablevision) of content [743], and UGC. Web systems that process personal data or UGC have to accommodate privacy, data protection and security concerns, which are partially governed by consumer protection and commercial laws.

7.1.3 Chapter's Organization

The remainder of the chapter is organized as follows. Section 7.2 presents techniques, tools, and challenges of web systems evolution research. The presentation is structured along a historical account of how the web has evolved in terms of the emergence of novel kinds of web systems: static web sites, dynamic web applications, web services, Ajax-based Rich Internet Applications, web systems leveraging cloud computing, and HTML5-based web systems (cf. Sections 7.2.1–7.2.6, respectively). For each kind of web system, where applicable, we highlight the most important research achievements in terms of state-of-the-art techniques and tools as they were proposed at the time.

In Section 7.3 we then focus on architecture, design and technology evolution of web systems. These three dimensions represent major challenges of web systems evolution research. Prominent challenges of architecture evolution are the migration of a web system towards SOA (cf. Section 7.3.1.1) or MDD (cf. Section 7.3.1.2); challenges of design evolution are the refactoring of a web system's design to meet new requirements (cf. Section 7.3.2.1) and to improve upon a certain quality, such as usability (cf. Section 7.3.2.2); a challenge of technology evolution is the migration towards a new platform and/or technology such as Ajax (cf. Section 7.3.3).

Section 7.4 provides a concise overview of the research topics of web systems evolution, including the topics covered in Sections 7.2 and 7.3. This section also describes evolution research topics that are unique for the web domain. Section 7.5 identifies research venues and journals as well as outstanding dissertations for further reading, and Section 7.6 concludes the chapter with parting thoughts.

7.2 Kinds of Web Systems and their Evolution

This section describes the different kinds of web systems that have been targeted by web systems evolution research: static web sites, web applications, web services, Ajax-based Rich Internet Applications, and cloud computing. These web systems—and the accompanying major research topics and challenges—are introduced in the following subsections as they have emerged over the history of the web. This structuring should allow readers that are not familiar with the overall research to better place and assess individual publications and research achievements. This section also highlights that each evolutionary step of the web itself had a corresponding impact on evolution research.

To better understand and classify approaches for web systems evolution, one can distinguish between different *views*—client, server/deployment or developer—that an approach supports [469]. These views address the user perspective of an approach or tool in the sense of what kinds of information are presented to web developers and maintainers.

Client view: The view of the web system that a user sees (typically using a web browser). For this view, information can be obtained by automatically crawling[4] the web system, which is accomplished without requiring direct access to a web system's sources: The web system has to be treated as a black box, only its output in terms of served web pages can be observed and analyzed.

Server/deployment view: The view of the web system that a web server sees (accessing the local file system). This view provides access to the web system's sources (such as, HTML pages, Common Gateway Interface (CGI) scripts, JavaServer Pages (JSP), PHP: Hypertext Preprocessor (PHP) scripts, and configuration files).

[4] Extracting facts from a web system based on the client view is called crawling or spidering.

Developer view: The view of the web system that a developer sees (using a web development tool such as Dreamweaver, or an IDE such as Eclipse, and a web server or an application server such as Apache or Apache Tomcat, respectively). This view is, by necessity, dependent on the tool's abstractions and features.

The three views introduced above are all of potential interest for web systems evolution. For example, the developer view shows the high-level web design such as information about templates; the server view is the one the web server uses and thus important for server maintenance and security; finally, the client view is the one that the end user sees and thus is important to assess external quality factors of the web system, such as navigability, learnability, accessibility, and usability. For effective web systems evolution an approach should ideally support all three views and track dependencies among them.

7.2.1 Static Web Sites

The first technological wave of the web consisted of static web sites that were primarily coded in HTML (so-called *brochure-ware* web sites [850]). A seminal paper raised awareness and popularized the notion that the web was predisposed to become "the next maintenance mountain" [141]. As a starting point for further evolution research, it was recognized that features of web sites could be conceptually mapped to software and, hence, that there was a need for web site evolution research in areas such as development process, version management, testing, and (structural) decay.

One key focus of research at the time was on metrics and (link) structure of web sites. Metrics for web sites typically analyze the properties of the HTML code. Actual metrics are often inspired by software and/or hypertext metrics. The evolution of a web site can then be tracked by analyzing historical snapshots and their associated metrics [141] [909]. The link structure of a web site is similar to the call structure of a program written in a procedural programming language. The nodes of the graph represent web pages and directed arcs between nodes represent a hypertext link between them. Different node types can be used to distinguish between HTML pages, image files, 'mailto:" URIs, etc.

The graph can be constructed by crawling the web site, starting from its home page.[5] One such tool adapted a customizable reverse engineering tool, Rigi [465], with functionalities for the web-domain. It allowed interactive exploration of the link structure of a crawled web site and to apply automated graph layout algorithms (cf. Figure 7.1) [569]. Based on such graph structures static properties can be verified, such as unreachable pages (i.e., pages that are available at the server side, but not accessible via navigation from the site's home page) and the shortest path to

[5] Typically there is the assumption that all pages are reachable from the home page. However, there are also analyses to detect unreachable pages (see below).

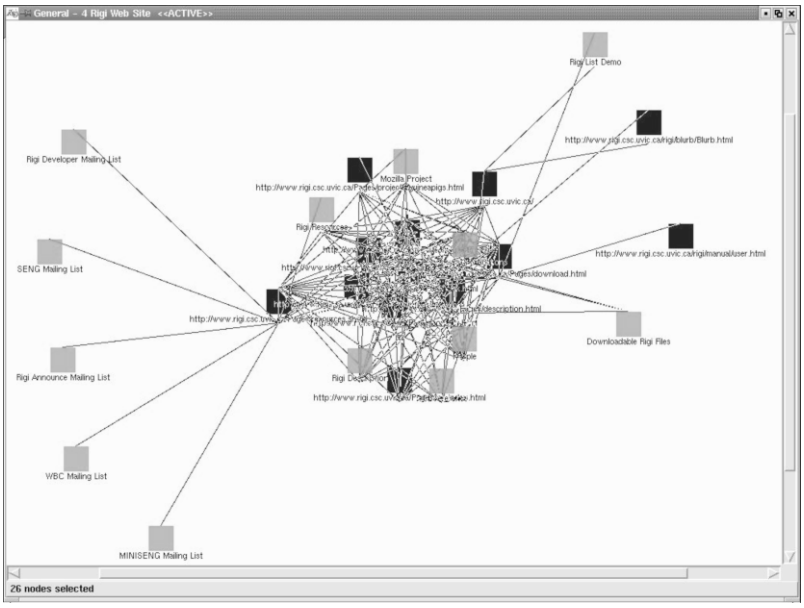

Fig. 7.1: Link structure of a web site consisting of 651 nodes rendered with the Rigi tool [569].

each page from the homepage [717]. The latter can be useful, for instance, for a rudimentary usability assessment.

7.2.2 Web Applications

Over the years, new web sites emerged (or existing web sites evolved) to support dynamic behavior both on the client-side (e.g., via JavaScript) and the server-side (e.g., via CGI and PHP).[6] This new breed of web sites were termed *web applications*. In order to accommodate the increasing sophistication of web applications over the years—which is also a reflection of the Web 2.0 (Chapter 6)—, the research

[6] Scripting languages have always played a prominent role in realizing web systems. On the server side, before dedicated scripting languages such as PHP and JSP became available, Perl was a popular approach for ad-hoc composition of web sites. (In 1999, Perl has been called "the duct tape of the Internet" [359].) Since around 2010 the ability of server-side JavaScript has gained momentum (e.g., the Node.js library). Its proponents want to close the conceptual gap between client and server technologies.

literature has also taken up the term Rich Internet Applications (RIAs) to distinguish these technically complex web applications from the more primitive ones.[7]

RIAs are web applications that are characterized by a user experience that is highly interactive and responsive so that they can rival the experience that desktop applications can offer. In this respect, the "rich" in RIA refers to the complexity of the underlying data that the user can manipulate as well as the user interface itself. The client side of RIAs is typically realized with a combination of JavaScript, CSS and HTML. While web sites use little JavaScript that is often self-contained and hand-coded, web applications often use a substantial amount of JavaScript that builds on top of existing libraries and frameworks (e.g., jQuery and Dojo). Compared to early web applications that can be characterized as thin client, RIAs are realizing more of the web system's functionality on the client side (i.e., fat client). Furthermore, links are often encoded with client-side scripting and their targets have no obvious semantic meaning [602]. As a consequence, such web applications cannot be simply crawled and understood based on a static link structure anymore.

Static web sites, which have HTML-encoded links, are straightforward to crawl (and because of this many tools could afford to implement a custom solution for this functionality). However, with the introduction of more and more dynamic web applications with scripted links these crawlers became very limited because the navigation model that they are producing reflects an increasingly smaller subset of a web system's whole navigation space. A web application is often based on events that trigger JavaScript code that manipulates part of the current page's Document Object Model (DOM), in effect causing a state change in the web application. At the extreme, a single-page, Ajax-based web application may not even offer a single static link, resulting in an empty navigation model for a traditional crawler. Another problem that makes it difficult or impossible to construct a complete model is the "hidden" web caused by interactive query forms that access a server-side database.

Since many analyses for web systems evolution are based on the client view an accurate crawler is highly desirable. Unfortunately, crawling techniques did consistently lag behind the latest web systems and handling the dynamic features was addressed only inadequately for many years. The Crawljax tool, introduced in 2008, offered a solution to this problem [602]. It automatically constructs a state-flow graph of the target web system where different states are based on comparing the states' DOM trees. State transitions are performed by a robot that simulates actions on "clickable" elements (i.e., DOM elements with attached listeners). However, the tool's authors caution that "there is no feasible way to automatically obtain a list of all clickable elements" [602]. State changes are determined by an edit distance between the source and target DOMs. The edit distance uses a similarity threshold that can be varied by the tool user, where one possible setting corresponds to matching for identical trees. Besides the edit distance's threshold other settings can be used to control the crawling behavior such as maximum number of states and ignoring of certain links based on regular expressions. ReAJAX is another example of a sophisticated crawler based on a similar approach than Crawljax (cf. Section 7.2.4).

[7] However, it should be noted that RIA is not clearly defined and different authors attach different meanings to it.

In response to the advent of web applications, new approaches were developed to capture the increasingly dynamic behavior of web sites and their increasing heterogeneity in terms of the employed standards and technologies. This research met a need because development tools lacked in functionality for web site evolution: In 2001, a study of two popular web development tools at the time (FrontPage and Dreamweaver) showed that they had rather limited support for understanding and reverse engineering of web sites and that support was mostly restricted to static features [850]. Maintenance activities that were supported by these tools at the time are, for example, validation of HTML and XML documents, reports of usage-violations of ALT and META tags, link checking, metrics that summarize characteristics of web pages, and page download-time estimates.

In order to provide suitable information to reverse engineers who have to understand ASP-based sites, Hassan and Holt extract information from HTML, VBScript, COM source code, and COM binaries [374]. During the extraction process, each file in the local directory tree that contains the web site is traversed, and the corresponding file's extractor (depending on the file type) is invoked. All extractors' output is first consolidated into a single model and then visualized as a graph structure. This graph structure provides an architectural view of the web system that can be used as a starting point for evolution (cf. Figure 7.2). There are also approaches that combine static and dynamic analysis. For example, one proposed method leverages an extension of UML to show the architecture of the web application as class diagrams and its dynamic behavior with sequence and collaboration diagrams [242].

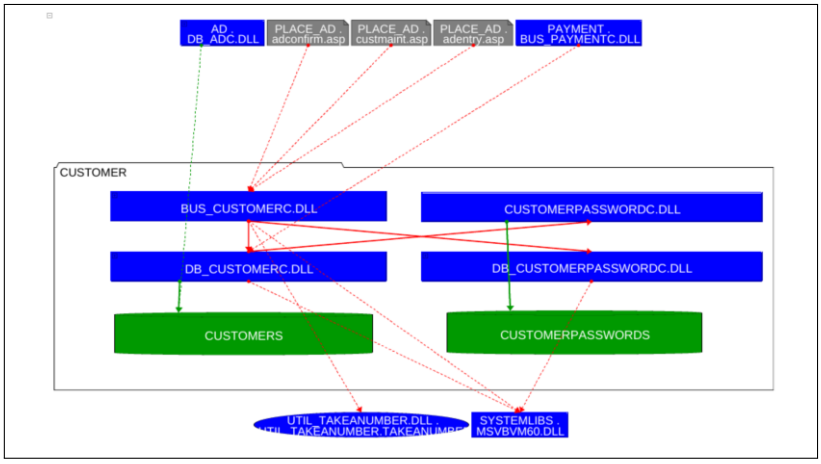

Fig. 7.2: Architectural view of a web system's components: Blue boxes are DLL files, gray boxes are ASP files, blue ovals are COM objects, green tubes are databases [372].

To improve evolvability, restructuring of server-side code has been proposed. For instance, Xu and Dean automatically transform legacy JSP to take advantage of an added JSP feature—the so-called custom tag libraries—to improve future maintainability by more clearly separating presentation from business logic [933]. Research has also tackled the migration away from static, HTML-only sites towards dynamic ones. For example, Estiévenart et al. have a tool-supported method to populate a database with content extracted from HTML pages [281]. This database can then be used to build and serve pages dynamically. Ricca and Tonella have realized a conceptually similar approach [718].

Web applications, and especially RIAs, are often developed with sophisticated frameworks and tools that provide higher-level concepts, which then need to be realized with a generator that produces code that can be executed by the web server. An unusual example is the Google Web Toolkit: it allows coding in Java with dedicated APIs and widgets and this code is then compiled to optimized JavaScript. Dedicated functionality for web site evolution can be added to tools if their architecture is plug-in based. Such an approach has the advantage that the user can work within the developer view. The REGoLive tool adds reverse engineering capabilities to the Adobe GoLive web authoring tool [354]. For example, REGoLive provides a graph-based structure of a web site, showing artifacts—including web pages, CSS files and JSPs as well as tool-specific entities such as templates and so-called smart objects—and their dependencies.

RIAs typically make extensive use of JavaScript on the client side and the resulting code base can be significant.[8] For instance, Google's GMail has more than 400,000 lines of hand-written JavaScript [430]. Thus, JavaScript is an important consideration for web systems evolution. It is a dynamic, weakly-typed language that offers an interesting combination of language features, mixing imperative, object-based and functional programming with concepts such as mutable objects, prototype-based delegation, closures, and (anonymous) functions objects. In JavaScript pretty much everything can be manipulated at run-time (introspection and intercession), there is no information hiding and there is "eval" functionality that allows to execute an arbitrary string as JavaScript code. As a result, JavaScript features make it difficult for static analyses to produce meaningful results, and dynamic analyses are a more promising approach for analyzing the behavior of JavaScript.

JavaScript has no explicit concept of classes and as a result various idioms are used to mimic this concept. If multiple idioms are used in a single code base maintainability becomes more difficult. Gama et al. studied 70 systems and found five different idioms in practice [310]. Based on these idioms they developed an automated code transformation that normalizes a code base to a common idiom. The authors observe that "there seems to be remarkably little other work on JavaScript style improvement" but one would expect that research interest in this area will pick up in the future. Another transformation example is an approach and tool for extracting a subset of client-side JavaScript code that encapsulates a certain behavior [560]. With this dynamic analysis a certain (usage) scenario such as using a UI widget is

[8] RIAs can be also realized without JavaScript if they are based on proprietary technology (e.g., Adobe Flex or Microsoft Silverlight).

first interactively executed and tracked. Based on this run a dependency graph is constructed that contains HTML, CSS, and JavaScript nodes along with their structural, data and control flow dependencies. This enables to extract a self-contained subset of the code that is able to reproduce the usage scenario. This approach can be also used for dead code removal (e.g., to speed up page load time) if the scenario is able to capture all expected behaviors of the web application.

7.2.3 Web Services

Around the time that web applications established themselves, the concept of *web services* started to become more prominent. The move towards web services was mostly driven from a business perspective that envisioned cost savings and increased flexibility [849] [20]. Web services are closely related to Service Oriented Architecture (SOA) in the sense that web services are an enabling technology for realizing a system that adheres to the service-oriented architectural style [830] [592, Chapter 7]. From this perspective, migration towards web services can be seen as architecture evolution and is discussed in more detail in Section 7.3.1.

Evolution of a web service entails significant challenges: distributed components with multiple owners, distributed execution where multiple workflows are executed concurrently, and machine-generated description files (e.g., WSDL, XSD and BPEL) [918] and messages (e.g., SOAP messages). Understanding a Web Service Description Language (WSDL) specification can be complex because it contains a number of concepts (i.e., types, messages, port types, bindings and services) that can be highly interrelated via referencing. Since WSDL provides a high-level description of the important aspects of a web service, it plays an important role when a service—or a system that uses the service—is evolved.

Examples of analyses based on WSDL files are clone detection of services and enabling of automated service discovery [568] [348]. To obtain meaningful results, the WSDL files are first suitably restructured (so-called contextualization) by inlining referenced information. This allows to apply established algorithms such as topic models [348] and near-miss clone detection [568] to find similar operations. These similarities can be used as input for maintenance activities, but also for web service discovery (i.e., finding an alternative service).

Fokaefs et al. [298] present an empirical study on the evolution of web services by applying a differencing technique to WSDLs. The WSDL files are first suitably stripped to form another valid XML representation that reflects the client's perspective of the service. Pairwise differencing of the XMLs are then performed by the VTracker tool, which is based on a tree-edit distance algorithm that calculates the minimum edit distance between two XML trees given a context-sensitive cost function for different edit operations. The evolution of the successive WSDLs can be studied with the percentage distribution of change, delete and insert operations. Among other cases, the authors have tracked the evolution of 18 WSDL versions of Amazon's Elastic Cloud. The analysis showed that the service underwent rapid ex-

pansion (additions are dominating) while the existing interface was kept relatively stable (deletions and radical changes are avoided). The authors could also find correlations between WSDL changes and business announcements.

SOAMiner is a static analysis tool for searching and indexing a collection of service description files, including WSDL [918]. It is based on Apache Solr, which provides a *faceted* search infrastructure. Examples of search facets are file types and XML tag names. A user study found that "faceted search capability proved to be very efficient in some cases" and that this approach can be more effective compared to text-based searching because the latter requires more domain knowledge.

The authors of SOAMiner also present a dynamic analysis tool. The feature sequence viewer (FSV) visualizes sequence diagrams of messages passed between web services [918]. The sequences are extracted from trace data and a simple heuristic is used to determine messages that are relevant for a certain scenario. De Pauw et al. have also realized an approach based on trace data that relies on message content and timestamps [225]. Their analysis is able to find identifying keys such as order numbers, which are used by the web services to correlate messages and to realize possibly asynchronous workflows. This enables to find "semantic correlation" between different kinds of messages (e.g., an order number that is first used in a message to order an item and then, in a subsequent message, to confirm its shipment). In effect, sequence diagrams, which only show the control flow (such as the FSV tool, see above), are augmented with dependencies that show correlations of message content.

As described above, a standard approach for a web service is to utilize WSDL and SOAP, which specify the service's contract and API in terms of permissible invocations and required data types. This flavor of web service is also referred to as *WS-* web service* [683]. Alternatively, the API of a web service can be also realized with Representational State Transfer (REST) [293]. The basic approach is to design the data model as a network of data items, or so-called resources, and to define URIs to access, navigate and manipulate these resources with HTTP request methods (i.e., GET, PUT, POST and DELETE). Typical data encodings for such RESTful web services are JSON or XML, but plain text is also conceivable. Alarcón and Wilde propose the Resource Linking Language (ReLL) to model RESTful web services [15]. They have developed a crawler, called RESTler, that can extract a resource network from a RESTful web service based on the service's ReLL model. The obtained resources in combination with the ReLL models can then be used for composing new, mashed-up services.

According to Pautasso and Wilde, "it is not possible to simply say that one variety is better than the other, but since RESTful web services gained momentum, it has become clear that they do provide certain advantages in terms of simplicity, loose coupling, interoperability, scalability and serendipitous reuse that are not provided to the same degree by WS-*" [683]. Thus, support for migrations from WS-* to RESTful web services, and vice versa, is desirable. Strauch and Schreier present RESTify, a semi-automated approach to transform a WSDL-based web service to a RESTful design [798]. The approach supports, for instance, the identification of resources from a WSDL description, and the definition of resource access by defin-

ing a mapping from a WSDL operation to URI request sequences. Research has strongly focused on REST as migration and reengineering target (e.g., [534] [816] [798]), neglecting the opposite direction.

7.2.4 Ajax-based Web Systems

Another major evolution of the web is marked by the possibility of a web application to initiate an asynchronous connection to obtain data and presentation information (i.e., *Ajax programming* [314]). RIAs that employ Ajax frameworks and technologies result in highly sophisticated web applications whose functionality rivals and surpasses native desktop applications. Compared to traditional web sites and web applications in which contents and functionalities are distributed among several pages that the user navigates, an Ajax-based RIA might consist of one single web page whose content and user functionalities change dynamically in consequence of the user actions, without any page reloads. More frequently, however, Ajax-based RIAs are a combination of traditional web applications and enriched user-interface features, which are made possible by Ajax technology. JavaScript is used as the main coding language to implement features, both on the client and, increasingly, on the server side.

Of course, for this new stage, yet again novel approaches are needed for effective web systems evolution. ReAJAX is a reverse engineering tool for single-page Ajax RIAs [561]. Similar to Crawljax (cf. Section 7.2.2) the web system is dynamically executed to construct a state model based on the pages' DOM representations. However, with ReAJAX the model extraction can be customized by abstracting the DOM representation with a function that maps DOM elements to higher-level values that strives to capture the state of the application based on its GUI elements. For instance, for a certain application it may make sense to abstract an HTML table with three distinct values that represent an empty table, a table with one row, or a table with more than one row. State transitions represent a change in the abstracted DOM rather than the actual DOM and hence can represent meaningful states in the application's logic. The state model is constructed by manual or automatic execution of a set of execution scenarios. The quality of the resulting state model varies depending on the coverage of the scenarios and the suitability of the DOM abstractions. CReRIA is another example of a reverse engineering tool that is based on dynamic analysis and creates a state model [26] [27].

Research has also tackled the migration from legacy web systems towards RIAs and/or Ajax-enabling them ("ajaxification"). In 2007 Mesbah and van Deursen observed that "today we have a new challenge of migrating classic web applications to single-page web applications" [601]. This kind of migration can be seen as technological evolution as discussed in Section 7.3.3.

7.2.5 Web Systems Leveraging Cloud Computing

Arguably, cloud computing marks another major step in the web's evolution. However, it should be stressed that cloud computing is an independent principle that applies to software systems in general. The "running [of] applications within a network server or downloading the software from the network each time the software is used" is one of its prominent characteristics [817]. Cloud computing can be defined as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [585]. Importantly, the access to these resources needs neither be realized with a browser-based user interface nor with web-based technologies.

Tupamäki and Mikkonen point out that there can be mismatches between web principles and cloud computing (e.g., single-server design) [861]. Still, existing web systems are among the most promising candidates to evolve towards the cloud—or often have already properties that are now associated with mainstream cloud computing. Ajax-based RIAs typically provide an application/service along with storing the corresponding user-data on the server-side (e.g., webmail such as Yahoo! Mail and GMail, and storage such as Dropbox and Microsoft's SkyDrive)—properties that are labeled as Software-as-a-Service (SaaS) by cloud computing.[9] The underlying service level, Platform-as-a-Service (PaaS), can be seen as web-based middleware to realize SaaS. For example, Windows Azure and Google App Engine provide a platform of different services, which can be realized to build SaaS-based web systems based on an integrated set of diverse APIs. Thus, similar to SOA and web services, different (heterogeneous) PaaS and Infrastructure-as-a-Service (IaaS) based services can be orchestrated to realize a higher-level service.

Every advent of a new technology poses the challenge of how to best utilize it in the context of an existing (legacy) system. Established techniques for transitioning to a new technology include wrapping, migration, and reengineering (cf. Section 7.3). In this context the potential mismatch between the principles and architectures of web application frameworks and cloud computing platforms present a significant challenge [861]. Kienle et al. observed in 2010 that "surprisingly, there is still comparably little discussion of [web] system evolution towards the cloud such as when and how systems should be migrated to the cloud and techniques to accomplish such a migration in an effective manner" [462]. This observation still holds true as of this writing.

[9] HTML5 enables clients to utilize local storage; thus, the location of (personal) data may partially move from the server to the client. However, leveraging this capability may not be in the interest of the service provider.

7.2.6 HTML5-based Web Systems

The HTML5 standard is still at a draft stage and far from being finalized, but it is already well on its way to become a ubiquitous platform for building all kinds of (web) systems. Thus, it can be expected that HTML5 will play a significant role in web systems evolution.

It appears at this point that there are two major competing approaches: HTML5-based web systems, which are vendor-agnostic and represent the *open web*, and native web systems, which are vendor-specific [608]. Examples of the latter are *apps* running on Apple's iPhone and iPad or Google's Android devices. While these apps can utilize web-based protocols and principles (e.g., HTML and REST) and while their look-and-feel can be similar to browser-based systems, they are built with vendor-specific platforms and native graphics libraries, and are typically distributed in binary form. It remains to be seen if both approaches will coexist or if one will extinguish the other. In the following we focus on open web systems only, because in many respects native apps are close to traditional desktop software.

HTML5 significantly expands the scope of previous HTML standards with the aim to enable web systems to come even closer to native desktop applications (without having to rely on browser plug-ins) [608]. Specifically, web systems can utilize local storage and built-in audio/video support, temporarily function without network connection, draw on a 2D canvas for procedural, interactive graphics, and render hardware-accelerated (3D) graphics. HTML5's capabilities may cause development to the move away from native, vendor-specific apps, meaning in effect that "browser technology will dominate over operating systems" [50]. This would further strengthen the trend towards truly web-based software.

In the past, web systems evolution research did address the migration from web applications to early mobile devices because this was rightly perceived as "the next big thing" [410]. If the trend towards HTML5-enabled mobile applications holds true, then research should tackle the migration from mobile applications written for native platforms to HTML5 technologies.

7.3 Dimensions of Evolution

As the previous section illustrates, research in web systems evolution has come up with a rich set of approaches and techniques, typically with accompanying tool support. In the following, we structure the discussion along three dimensions of evolution: architecture (cf. Section 7.3.1), (conceptual) design (cf. Section 7.3.2) and technology (cf. Section 7.3.3). Each of these have in common that an existing web system is migrated or reengineered, and, in any case, updated towards a new system. Depending on the dimensions, the new system differs in its architecture, design, or technology compared to the old one. However, it should be stressed that these dimensions are not orthogonal to each other, meaning that evolution in one dimension can imply or require evolution in the other ones as well.

7.3.1 Architecture Evolution

Evolving the architecture of a system means a fundamental change on how functionality is distributed into components and how these components communicate among each other. For instance, a monolithic system may be "split" into (loosely coupled) components for user interface, application/business logic, and data persistence, by using a client-server three-tier architecture with the client side running the user interface component [157]. Also, such a three-tiered client-server system may be further evolved to move part of the business logic onto the client. Such an architecture evolutions often also impacts the system's technology. In the previous example, the client-server architecture may be realized with web-based middleware technology (such as Java Business Integration or IBM Integration Bus), the business logic to be moved onto the client side may be implemented by using a client-side scripting language (such as JavaScript), and data persistence may be realized using some data persistence framework (such as Hibernate).

7.3.1.1 Towards Service Oriented Architecture

As mentioned before (cf. Section 7.2.3) web services are a popular approach—but not the only one—to realize a system that adheres to SOA. Since our focus is on web systems evolution, we restrict our discussion mostly to web services. However, there are generic approaches for migrating systems to SOA (e.g., SMART [518] and Heckel et al. [592, Chapter 7]) that can be instantiated to suit the context of web systems migration: the existing system is then any kind of web system and the SOA-enabled target system is based on web services. Generally, the purpose of migrating towards web services is the identification of existing (higher-level) functionality, isolating and extracting them from the existing system and making them accessible via web service interfaces.

One design principle for SOA—and, by extension, for web services as well—is composability where each (basic) service provides a well-defined and self-contained (business) need. An application is then realized by service composition. Web services are realized with WSDL (to specify their interfaces) and Simple Object Access Protocol (SOAP) (for messaging). A more lightweight approach for web services based on so-called web APIs eschews WSDL/SOAP in favor of HTTP-encoded request messages and JSON for data exchange. The latter approach is often used in combination with REST (cf. Section 7.2.3) and Ajax (cf. Section 7.2.4).

In 2008, Hainaut et al. observed that migration to SOA "appears as one of the next challenges" [592, Chapter 6, page 109]. Indeed, migrating existing code towards web services is a common evolution scenario, and Almonaies et al. present a survey of approaches [21]. Arguably, a web site may already offer "services" through HTTP response/request pairs that are initiated through user interactions. Jiang and Stroulia analyze these interactions with pattern mining to semi-automatically identify service candidates [441]. Almonaies et al. present a semi-automatic approach to migrate a monolithic PHP-based web application towards web services [20]. Poten-

tial business services are first identified and separated, then PHP code is restructured to expose functionality as services (based upon the Service Component Architecture (SCA) specification, which is available as a PHP implementation) and to allow communication between services (based upon SCA's Service Data Objects).

A migration does not necessarily start with "web-enabled" code, it may as well be an EJB application that is rearchitected towards a system composed of a set of web services [531]. Another scenario is the wrapping of COBOL mainframe programs so that their functionality is exposed as web services [782]. Compared to a full-fledged reengineering approach, "the complexity of the wrapping techniques is low, since there is no deep analysis of the legacy system" [21].

7.3.1.2 Towards Model-Driven Engineering

The emergence of the web as a prominent platform to develop and deploy systems and services—i.e., in other words cloud computing's SaaS, PaaS and IaaS (cf. Section 7.2.5)—and the increasing complexity of these systems has led to the need of tools and methodologies that streamline their design, development, and maintenance. Model-Driven Engineering (MDE) advocates the systematic use of models and model transformations throughout the life cycle of a software system, from requirement specification, to design, implementation (usually envisioned as automatic code generation), and maintenance [756]. By promoting automation of tasks and reuse of artifacts, MDE aims to increase productivity and to reduce development time and costs. One of the better known MDE initiatives is the Model-Driven Architecture (MDA) software development framework defined by the OMG [650]. MDA proposes a particular approach for realizing MDE by leveraging OMG standards (e.g., UML and MOF).

The engineering of web systems is a specific application domain in which the principles of MDE have been successfully applied, originating the rich and lively research area of Model-Driven Web Engineering (MDWE). By decoupling the functional description of applications (models at various levels of abstraction) from their implementation (code for a given platform) and by enabling the modification of the first and re-generation of the last out of the first, MDWE methods ease the evolution and adaptation of web systems to continuous changing requirements and emerging technologies. In this process, model evolution plays an important role (cf. Chapter 2)

The list of web engineering methods which natively adopt a model-driven approach to the development of web applications or which have been extended towards MDE includes: UWA [252], WebML [750] and its extension towards RIAs [304], UWE [480] and OOHDM [752], and OOWS [299]. Valderas and Pelechano present a comparative study of these and other MDWE methods which analyzes the techniques they propose for specifying functional, data and navigational requirements as well as the mechanisms provided for automatically translating these requirements into conceptual models [866]. According to their survey, only the WebML model explicitly addresses evolution concerns in a dedicated "Maintenance and Evolution" phase.

The reverse engineering of an existing web application with one of the above listed methods opens the door to reaping the benefits of MDE. An approach that enables recovering user–centered conceptual models from an existing web application according to the UWA methodology [865] is RE-UWA [98]. The approach is able to recover the content, navigation, and presentation models of the application, as observed from a user's perspective. These models can provide effective support for maintainer when deciding on some change/improvement to be applied to the application, with respect to its external, user-perceived, quality. The recovered models, eventually evolved according to new requirements, can be used then as input for a UWA model-driven forward engineering process, which is supported by dedicated tools [97]. Similarly, an approach that abstracts WebML hypertext models [137] from existing web applications is proposed by Rodriguez-Echeverria [724]. The approach can be used as the initial step towards modernizing the existing application into a RIA-based one using the WebML model-driven web engineering method [304].

7.3.2 Design Evolution

Often the intent of evolution is improving some external and user-perceivable quality characteristics of a system, rather than properties related to its internal realization. When this intent applies, the evolution process is usually first accomplished (conceptually) at the design level with the help of a (web) design model (cf. Section 7.3.1.2), to move then to the implementation level where it may be realized by selecting suitable technologies or algorithms.

In this section we present two approaches for web application design evolution: a redesign approach and a refactoring approach. The intent of the former is to modify the behavior of the web system while the latter preserves it.

7.3.2.1 Meeting New Requirements

When evolution is driven by the need to modify the behavior of the application (e.g., to implement new business rules and meet new or evolved requirements [592, Chapter 1]) or by the opportunity to improve aspects influencing the external quality characteristics of the application (e.g., its usability), evolution should be carried out at the design level. To this aim, it is desirable to rely on approaches that enable (1) to recover the current design model of a web application, and (2) to modify the recovered design model in order to effect the evolution's goals. RE-UWA is such an approach for the domain of web systems design evolution (already discussed in Section 7.3.1 in the context of MDE) [96].

The RE-UWA design evolution approach is based on a three-step process:

Reverse Engineering: A semi-automatic reverse engineering phase is used to analyze the HTML pages of the system's front-end with the goal to abstract its "as-

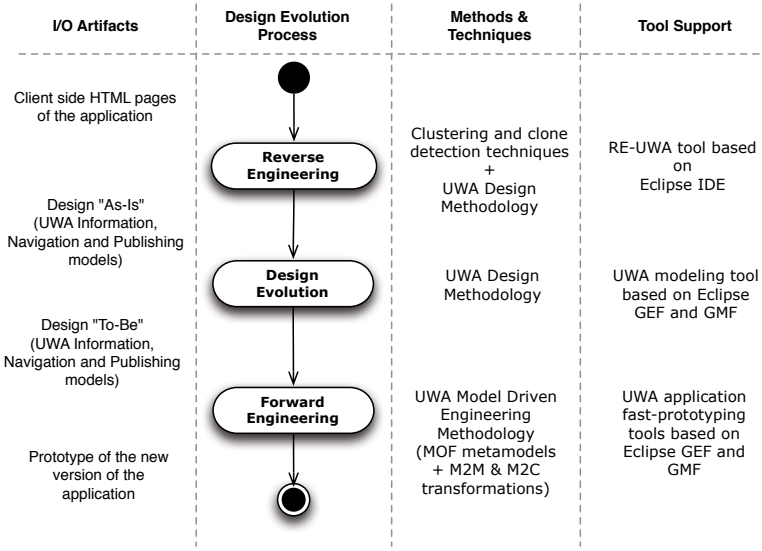

Fig. 7.3: The RE-UWA web application design evolution approach.

is" design. This phase applies clustering and clone detection techniques on the HTML pages of the application and is supported by the Eclipse IDE environment.

Design Evolution: This phase leverages the recovered models from the previous phase. The (new) requirements—which identify shortcomings and opportunities for improvements in the current design—are then used to construct the desired "to-be" design. This phase is supported by a set of modeling tools, which are build on top of the Eclipse Graphical Editing Framework (GEF) and the Eclipse Graphical Modeling Framework (GMF).

Forward Engineering: In this phase, the "to-be" design model is used to produce the "to-be" version of the web system. The UWA fast prototyping tools [97] can be also used to quickly implement a prototype of the new application and to verify/validate the new design.

Thus, the whole approach leverages the UWA design methodology to guide both the reverse and forward engineering design processes, and the UWA design models as the formalism to represent the "as-is" and "to-be" designs of the web system. Figure 7.3 summarizes the whole approach with the involved activities and supporting tools and techniques.

7.3.2.2 Improving Usability

The design of a web application can also be evolved to improve *quality in use* characteristics such as *usability* while preserving its behavior and business rules. To this aim, refactoring techniques can be applied to the design models of the application [316].

Refactoring is a technique that applies step-by-step transformations to a software system to improve its quality while preserving its behavior [596]. Originally introduced by Opdyke and Johnson in the early 90's [664] and mainly focused on restructuring a class hierarchy of some object-oriented design, refactoring became popular a few years later with Fowler's book [301], which broadened the perspective to a wider spectrum of code-centric refactorings and motivated its application for improving internal quality characteristics of software such as maintainability and evolvability. Since then, refactoring has further broadened both in scope and intent, as refactoring techniques have been applied to UML models [804], databases [29], and HTML documents [368]. In all cases the basic philosophy of refactoring has been kept (i.e., each refactoring is a small behavior-preserving transformation), but, as the scope, also the intent of refactoring has expanded to target external and quality in use characteristics of software, such as learnability and effectiveness.

Garrido et al. present catalogs of refactorings for the navigation and presentation models of a web application aimed at improving its usability [315] [316]. Each of the refactorings included in the catalog is characterized by a scope (i.e., the software artifact to which it applies), an intent (i.e., one or more usability factors it aims to improve), and a bad smell (i.e., the symptoms that may suggest applying the refactoring). The usability improvement approach that results from the application of the refactorings in the catalog is agnostic with respect to the method and the technologies adopted to develop the application, as all refactorings are described by showing how they affect the corresponding web page. Table 7.1 summarizes a subset of the refactorings included in the aforementioned catalogs.

7.3.3 Technology Evolution

Technology evolution of a system can be triggered by the retirement of a technology because the vendor supports it no longer, or by the realization that the employed technology no longer matches the system's requirements. In the former case, the vendor may provide a migration tool. For instance, a helper tool supported the migration from IBM's Net.Data legacy technology to JSP [499]. The latter case can mean, for instance, that internal qualities (e.g., maintainability), external qualities (e.g., performance), or both are increasingly difficult or impossible to meet because new requirements and old technology are at a mismatch.

For web systems, each new wave of the web (as described in Section 7.2) has triggered a technological evolution. However, web systems, especially early ones

Table 7.1: A subset of refactorings for usability improvement in web applications proposed by Garrido et al. [315] [316].

Refactoring	Intent	Scope
Convert images to text In web pages, replace any images that contain text with the text they contain, along with the markup and CSS rules that mimic the styling.	Accessibility	Code
Add link Shorten the navigation path between two nodes.	Navigability	Navigation model
Turn on autocomplete Save users from wasting time in retyping repetitive content. This is especially helpful to physically impaired users.	Effectiveness, accessibility	Code
Replace unsafe GET with POST Avoid unsafe operations, such as confirming a subscription or placing an order without explicit user request and consent, by performing them only via POST.	Credibility	Code
Allow category changes Add widgets that let users navigate to an item's related subcategories in a separate hierarchy of a hierarchical content organization.	Customization	Presentation model
Provide breadcrumbs Help users keep track of their navigation path up to the current page.	Learnability	Presentation model

with limited functionality, were not necessarily migrated in the strict sense but rather rebuilt from the ground up with new technology.

Another technological evolution is driven by mobile devices. They do not only bring new form factors and input devices, but also new technical challenges to keep web systems responsive. Web systems are typically developed with the expectation of a PC and a wired base connection, but smartphones and tablets can be less powerful than PCs and latency is more pronounced for wireless connections. Also, the performance when executing JavaScript on mobile devices is much reduced compared to PCs [944]. As a result, to make web systems responsive and fast for mobile devices, different techniques and algorithms are needed. For example, basic rules for achieving this are reducing of HTTP requests by concatenating files, avoiding redirects, limiting the number of connections, and replacing images with CSS3-based renderings. To accommodate mobile devices, the web system's APIs for the client can be changed and extended to handle device profiles and to allow the client to control limits on data [572].

Towards Ajax

From a technical point of view the introduction of asynchronous connections in the browser is a minor functional addition whose full potential was not recognized until

Ajax programming was proposed (cf. Section 7.2.4). However, as Ajax-based RIAs have demonstrated, Ajax can have a huge impact on the user experience. Consequently, if a web system is migrated towards Ajax it is often done with the goal to improve the system's usability (i.e., a design evolution, cf. Section 7.3.2.2) by taking advantage of Ajax's unique capabilities. Similarly, the prior technical shift towards more dynamic web systems and RIAs was intertwined with the evolution for usability improvements.

Chu and Dean have developed a transformation for "ajaxification" of JSP-based web applications [185]. They are handling the use of a user interface (UI) component to navigate a list of items where only a subset of list items is displayed for each page. A typical example would be the list of search results of a search engine with controls for "previous"/"next" as well as direct links to all result pages. The basic idea goes as follows. The user has to identify and annotate the sections of the JSP code that is responsible for rendering the list (e.g., a loop with a database query and markup code for a table row). The marked code is then sliced to produce an executable subset of the marked JSP code. The slicing criteria can be controlled with suitable annotations. As a result, the sliced code provides, when called, the rendering data that is sent in response to an Ajax request. Also based on manual markup, the JSP source is transformed to make an Ajax call that pulls in the list items. To make the list item's rendering data match with the needed HTML/DOM structure of the hosting page, the transformation has to make suitable adaptations. The approach has been tried out on four applications and for all of them the visual rendering is preserved.

Mesbah and van Deursen propose Retjax, an approach and tool for migrating multi-page RIAs to Ajax-based, single-page web systems [601]. The goal is to find UI components that can be transformed to leverage Ajax. Multi-page RIAs have to build a whole new page whenever information in a UI component changes. A single-page approach, in contrast, would refresh only the UI component, which is embedded within the page. To identify the UI components, first a model of the existing web application is constructed. Retjax uses HTML Parser for this purpose, but this step could employ other advanced crawling and reverse engineering techniques. Based on the model, the navigation paths that users can follow when exercising the web application are followed; the depth of the followed links can be set. When a new page is encountered then all of the target pages are retrieved and only these are clustered based on schema similarity. The schema of each page is obtained by converting the HTML into a well-formed XHTML (with the JTidy tool) and automatically extracting a DTD from it (with the DTDGenerator tool). The clustering is performed with the edit distance between the pages' DTDs using the Levenshtein method [517]. The similarity threshold can be set. On the clustered navigation paths a differencing algorithms determines the HTML structural changes between page source and targets. The result is a set of candidate components where each one is examined for promising UI elements (e.g., button or tab) that can be converted to Ajax. The authors propose to describe the candidates with the help of a generic Ajax-based representation model (which could take inspiration from static UI mod-

els such as Mozilla's XUL) that can then be used to drive the transformation from the generic model to a platform-specific representation.

7.4 Research Topics

Sections 7.2 and 7.3 have covered a large part of the research landscape of web systems evolution. In Table 7.2, the references of research contributions that are given in bold face have already been discussed in previous sections.

Table 7.2: Examples of "classical" research topics and selected research contributions.

Research topic	Examples of web-related research	F	M	A	T	V
architecture recovery	of web sites **[569]**			•		•
	of web applications **[373] [374]**			•		•
clone detection	in web sites [243]			•		
	in web applications [815] [494]			•		
	in web services (WSDL) **[568]**			•		•
clustering	of web pages via keyword extraction [853]			•		
	of web applications **[601]** [244]			•		
dead/unreachable code	removal of JavaScript code **[560]**			•		
fact extraction	HTML **[569] [909]**	•				
	crawling of RIAs (with Ajax) (Crawljax) **[602]**	•				
	crawling of single-page Ajax **[561]**	•				
	crawling of RESTful web services **[15]**	•				
	J2EE web projects (in WebSphere) [464]	•				
metrics	based on HTML code **[909] [141]**	•		•		
	based on WSDL differencing **[298]**			•		
migration	from static sites to dynamic web applications **[718]**				•	
	from ASP to NSP [375]				•	
	from EJB to web services**[531]**				•	
	from web application to single-page Ajax **[601]**				•	
	to Ajax **[185]**				•	
	to web services **[20]**				•	
	involving MDE **[98] [97] [724]**	•	•	•	•	
refactoring	of web applications design models **[315] [316]** [154]			•	•	
restructuring	of multilingual web sites [852]				•	
	of JSP code for renovation **[933]**				•	
	of web transactions [848]	•	•	•	•	
sequence diagrams	for web applications [38]			•		•
	for web services **[918]** [225]			•		•
slicing	of web applications **[851]**		•			
testing	of web applications **[717]**		•			
	of web services [783] [780]		•			
wrapping	of web applications with WSDL **[441]**				•	
	of COBOL with web services **[782]**				•	

Legend: F: fact extraction, M: modeling, A: analysis, T: transformation, V: visualization.

Many of these topics have their roots in the more general and traditional areas of software evolution and maintenance research. To illustrate this point, Table 7.2 provides some examples of "classical" research topics (first column) along with research that has addressed—and suitably adapted—these topics for the web's domain (second column). The table also identifies if a research's main contribution lies in the areas of fact extraction (F), modeling (M), analysis (A), transformation (T), or visualization (V). For understanding a certain research contribution, it is often beneficial to know the functionalities that its techniques cover and how these functionalities are relating to each other. Reverse engineering approaches are primarily concerned with fact extraction (F) and analysis (A), whereas forward engineering primarily deals with modeling (M) and transformations (T). Both can also employ visualizations (V) to present (intermediary) results. Examples of analyses in reverse engineering are clustering, clone detection, and architecture recovery; examples of transformations in forward engineering are restructuring, refactoring, dead code removal, wrapping and migratwion.

Besides the "classical" research topics covered in Table 7.2, there are also research topics that are unique—or much more pronounced—for web systems evolution. Important topics that exemplify this point—content analysis, accessibility, and browser-safeness—are discussed in the following.

Originally, metrics-based evolution research has exclusively focused on the code and structure of a web system, but it was then realized that evolution can be also tracked by analyzing the (textual) content of a web system with appropriate metrics. An early example of such research is textual analysis of multilingual sites to find matching pairs of pages for different languages [852]. More recently, the evolution of Wikipedia in terms of number of edits and unique contributors has been analyzed [263], and another evolution study looked at legal statements of web sites, analyzing their length and readability [468]. The latter was measured with established readability metrics, namely SMOG and Flesch Reading Ease Score (FRES) [468]. Content analysis is an example of how web system evolution research has continuously broadened its focus.

An interesting example of a research area that has much broader and more prominent focus in web systems evolution than classical evolution research is *accessibility*, which is concerned with "how people with [varying degrees of] disabilities can perceive, understand, navigate, and interact with the web, and that they can contribute to the web" [410]. In fact, accessibility has been a constant throughout the history of web evolution research [467]. In 1999, Eichmann cautioned that for the development of many web systems "little attention is paid to issues of comprehension, navigation or accessibility" [271]. Cesarano et al. tackle the problem of usability of web pages for blind users [168]. They point out that web pages are designed for viewing on a two-dimensional screen while screen-reader tools for the blinds are reading the content in a linear, one-dimensional fashion. Consequently, the reading order should be redefined for blinds. Di Lucca et al. present refactoring heuristics for the automatic reordering of the items on a web page based on structural analysis and on summarization, with the purpose to reduce the "reaching time" (i.e. the time needed to reach the most relevant content of the web page) [245].

In contrast, Berry provides a detailed classification of characteristics of hearing impaired individuals and their respective accessibility issues [99]. These issues became more and more acute in the last years with the popularity of services that exploit the Internet as a medium to transmit voice and multimedia such as Skype and YouTube. Berry also points out that accessibility requirements of sight-impaired individuals can contradict the ones of hearing-impaired individuals. While accessibility research is often focused on seeing and hearing impaired, Boldyreff points out that "web accessibility encompasses a variety of concerns ranging from societal, political, and economic to individual, physical, and intellectual through to the purely technical. Thus, there are many perspectives from which web accessibility can be understood" [125].

An example of a research area that is unique to web site evolution is *browser-safeness*, meaning the requirement that a web system should look and behave the same across different web browsers. This problem is almost as old as the web and it is increasingly challenging to satisfy because of the large number of browsers and browser versions. While browsers try to adhere to (ambiguous) web standards, they are also trying to accommodate legacy web systems that are (partially) violating these standards. Also, JavaScript engines of different browsers differ in more or less subtle ways (e.g., as exposed by the test262 suite [182]).

The state-of-the-practice to address this problem are tools that take screenshots of the web system running in different browsers so that they can be inspected manually for differences. The WebDiff tool identifies cross-browser issues by taking pairs of screenshots of a web page in combination with analyzing the respective DOMs [183]. Variable elements in the DOM that are changing when pages are reloaded (e.g., advertisements) are filtered out, and then a structural comparison of the DOMs is performed. The DOM is also used to guide a visual analysis that does a graphical matching of DOM elements. The tool reports each mismatch so that it can be further inspected by the tool user. The CrossT tool offers a complementary approach that can be applied to a whole web application [600]. CrossT crawls the web system in the same manner using different browsers (i.e., Internet Explorer, Firefox and Chrome are supported), constructing for each a navigation model based on a finite state machine. A state corresponds to a "screen" as observed by the user, transitions are actions executed on the screen such as clicking a button. The constructed models are compared pairwise for discrepancies. Differences are detected in the navigation model as a set of traces that exist in one model but not in the other, and for two state pairs by comparing the underlying DOMs of the states' screens. When comparing DOMs the algorithm has to account for different DOM representations in the browsers.

7.5 Sources for Further Reading

Web systems evolution has been an active research area for over 15 years and consequently there are many resources available for studying. Thus, this chapter, by necessity, only represents a (biased) selection of this growing field, but constitutes a good starting point for the reader to explore further. In this vain, this section identifies the field's key research venues and journals as well as outstanding dissertations.

The annual WSE symposium has targeted this research area since 1999 and has featured the most influential research trends as they have emerged and changed through the years. In fact, many of the references in this chapter are WSE publications. Special issues for WSE 2002, 2006 and 2010 have been published with Wiley's JSEP [461].

Research on web systems evolution can also be found in venues for software maintenance (ICSM, CSMR), reverse engineering (WCRE), program comprehension (IWPC/ICPC), and software engineering (ICSE). Publications in the aforementioned venues are typically dealing with techniques and tools for web systems comprehension, analysis and migration. The communities on web, hypertext, multimedia and documentation/communication are also conducting research on web systems evolution, albeit at a context that is typically broader than what is covered in this chapter. Examples of interesting venues are the ACM Special Interest Group on the Design of Communication (SIGDOC) conference, the ACM Web Science Conference (WebSci), the International Conference on Web Information Systems Engineering (WISE), the International Conference on Web Engineering (ICWE), and the International World Wide Web Conference (WWW). In terms of dedicated journals, there are the ACM's Transaction on the Web (TWEB), Rinton Press' Journal of Web Engineering (JWE), Inderscience's International Journal of Web Engineering and Technology (IJWET), and Emerald's International Journal of Web Information Systems (IJWIS).

Last but not least, there is a growing number of Ph.D. theses that target web systems evolution, testifying that the research community has recognized the relevance of this subfield of software evolution. To name some outstanding dissertations: "Analysis, Testing, and Re-structuring of Web Applications" by Ricca [715], "Reengineering Legacy Applications and Web Transactions: An extended version of the UWA Transaction Design Model" by Distante [251], "Analysis and Testing of Ajax-based Single-page Web Applications" by Mesbah [599], and "Reverse Engineering and Testing of Rich Internet Applications" by Amalfitano [27].

7.6 Conclusions

This chapter described the key research topics and achievements in the domain of web systems evolution. It can be expected that web systems evolution research will continue to be of great relevance due to the fact that more and more software functionality is made available via web-based infrastructure. The impact of HTML5 is already felt in this respect and should further intensify the move towards open web systems. Web technologies are highly dynamic by nature, making them predisposed as a platform for building *highly-dynamic systems* (cf. Chapter 7.6), incorporating features such as customization/personalization, resource discovery, late binding, and run-time composition of services. Thus, future evolution research should provide techniques and tools for reasoning about dynamic properties.

Other evolution drivers that may have a strong influence on future research are the Internet of Things (IoT) and the Web 3.0. IoT would expand the web's reach significantly, incorporating devices of any scale. Web 3.0 would make large-scale semantic reasoning feasible. In a sense, the web will increasingly blend the ecosystems of cyberspace and biological space (cf. Chapter 10). Both IoT and Web 3.0 would open up many new research avenues, but also demand a much more inter/multi-disciplinary approach to research, which does not only address technical concerns, but also others, such as society, law/regulation, economics and environmental sustainability.