Device Agnostic CASS Client

Kari Salo, Udeep Shakya, and Michael Damena

School of ICT, Helsinki Metropolia University of Applied Sciences
{kari.salo,udeep.shakya}@metropolia.fi, mikolucky@gmail.com

Abstract. The growth in take-up of smartphones and tablet devices has made longitudinal and context-aware documenting of daily life easier. The Contextual Activity Sampling is a research methodology for the contextual tracking of activities. To support this methodology, an IT-system called CASS (Contextual Activity Sampling System) was developed. It consists of a backend service and a front-end system. The front-end system needs to run in different devices. Instead of developing a separate software for all major device platforms we designed and implemented a software architecture that is based on HTML5 and enables basic functionalities to run in browsers and enhanced functionalities to run as native applications. Thus CASS usage as a research tool will be widened as it supports a large base of different types of devices from PCs to tablets and smart phones.

Keywords: html5, web technologies, contextual activity sampling, JavaScript framework, cross-platform development, AngularJS, PhoneGap.

1 Introduction

The growth in take-up of smartphones and tablet devices has made longitudinal and context-aware documenting of daily life easier. These devices come with different operating systems, different operating system versions, different screen sizes, different screen resolutions, etc. People have their own favorite smart phones or tablets and it is not feasible to ask them to carry another smartphone or tablet in order to participate into some daily life research. Thus we need to provide a query system that supports most common device platforms.

Let's have a look at a query system called CASS (Contextual Activity Sampling System). The Contextual Activity Sampling is a research methodology for the contextual tracking of activities. To support this methodology, an IT-system called CASS [1] was developed originally in an EU funded program called "Developing Knowledge-Practices Laboratory" which was part of The Sixth Framework Programme / the Information Society Technologies (IST).

Figure 1 illustrates the overall architecture of the CASS, which is a distributed system based on client-server software architecture [2]. The research administrator creates a research using an Admin Console, which accesses the Backend Service. The research is stored into a database which is part of the Backend Service. Furthermore,

A. Marcus (Ed.): DUXU 2014, Part II, LNCS 8518, pp. 334–345, 2014.

the research administrator adds new respondents and sets the sampling strategy. Respondents obtain the surveys related to the research. In order to do so, they use the CASS-Q client application on their smartphones and tablets, which are connected to the internet. Once the survey is completed, it is sent back to the Backend Service. The answers are then stored into the database for analysis.

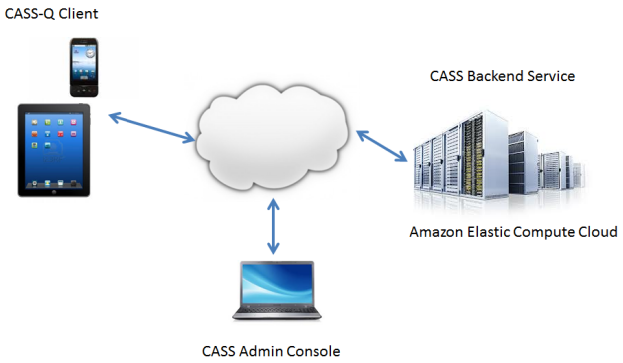

Fig. 1. CASS Architecture

CASS can be tailored to a wide variety of contexts, and has been used for example in studying university students' experiences [3], mobile work [4], and professional product design [5].

We have been further developing CASS Backend capabilities and CASS-Q clients in national Indoor Environment program funded by TEKES (the Finnish Funding Agency for Technology and Innovation). In the CASS-Q area, we have developed native clients for Android platform, Windows Phone 7.5 platform and iOS6 platform. During the project, we realized that developing and maintaining three different CASS-Q versions is far too resource consuming. In addition, we have no support for desktop users.

In CASS we have carefully defined the interface between backend service and front-end client. Thus to be able to support most common user end device platforms we need to redesign and implement CASS-Q part of the architecture. To address our challenge we will have three options:

1. design and implement native applications for most common device platforms
2. design and implement a browser based client for most common browsers
3. design and implement a web technology based application and port it as a nativeapplication for most common device platforms using some of cross platform development frameworks

We have already seen that option 1 is viable when there are enough experienced developers. This, however, is not the case in our organization. We decided to combine options 2 and 3 in order to support users who would respond to queries using laptops, tablets or smart phones. So our goal is to design and implement a software architecture that relies on web technologies and combines options 2 and 3.

2 Methods

In order to reach our goal we divided the project into four phases:

1. initial feasibility study concentrating on finding the critical issues
2. feasibility study on JavaScript frameworks
3. revisit most critical issues and their status
4. develop responsive CASS-Q client using HTML5

The starting point for the initial feasibility study was Android CASS-Q client's functionalities and features [6]. We decided to implement main functionalities using HTML5, CSS3, JavaScript and jQuery. At this point we did not spend time on the software architecture design. Our aim was to implement quickly a rough prototype in order to find the problem areas. After finding the problem areas we studied what other developers had found out and how the W3C standardization was proceeding in those areas.

The second phase concentrated mainly on JavaScript frameworks. Sometimes JavaScript libraries are called JavaScript frameworks. Our definition is that JavaScript libraries provide helper functionalities and JavaScript frameworks define architecture that you are supposed to follow [7]. Our motivation for this second phase was to address two main questions: how to divide large JavaScript programs into manageable modules and how to ensure the maintainability of the code when developers change. First we made an online study to find out what the most common frameworks and what their capabilities based on developer comments. After that we defined selection criteria for the frameworks, selected three most promising frameworks, compared them based on defined criteria and online developer comments, and finally implemented a limited prototype using selected framework.

As the initial feasibility study was carried out late 2012 until early 2013 it was useful to check if the situation concerning critical issues had changed in a year. Therefore, we revisited relevant W3C working group documents and run our test scripts on most common browsers to check what can be implemented and what is still in the planning phase.

Based on findings on earlier phases we were ready to design and implement web technology based CASS-Q client. Web technology meaning combination of HTML5, CSS3, JavaScript, JavaScript libraries and frameworks, and WebAPIs [8-9]. We started by giving priorities to features and functionalities in order to be able to design and implement incrementally the software. We were not strictly following SCRUM methodology, but applying it. The main idea was to first get core functionalities up and running using selected JavaScript framework and then iteratively add problematic functionalities (found during critical issue revisit) one by one. Naturally we developed and tested also those core functionalities incrementally.

3 Results

3.1 Initial Feasibility Study

In the first phase we developed a prototype that was able to send a token (=query id) to backend service, receive a query as an xml message, parse received xml message into four JavaScript arrays, display open text, open number, single choice, multiple choice and number range type of questions, generate answer form for each question type, check that all questions are answered and finally parse an xml-message from answers and send it as http post to backend service.

All the functionality we implemented using standard JavaScript and jQuery 1.7.2. We used jquery.ui.slider for implementing number range as a slider and touchwipe jQuery plugin to handle swipe type of touch gestures.

Communication between CASS-Q client and Backend service is rather simple. Initial query is sent using http get-method containing query id. As a response backend will send an xml message containing query with questions. Finally, the client will return the answers as an xml document in the http post-method's body. All the communication was implemented using XMLHttpRequest-object.

Traversing through xml-document and storing questions in array structures was implemented mainly using jQuery's find and each methods. We used four arrays to store questions and answers:

- survey array for query's general info
- item array for individual questions
- items array containing all item arrays
- answers array containing all answers

In order to manage tokens we used localStorage API [10]. User will receive the token from research administrator and he or she needs to input that into CASS-Q client. The token will be stored permanently until the user deletes it.

We tested our prototype first with Firefox 16.0.1. To test also touch gestures we used Android phone (Samsung Galaxy S2) as a test device. First we tested with the phone's standard browser. Then we used PhoneGap framework to wrap our application as native Android application. After building and deploying the application into phone we tested again. No problems occurred in the tests. Screen captures from the testflow are illustrated in figure 2.

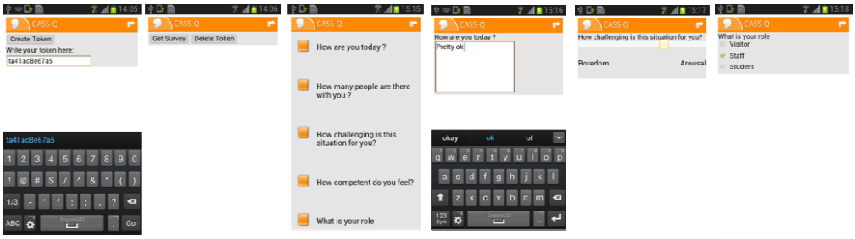

Fig. 2. Test flow after using PhoneGap framework

So far everything worked better than expected. The next step was to concentrate on media type of questions. This means that user should be able to take a photo, record an audio answer or a video. Finally, when sending all the answers also media files should be uploaded to CASS Backend Service.

We started with a simplified solution where users would use some native applications to take a photo or record an audio or video, and CASS-Q client would provide the functionality to find and upload those media files. We used File API, HTML5's <input type="file"> element, FormData interface, and XMLHttpRequest-object. This time we used also jQuery Mobile to scale mobile devices' different screen sizes. The following JavaScript code describes how to read (image) file content into memory, display the file and upload it to server.

```
$(document).on("pageinit", function(){
  $("#getimg").change(function(evt) {
    var files = evt.target.files;
    var file = files[0];
    var reader = new FileReader();
    reader.onload = function(e) {
        $("#myimg").attr("src",e.target.result);
      }
    reader.readAsDataURL(file);
    var fd = new FormData();
    fd.append('image', file);
    var jqxhr = $.ajax({
      url: 'http://users.metropolia.fi/XXXX/ulimg.php',
      data: fd,
      cache: false,
      contentType: false,
      processData: false,
    type: 'POST'}).done(function(data) {
$('#txt').text(data);
    });
  });
});
```

Once again we tested this file selection and upload approach with desktop and mobile browsers. Test results can be found from table 1. This test resulted into a change in CASS Backend Service. We had to develop a new file receive module in order to simplify the file upload from CASS-Q client into the CASS Backend Service.

The next step was to test how to get access to user's local camera/microphone stream. Before developing our own test software we read several online articles about the topic. At this time W3C's WebRTC working group had published their draft version of getusermedia APIs [11]. We tested how to capture snapshots from the webcam connected to laptop. It worked only in Chrome version 22.0.1229.94.

Table 1. Select file and upload test results

Browser	Test result	Notes
Firefox 16.0.1	OK	
Crome 22.0.1229.94	OK	
Safari 5.1.7	Failed	
IE 9.0.8112.16421	Failed	
Samsung Galaxy S2 Native browser	Failed	
Samsung Galaxy S3 Native browser	OK	File selection also provides possibility to select camera, audio recorder or video recorder
iPhone 5 Native browser	OK	File selection also provides possibility to select camera, audio recorder or video recorder

3.2 JavaScript Framework Study

Currently, there are a lot of JavaScript frameworks available to choose from [12]. No doubt one framework will fulfill the needs of a certain project but it might be less beneficial in another one. Hence, it is always better to go with a framework that can provide closely all the features that the remaining frameworks provide. This also helps in avoiding picking different frameworks for different projects.

Without going deeper into the details, the main reason to go the framework way is to have large codebase clean and organized, extensible and maintainable. Those features were needed in developing CASS-Q client. Different frameworks may provide all or part of these main functions in their own specific way. The way the aforementioned main futures are provided has made some frameworks hard to learn in short time. This in turn has affected the size of community they have and hence their growth and maturity. Based on the needs we have for CASS-Q client app implementation, after considering the above core features, we have looked into specific features/functionalities provided by the selected frameworks. In addition, we have studied what the other developers have written [13-15]. Therefore, the final selection criteria considered were first of all the component organization / design patterns followed and data binding ability as well as modularity / code organization. Also extensibility, browser support, the ease of use along with learning curve and documentation were found to be important. Other important criteria included routing / bookmarking capability, ability to work with other libraries, maturity and size of the community, templating mechanism, support for localization and testability.

From the selected frameworks, two are complete to be named Framework and those are AngularJS and Ember and Knockout is a JavaScript library. All the selected frameworks/libraries have both pros and cons but stood out from others based on specific useful feature they have. Ember was selected as it was a complete framework and has a wide footprint in the HTML5 SPA (Single Page Application) industry. AngularJS was selected as it is one of the first to be named a framework and is a

product of Google, which partly ensures its future growth. Knockout was selected as it is a fairly light and applicable to part or all of a project. Those are the general features considered.

Table 2 below shows the key features that we took into consideration while comparing the frameworks that were finally selected.

Table 2. Comparison of frameworks

Features	AngularJS	Knockout.JS	Ember
Learning curve and documentation	Medium learning curve and well documented	Easy learning curve and well documented	High learning curve
Framework/Patterns followed	MVW (Model View Whatever)[1]	Model-View-ViewModel	Pure MVC
2 Way data binding	Yes	Yes	Yes
Modularity/Code organization	Yes	Yes	Yes
Routing/Bookmarking capability	Yes	Yes	Yes
Directives	Yes	No	No
Opinionated	Yes	No	Very much
Ability to work with other libraries	Yes	Yes	Not easy
Browser support	Yes and goes back till IE8	Yes and goes back till IE6	Yes but has some issues with IE8, IE9
Maturity and size of the community	14, 000 Git Hub Watchers	4, 153 Git Hub Watchers	8, 154 Git Hub Watchers
Templating mechanism	DOM Based Templating	DOM or String-Based Templating	Uses Handlebars
Testability	5	4	4
Leadership	Google	Steve Sanderson	TILDE INC.

As shown in Table 2, most of the considered features are fulfilled by AngularJS and that has impacted our selection decision. Since the future extensibility of the CASS-Q client was one of the considerations that was taken seriously, a framework

[1] Early versions of Angular implemented model-view-controller pattern. Newer versions have moved towards model-view-viewmodel pattern. As there is heated debate ongoing of different variations of model-view-controller pattern, Igor Minar (one of the AngulaJS architects) defined that AngularJS follows model-view-whatever pattern. His message was that concentrate on developing great apps instead of debating on minor details [16].

with maturity, fast growth rate, potential owner and expanding community was selected and that was AngularJS.

3.3 Critical Issue Revisit

During the revisit to most critical issues, we found out that in less than a year a lot of progress had happened. The HTML Media Capture specification [17], which defines an HTML form extension, had reached W3C Candidate Recommendation phase and there was a very useful capture attribute defined. The idea behind capture attribute is to use of a media capture mechanism, such as a camera or microphone, from within a file upload control. In practice we could add the following tag into our HTML-document:

```
<input type="file" id="getimg" accept="image/*" capture/>
```

This should offer the user a possibility to define where to get the image: existing image from the memory or use camera to take a photo. We tested this approach with desktop and mobile browsers. Test results can be found from Table 3.

Table 3. Capture image and upload test results

Browser	Test result	Notes
Firefox 24.2.0	Failed	Capture not supported
Crome 32.0.1700.76	Failed	Capture not supported
Safari 5.1.7	Failed	Capture not supported
IE 10.0.12	Failed	Capture not supported
Samsung Galaxy S4 Native browser	OK	
iPhone 5 Native browser	OK	

The Media Capture and Streams specification [19], which defines a set of JavaScript APIs, has reached W3C Working Draft phase. Methods defined in this specification are useful for our purposes. We developed two tests based on this specification:

1. capture image from device's video stream and upload the image to server
2. record audio from device's microphone and upload the audio file to server

For the first test, we defined html document that contained video and canvas tags, and JavaScript file that contained all the functionality. By calling getUserMedia-method with a video as source parameter we were able to get access to device's video stream and pass it as a source for the video tag. Then we added click event handler which captured current frame from the video stream and using HTML5 canvas element's 2D Context interface's drawImage method. Now the image was added to the canvas element. Then we used canvas element's drawImage method to get captured image in encoded format. Then we decoded image, passed decoded image as a parameter to Blob object constructor. Finally, we created FormData object, appended Blob object into the form and used XMLHttpRequest-object to upload image into server. Test results can be found from Table 4.

Table 4. Capture image from video stream and upload test results

Browser	Test result
Firefox 24.2.0	OK
Crome 32.0.1700.76	OK
Safari 5.1.7	Failed
IE 10.0.12	Failed
Samsung Galaxy S4 Native browser	Failed
iPhone 5 Native browser	Failed

For the second test we used Matt Diamond's Recorder.js plugin [18] for capturing audio stream from device's microphone, recording from the stream and capturing recorded audio as a Blob object. Then we uploaded audio recording in a similar way like in previous test. This worked only in Chrome v. 32. It was supposed to work also in Firefox v. 25, which was not available during tests. The second test was utilizing Web Audio API and MediaStream Recording W3C Working Drafts. Working draft document may be updated, replaced or obsoleted by other documents at any time.

3.4 Responsive HTML5 Client

Based on feasibility studies and practical experiments, we made a decision to design and implement a CASS-Q client using model-view-control pattern based software architecture, which is very typical for this type of GUI intensive system [19]. AngularJS Framework and its code modularization following Model-View-ViewModel pattern supported well our architecture selection. As HTML5 and JavaScript media capture related APIs are still at a very immature stage, we divided the client into two subcomponents: core CASS-Q and device CASS-Q (see Figure 3). If audiovisual features are needed in near future then device CASS-Q is implemented using PhoneGap cross platform development framework together with AngularJS.

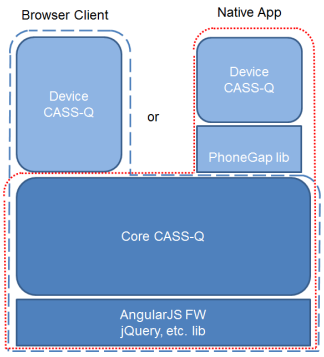

Fig. 3. CASS-Q Architecture

At the time of writing this article, we have developed a responsive core CASS-Q prototype that is able to send a token (=query id) to Backend Service, receive a query as an xml message, parse received xml message into three JavaScript objects, display open text, open number, single choice, and number range type of questions, generate answer form for each question type, check that all questions are answered and finally parse an xml-message from answers and send it as http post to Backend Service. In implementation we used AngularJS 1.2.10, jQuery 1.10.2 and Bootstrap 2.0.2.

We tested prototype using desktop, tablet and smart phone browsers. Test results are described in Table 5. For us the test result was OK even if the input-tag's new number type attribute was not supported as there are workarounds, like jQuery UI Spinner widget.

Table 5. Core CASS-Q prototype test results

Browser	Test result	Notes
Windows PC Firefox 26.0	OK	HTML5 number type partial support
Windows PC Crome 32.0	OK	
Windows PC IE 10.0	OK	HTML5 number type partial support
iMAC Safari 6.0.5	OK	
MacBook Pro Safari 7.0.1	OK	
Samsung Galaxy S4 Native browser	OK	
Nexus7 Chrome32.0	OK	
iPhone 5s Mobile Safari	OK	HTML5 number type partial support
iPad mini Mobile Safari	Failed	HTML5 number type partial support, not scaling properly
Nokia Lumia 920 IE Mobile	Failed	several problems

The next step was to develop a device CASS-Q module. The first prototype was based on the following idea: the routeProvider object will select which partial to use based on the availability of Cordova library, define separate controller for each media type, and define in each partial which controller to use. While writing this article, we implemented the camera usage to test our idea. So we had a core CASS-Q and device CASS-Q modules and also Cordova (PhoneGap) JavaScript library. We used Eclipse with ADT-plugin to build our CASS-Q prototype as an Android native application. We installed and ran the application in Samsung Galaxy S4 and Nexus7 devices and it worked as expected.

4 Discussion

We could have selected any popular JavaScript framework as the most important features are code modularization and proper structure in order to ensure the maintainability of the code. This is especially important for us as most of the development work is done by students who will at the most work one year and then

we will assign new students who will continue with the development work. So far AngularJS has fulfilled our needs.

The Core CASS-Q contains backend communication, XML-parsing, query and answer data structures. Core CASS-Q works in most common browsers and thus supports a large base of different type of devices from PCs to tablets and smart phones. In many cases this is sufficient for documenting of daily life activities. Thus we are lowering the barrier to use CASS as a research tool as the respondents can now use their own tablet or smartphone or laptop to answer the queries.

If the daily life activity research requires also audio or video feedback from the respondent then today's HTML5 is not yet ready for that. Standardization workgroups look for new ways to design media related APIs and different browser engines support different features. At the moment, the only reliable approach is to rely on some proprietary libraries that provide access to mobile device's native level APIs. In order to utilize these libraries there are two approaches: widgets and cross platform frameworks, like PhoneGap. In the current version, we have added audio, video and camera capture support using PhoneGap framework, which means that instead of browser based CASS-Q there is a native app to be installed into respondent's mobile device. If respondents will use browser based CASS-Q client and query includes media related questions then the user will see the question and message that his CASS-Q is not supporting this type of questions. However, this does not prevent respondents to answer this type of query as "media not supported" info will be passed to CASS backend with rest of the answers.

Device CASS-Q contains today media elements and in the future it will contain also device specific APIs, like device orientation, web telephony, network information, proximity events, etc. Device CASS-Q can also be targeted to widget engines and in those cases we will just replace PhoneGap library with a similar library that enables device specific features. Thus, we are also prepared for new mobile platforms like Tizen.

Acknowledgements. This work has been carried out and financed as part of the RYM Indoor Environment program (project number: 1064/31/2011, wbs 462054) funded by Tekes (the Finnish Funding Agency for Technology and Innovation) where Metropolia worked as a subcontractor for the University of Helsinki. The contribution and cooperation of the University of Helsinki, Faculty of Behavioural Sciences, Department of Teacher Education are gratefully acknowledged.

References

1. Muukkonen, H., Hakkarainen, K., Jalonen, S., Kosonen, K., Heikkilä, A., Lonka, K., Inkinen, M., Salmela-Aro, K., Linnanen, J., Salo, K.: Process-andcontext-sensitive research on academic knowledge practices: Developing CASS-tools and methods. In: Chinn, C., Erkens, G., Puntambekar, S. (eds.) Computer Supportive Collaborative Learning: Mice, Minds, and Society. Proceedings of the Seventh International Computer Supported CollaborativeLearning Conference, pp. 541–543. ErlBaum, Mahwah (2007)

2. Garlan, D., Shaw, M.: An Introduction to Software Architecture. In: Ambriola, V., Tortora, G. (eds.) Advances in Software Engineering and Knowledge Engineering, vol. I, pp. 1–40. World Scientific Publishing Company, New Jersey (1993)
3. Muukkonen, H., Hakkarainen, K., Inkinen, M., Lonka, K., Salmela-Aro, K.: CASS-methods and tools for investigating higher education knowledge practices. In: Kanselaar, G., Jonker, V., Kirschner, P., Prins, F. (eds.) Proceedings of the 2008 International Conference for the Learning Sciences (ICLS), vol. 2, pp. 107–115. International Society of the Learning Sciences (2008)
4. Muukkonen, H., Toikka, S., Vartiainen, M.: Contextual Activity Sampling System (CASS) for tracking activities, places, and affects of mobile work. In: Abstract book of 15th Conference of the European Association of Work and Organizational Psychology Maastricht, May 25-May 28, pp. 570–571 (2011)
5. Seitamaa-Hakkarainen, P., Laamanen, T.-K., Viitala, J., Mäkelä, M.: Materiality and Emotions in Making. Techne Series A 20(3), 5–19 (2013)
6. Google Play, https://play.google.com/store/apps/details?id=fi.metropolia.cass.main&hl=en
7. Rich JavaScript Applications – the Seven Frameworks, http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne-of-js-2012/
8. WebAPI, https://developer.mozilla.org/fi/docs/WebAPI
9. JavaScript APIs Current Status, http://www.w3.org/standards/techs/js#w3c_all
10. Web Storage, http://www.w3.org/TR/webstorage/
11. Media Capture and Streams, http://dev.w3.org/2011/webrtc/editor/archives/20121115/getusermedia.html
12. ToDoMVCHelping you select an MV* framework, http://todomvc.com/
13. The Battle of Modern JavaScript Frameworks, http://www.softfinity.com/blog/the-battle-of-modern-javascript-frameworks-part-i/
14. Choosing a JavaScript MVC Framework, http://www.funnyant.com/choosing-javascript-mvc-framework/
15. A Comparison of Angular, Backbone, CanJS and Ember, http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/
16. MVC vs MVVM vs MVP, https://plus.google.com/+AngularJS/posts/aZNVhj355G2
17. HTML Media Capture, http://www.w3.org/TR/html-media-capture/
18. Media Capture and Streams, http://www.w3.org/TR/mediacapture-streams/ A plugin for recording/exporting the output of Web Audio API nodes, https://github.com/mattdiamond/Recorderjs
19. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, vol. 1, pp. 125–144. John Wiley & Sons Ltd., Chichester (1997)