专题：不断变化的Web应用程序

使用客户端重构的个性化Web可访问性

Alejandra Garrido,Sergio Firmenich,Gustavo Rossi,and Julián Grigera
阿根廷拉普拉塔国立大学
Nuria Medina-Medina 西班牙格拉纳达大学
Ivana Harari 阿根廷拉普拉塔国立大学

According to W3C accessibility standards, most Web applications are neither accessible nor usable for people with disabilities. Developers often solve this problem by building separate accessible applications, but these are seldom usable and typically offer less functionality than the original. Another common solution is to maintain a single application, but create an accessible view by applying on-the-fl transformations to each requested page －a solution that rarely suits all audiences. A third solution is described here: let users improve Web accessibility in their client browsers through interface refactorings, which offer many customized, accessible views of a single application.通过W3C可访问性标准，大部分Web应用程序对于残疾人群都是不可访问且不可用的。开发者经常通过创建独立的可访问性应用程序来解决这个问题，但是那很少可用，并且通常提供的功能比原来少。另一个常用的解决方法是保持单一的应用程序，但是通过对每个请求的页面采用FL变换来创建一个可访问的视图，是一个难以适用于所有用户的解决方案。这里描述的是第三种解决方案：让用户通过界面重构提高他们客户端浏览器的Web可访问性，它提供了许多单个应用程序定制的可访问视图。

1 Introduction

Refactoring was originally conceived as a technique to improve software’s internal qualities －-such as understandability and maintainability －while preserving semantics.1 In prior work, we adapted the refactoring approach to improve a Web application’s external attributes, such as usability.2 These Web refactorings consist of small navigation or interface transformations that enhance perceivable aspects of Web applications, such as user interaction and content presentation, while preserving functionality. Refactorings can also solve accessibility and usability problems for disabled users.3 Still, it’s usually impractical to address interface improvements for all audiences because disabilities can vary dramatically in nature (visual, cognitive, or motor), severity (blindness, color blindness, or strabismus) and extent (total or partial). In such different contexts, “one for all”is barely feasible.

重构的初衷是提高软件内部质量的一种技术－－如可理解性和可维护性，同时保留语义1。在以前的工作中，我们采用了重构方法来提高Web应用程序的外部属性，如可用性2。这些Web 重构包括小的导航或界面转换来增强Web应用程序的感知性方面，比如用户互动和内容呈现方式，同时保留功能。重构也可以解决残障用户3 的可访问性和可用性问题。还有，它对于解决所有观众的界面改进问题通常是不切实际的，因为残障用户在性质（视力障碍、认知障碍或运动障碍）、严重性（失明、色盲或斜视）和程度（完全或部分）上显著不同。在如此不同的上下文中，”所有人“是不可行的。

When applying refactoring to improve internal qualities, developers decide which transformations to apply and where, because they’re the ones benefiting from the improvement. As Brian Foote and Joseph Yoder put it, “Who better to resolve the forces impinging upon each design issue as it arises, as the person who is going to have to live with these decisions?”4 Moreover, different developers might prefer alternative solutions for the same “bad smell”(that is, the design problem that motivated the refactoring1). Following on this general philosophy, we believe that end users should be able to tailor a website’s interface for their own benefit.

当应用重构来提高内部质量时，开发者决定应用什么改变，以及在哪儿应用，因为他们受益于改进。正如Brian Foote和Joseph Yoder所说，“当问题出现时，谁能更好地解决冲击每个设计问题的压力？谁又是不得不依靠这些设计的人？4”此外，不同的开发者或许更喜欢可替代的解决方案，对于相同的“坏味道”（即诱发重构1的设计问题）。根据这个普遍的哲理，我们相信那些终端用户能够根据他们自己的利益定制网站的界面。

We propose empowering users (or close representatives) with the ability to select, in their client browser-

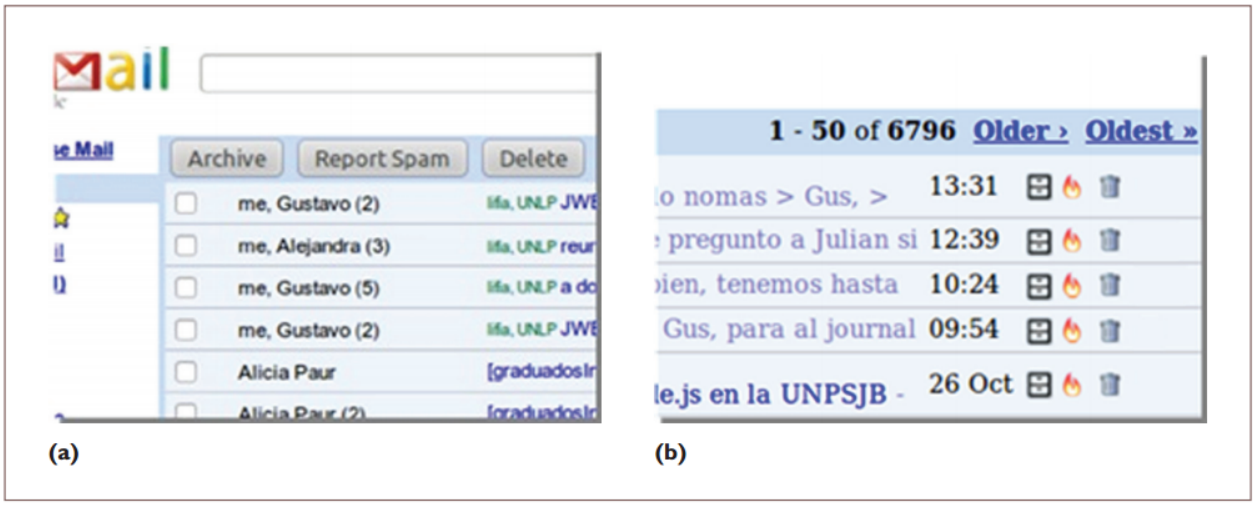

Figure 1: **Applying refactoring to Gmail. The Gmail interface (a) before and (b) after applying Distribute Global Menu to address accessibility issues with the checkbox and operations on top.(Gmail logo reprinted with permission.)对Gmail应用重构。Gmail界面前者(a)和后者(b)分别应用分布式全局菜单来解决复选框和顶部操作上的可达性问题。（Gmail图标已得到转载许可）**

s, their own interface refactorings for each site they access. We call our approach Client-Side Web Refactoring. CSWR allows for the automatic creation of different, personalized views of the same application to solve the particular bad smells that each user recognizes. (Developers should continue to focus on addressing general usability problems on the server-side, however,and reach the minimum level of accessibility,or “A”).
我们建议授权用户（或者是类似代表）有选择的能力，在客户端浏览器，对自己可访问的每个站点进行页面重构。我们称我们的方法为“客户端Web重构”（简称CSWR）。CSWR允许自动创建同一个应用程序的不同的、个性化界面来解决每个用户认为的特定的坏味道。（开发者应该继续致力于解决服务器端的一般的可用性问题，而且达到可访问性的最低水平或“A”）。
Here, we describe CSWR and a case study we ran with visually impaired users (though a similar solution could be applied for other disabilities).
在这里，我们描述CSWR和一个视障人士参与的实例研究（尽管类似的解决方案可应用于其他残障人士）。

2 Refactoring Example 重构示例

Figure 1a shows the inbox in Gmail, Google’s email reader. Gmail includes checkboxes on the left that let users select several emails, which is handy for applying an action to all of them.However, for visually impaired people using a screen reader, it’s uncomfortable to have to go back to the checkbox at the line’s beginning to select emails after reading the line, and then go back to the top to apply an operation after selecting the emails; they report this as a “bad accessibility smell”(an accessibility problem that motivates a refactoring).
图1a显示的是Gmail（Google的邮箱阅读器）的收信箱。Gmail包括左侧的复选框，可以让用户选择多个邮件，这很方便对所有邮件进行操作。然而，对于使用屏幕阅读器的视障人士，不幸的是观众读完阅读栏之后不得不返回到阅读栏的顶部复选框去选择邮件，并且选择完邮件之后又不得不返回到顶部去应用此操作；他们报告这种现象为“坏可闻”（诱发重构的一个可访问性问题）。
A refactoring that solves this bad smell is Distribute Global Menu, which distributes a menu of actions affecting a list of elements to each element individually. This eases the local application of an operation because it requires only a single click immediately after the element is read. Figure 1b shows the result of applying this refactoring to the Gmail inbox.The set of actions was removed from the top and attached to each email in the form of icons(each with an alternative text).
诱发这种坏味道的一个重构是分布式全局菜单，它分配一个单独影响每个元素的元素列表菜单。这使一个操作的本地应用程序变得更轻松，因为当一个元素

被阅读后需要仅仅一个简单的立即点击。图1b显示对Gmai收信箱应用这个重构的结果。这一系列动作是从顶部移走并且链接每个图标形式的邮件（每个对应一个可替代文本）。
However, some users who report the same bad smell are more comfortable using contextual menus, so the set of actions isn’t read with every email. For them, the Contextualize Global Menu refactoring is more appropriate. Also,experienced users prefer to keep the global menu so they can operate on several emails at once; for them, using the Postpone Selection refactoring will move the checkboxes to the last column.
然而，一些报告类似坏味道的用户能够更舒服地使用上下文菜单，所以这一系列动作不能被每个邮件读到。对于他们，上下文全局菜单重构更合适。而且有经验的用户更喜欢保持全局菜单，所以他们能一次操作多封邮件；对于他们，用推迟选择重构将移动复选框到最后一列。

3 Client-Side Web Refactoring 客户端的Web重构

As this example shows, a Web refactoring changes a Web application’s navigation structure or look and feel, preserving its content and operations while removing bad usability or accessibility smells. In previous work, we used Web refactorings to enhance navigation and presentation during the development life cycle. 2,5 We can generate a complete new version of an application with a specific aim －such as a mobile version －by systematically applying and composing refactorings. Here, we propose a similar approach to improve accessibility, where refactoring is applied after deployment and during actual use of the Web application, altering the interface in the browser itself.
正如这个例子所述，Web重构改变Web应用程序的导航结构或外观,去除可用性和可访问性的坏气味的同时保留它的内容和操作。在之前的工作中，我们用Web重构增强了开发生命周期2,5 中的导航和演示。我们能通过系统应用和组合重构，根据特定的目标生成一个应用程序的全新版本–例如一个手机版本。在这里，我们提出一个相似的方法提高可访问性，即Web 应用程序部署和实际应用期间使用重构，改变浏览器本身的界面。
Our CSWR approach has two key benefits:
我们CSWR方法有两个主要好处：
? Simpler maintenance －developers maintain a single core application, applying Web usability refactorings that address a general audience, while different refactored versions can be created by and for different users.
? 维护更简单－开发者维护一个单核心应用程序，其为解决普通观众问题应用Web可用性重构，当然不同的重构版本被创建且服务于不同的用户。
? Architecture independence －developing a CSWR requires little (if any) knowledge of the target application’s underlying architecture.
? 结构独立－开发一个CSWR需要很少（如果有的话）掌握目标应用程序的底层结构。

The engine behind CSWRs uses a client-side adaptation framework that aims to adapt existing applications by changing their DOM structure. 6 CSWRs are implemented by specializing the class AbstractRefactoring (provided in our framework) and redefining the method adaptDocument() with the refactoring’s mechanics.For example, consider the Split Page refactoring, which solves the problem of a saturated, complex webpage by dividing it into a set of simpler pages or sections. 2 In this case, the method adaptDocument() receives the DOM elements that represent disjoint page sections as parameters and creates a new page for each section, replacing the original page’s contents with an index to the new pages (see the bottom of Figure 2a).
CSMRs背后的引擎使用一个客户端的适应框架，旨在通过改变其DOM结构来适应现用的应用程序6。CSMRs通过实现类AbstractRefactoring（由我们的框架提供）并且用重构机制重定义方法adaptDocument()来实施。例如，考虑拆分页面重构，它通过将其划分成一组简单的页或段解决一个饱和、复杂页面的问题。在这种情况下，方法saturated()接收代表不相关页面部分的DOM参数作为参数，并且创建为每个部分创建一个新的页面，使用新页面索引代替原页面的内容(参加图2a的底部)。
To apply the Split Page refactoring to a specific page, you must first create an instance of SplitPage (such as the two instances in the middle level of Figure 2a), passing as parameters
应用拆分页面重构到一个特定页面，你必须首先创建一个SplitPage实例(如图2a 中部的两个实例)来传递参数
? a URL identifying the target page or a URI pattern for a set of pages of the same site(such as all Gmail pages in Figure 2a’s code);
? 一个识别目标页的URL或一组相同站点页面的URI模式（如图2a代码中的所有Gmail页面）and
和
? instances of the class Section , which contain DOM elements specified through xPath expressions. Because DOM elements might not always be identified through xPath queries based on attributes, absolute xPath expressions from the DOM root might be

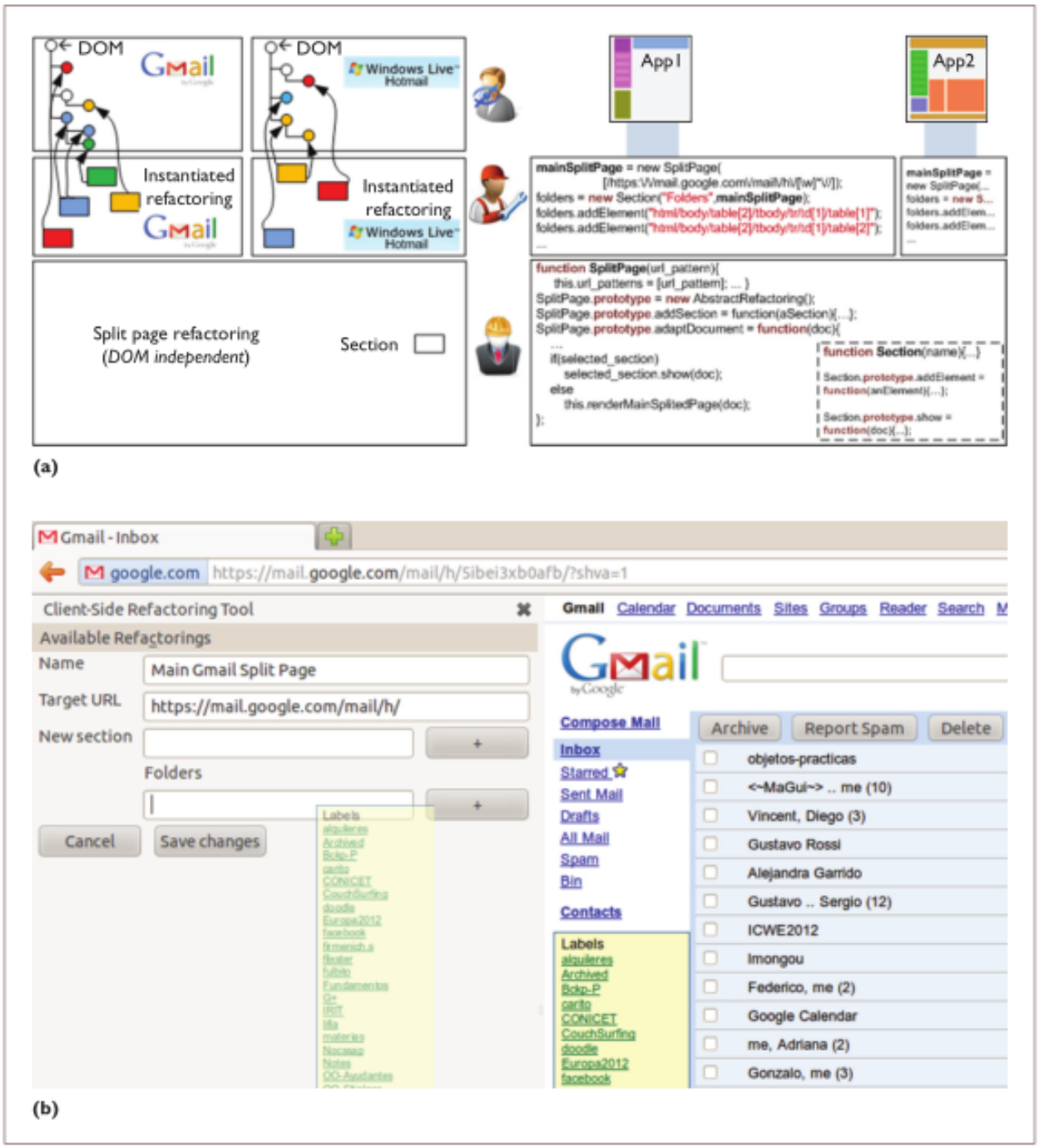

Figure 2: **Client-Side Web Refactoring. a) Split Page refactoring levels: the generic implementation at the bottom, two instances (for Gmail and Hotmail) in the middle, and the result on each DOM at the top. b) Split Page instantiation: the intermediary drags Gmail's Labels to create a new Foders section.客户端网络重构。a) 分页重构层次：底部是类的实现，中间是两个实例（对Gmail和Hotmail），顶部是每个DOM结果。b)分页实例：中介拖动Gmail标签来创建一个新的目录部分。**

required.
? 类部分的实例，其包括通过xPath表达式指定的DOM元素。因为DOM元素可能并不总是通过基于属性识别的xPath查询指定的，来自DOM根的绝对xPath表达式或许被需要。
The three levels in Figure 2a correspond to the three steps involved in the refactoring process, which can involve three user roles (pictured in the middle column):
图2a中的三个等级对应重构过程中的三个步骤，其包括三个用户角色（图中中间列）：
? The JavaScript (JS) programmer creates new refactorings by writing a parameterized script that reuses components offered by our framework's refactoring engine.
JavaScript(JS)程序员通过写一个参数化脚本创建新的重构，该脚本重用被我们框架的重构引擎提供的组件。
? An intermediary instantiates refactorings for a specific website. (The JS programmer can play this role as well). The refactorings are instantiated either by writing code (as in Figure 2a) or using our refactoring tool (as in Figure 2b). This graphical tool lets the intermediary point-and-click on the target page to select the components that act as values for each refactoring parameter. Figure 2b,for example, shows an instantiation of SplitPage, where the group of email labels is dragged to a Folders section that will go into a new page.
? 一个为特定网站的中介实例化重构。（JS程序员也能扮演这个角色。）重构可通过写代码（如图2a）或用我们的重构工具（如图2b）实现实例化。这个图形化工具让目标页上的中介点击来选择作为每个重构参数值的组件。例如，图2b显示SplitPage的一个实例，这里邮件标签组被拖到进入新一页的目录部分。
? End users install instantiated refactorings by choosing them from an accessible menu in their Web browsers; from then on, they can access the refactored website configured to their needs. Unlike traditional refactoring, we added this new step (separating it from instantiation) so that handicapped users unable to code a script or use a graphical refactoring tool can still be part of the process by choosing their own refactorings.
? 最终用户通过从他们Web浏览器中的可访问菜单选择来安装实例化重构；从此，他们可以访问按他们需要配置的重构网站。不同于传统的重构，我们添加新步骤（从实例中分离）使不能写脚本或用图形化重构工具的残疾用户也能通过选择他们自己的重构来成为过程的一部分。
CSWR's power comes not only from letting end users choose specific refactorings, but also from letting intermediaries compose refactorings in different ways. CSWR composition is done at instantiation time, and requires special handling because refactorings can interfere with each other. Similar to code refactorings, an applied refactoring might invalidate the next refactoring's preconditions. In this case, the preconditions of CSWRs are the existence of the DOM elements specified as parameters (xPath expressions). Thus, an applied refactoring might invalidate subsequent refactorings if it changes the xPaths that identify their target elements. For example, if Split Page and Distribute Global Menu are both instantiated on the original DOM's elements, and Split Page is applied first,the Distributed Global Menu won't find its target elements in their original xPath location.
CSWR的动力不仅仅来自于让最终用户选择特定重构，也来自于让中介用不同的方式组合重构。CSWR在实例化时候完成组合，并且需要特殊处理，因为重构能互相干扰。类似于代码重构，一个应用重构或许在下一个重构的前提下无效。在这种情况下，CSWR的前提是被指定为参数（xPath表达式）的DOM元素实体。所以，如果应用重构改变识别目标元素的xPath,它或许会使后续重构无效。例如，如果拆分页面和分布式全局菜单都在原来的DOM元素上实例化，并且拆分页面先被应用，分布式全局菜单不会在原来的xPath定位式中找到目标元素。
Our refactoring tool helps intermediary users correctly compose CSWRs so that they can create and distribute a complete, accessible version of an application as a composition of CSWR instances. When an intermediary user selects several refactorings to compose, the tool creates and suggests a possible sequence, first by placing structural refactorings (such as Split Page),then refactorings that adapt specific DOM elements (such as Distribute Global Menu), and finally by placing DOM-independent refactorings (such as Replace Image with their Alt Text).CSWRs are thus instantiated in order, and users can specify certain noninterfering CSWRs to be independent. With this information, the tool creates the menu of optional CSWRs, so that when end users install a composed set of CSWRs, they can still choose to activate or deactivate independent CSWRs individually.
我们的重构工具帮助中介用户正确地构成CSWR以至于他们能创建和分配一个应用程序完整的、可访问的版本作为CSWR实例组合。当一个中介用户选择若干重构去组合的时候，工具创建并推荐一个可能的序列，首先实行结构重构（如拆分页面），然后重构以适应特定的DOM元素（如分布式全局菜单），最后实行DOM独立重构（如用他们的Alt文本替换图像）。CSWR因此被按顺序实例化，用户能指定特定互不干扰的CSWR进行独立化。有了这些信息，该工具创建可选的CSWR菜单，这样当最终用户安装一套CSWR组合是，他们仍然可以单独地选择启动或顶

用独立的CSWR。