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

***Abstract*—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。**

I. 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 browsers, 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）。

II. METHODOLOGIES

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).

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). 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.