Just Do It: Fast and Easy Mobilization of *Spot Tasks* in Web-based Enterprise Applications

Uma Parthavi Moravapalle and Raghupathy Sivakumar
Georgia Institute of Technology, Atlanta, GA

ABSTRACT

In this paper, we consider the problem of mobilizing *Spot Tasks*, a special category of workflows within web-based enterprise applications. Spot tasks are simple workflows that can be finished by interacting with only one page of the application. We present *Taskr*, a do-it-yourself mobilization solution that users, regardless of their skills, can rely on to mobilize their spot tasks in a robust fashion. *Taskr* uses remote computing with application refactoring to achieve code-less mobilization and allows for flexible mobile delivery wherein users can execute their spot tasks through Twitter, Email or a native mobile app. We implement a prototype of *Taskr* and show through user studies that it has the potential to reduce task burden significantly.

ACM Reference Format:
Uma Parthavi Moravapalle and Raghupathy Sivakumar Georgia Institute of Technology, Atlanta, GA. 2018. *Just Do It:* Fast and Easy Mobilization of *Spot Tasks* in Web-based Enterprise Applications. In *Proceedings of 19th International Workshop on Mobile Computing Systems and Applications (HotMobile'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3177102.3177117

1 INTRODUCTION

The adoption of mobile devices, and in particular smartphones, has grown steadily over the last decade. Fifty-one percent of enterprise workers today use mandated apps for their business on their phones [2]. Seventy-seven percent of the workers rely on their personal smartphones to perform their work [4]. One of the key drivers of the adoption and use of smartphones is the self-perceived increase in productivity. Employees self-reported getting an hour of time back by relying on smartphone apps for their work. Intriguingly, employees were relying as much on company-issued mobile apps as they were on *bring your own application* apps [7].

Now consider an enterprise worker, Alice, who is a field salesperson. An average enterprise runs 400+ applications for its business operations. Alice is likely to interact with many of these applications, with examples ranging from Oracle HR, SAP ERP, Microsoft Sharepoint, and Salesforce CRM. If Alice desires to do some of her Salesforce tasks on her smartphone when she is away from her desk, she currently has to be dependent on either Salesforce releasing a mobile app or her employer building a custom mobile app that taps into the Salesforce APIs. In both cases, not only does the mobile app for Salesforce need to exist, but her specific task also has to make the cut through the de-featurization process necessary for mobilization, and has to be achievable with minimal burden within the design of the mobile app.

Interestingly, in spite of the increasing adoption of mobility in enterprises, studies show that over eighty percent of enterprise mobile apps are abandoned after the first use [1]. In this context, we identify a category of tasks called *Spot Tasks*, and present a strategy wherein Alice can perform the desired mobilization *herself* and *without requiring any support from either the application vendor or the enterprise*.

We define spot tasks as tasks that can be accomplished by the users interacting substantively with the desktop application *only on a single page*. The interaction on that page could be in the form of read, act, and navigate actions. Also, that specific page could be arbitrarily anywhere within the application's navigation tree. While we relax these definitions in subtle ways later in the paper, we also show how even such a constrained definition can support a wide variety of enterprise task profiles. For example, consider a purchase approval task on a typical SAP SRM (supplier relationship management) application. This could require the user to login and authenticate herself, navigate to"My Work", navigate to "Purchase Management", navigate to"Requisition Approvals", see a list of approval requests, check on those requests that need to be approved, click on the "Approve" button, and finally logout of the application. In this example, the first sequence of pages visited is for navigational purposes while the purchase request review and approval are done on a single page. Thus, we categorize such a task as a spot task.

Spot tasks are limited in capabilities, but have several critical advantages that make them an interesting candidate for a mobilization strategy. We present a mobilization solution called *Taskr* to mobilize spot tasks that exploits these advantages and delivers the following properties: (i) Configuration by doing: *Taskr* allows the user to perform the mobilization herself regardless of the user's technical skills. All *Taskr* requires for the mobilization of a spot task is for the user to be able to *perform the spot task* on the desktop application; (ii) Programmatic APIfication: Once the user configures what needs to be mobilized, *Taskr* programmatically creates the necessary APIs using purely a front-end strategy[1] that requires no access to source code from the application vendor, or even special provisions by the enterprise; (iii) Flexible mobile delivery: Since

Corresponding author: Uma Parthavi Moravapalle (parthavi@gatech.edu)

This work was funded in part by National Science Foundation grants IIP-1701115, CNS-1513884, and CNS-1319455, and the Wayne J. Holman Endowed Chair.

HotMobile'18, February 12–13, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5630-5/18/02…$15.00
https://doi.org/10.1145/3177102.3177117

[1] We elaborate later in the paper, but at a high level this involves relying on a remote-computing based approach to create the APIs.

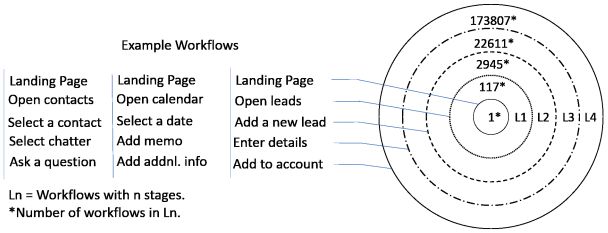


Figure 1: Complexity of the Salesforce desktop application

spot tasks are restricted to a single interaction page, and *Taskr* further imposes limits on the amount of content and actions mobilized on the interaction page, it allows for flexible delivery mechanisms on the smartphone. *Taskr*, specifically, allows the user to consume the mobilized tasks through Twitter (direct messaging), Email, and a Native Mobile App.

We implement *Taskr* on an AWS backend and an Android frontend, and conduct preliminary user experiments to evaluate its performance. The results are promising and show that not only does *Taskr* reduce the actions required to complete tasks (by over 35%) but also that users are more satisfied completing spot tasks with *Taskr* compared to the desktop or the mobile browser (by over $7x$). The rest of the paper is organized as follows: We define spot tasks in Section 2 and introduce *Taskr*'s design in Section 3. We then evaluate it in Section 4 and discuss related work in Section 5. Finally, we discuss some issues with *Taskr* and conclude in Section 6.

2 MOBILIZATION AND SPOT TASKS

2.1 Mobilization and Defeaturization

Enterprise desktop applications are complex and allow a wide variety of business functions. These applications support a large number of workflows - wherein each workflow represents a goal-oriented series of actions taken by the user[2]. Considering the constraints of the smartphone, it is not feasible for a mobile app to support all the desktop workflows. Therefore, the desktop application has to be *defeaturized* before it can be mobilized. For example, The Salesforce CRM web application has over a 180K navigational workflows at a depth of 4 levels (see Figure 1). On the other hand, in Salesforce1 mobile app (the mobile version of Salesforce CRM), there are only 48 navigational workflows at the first level (as opposed to 117 in the desktop version).

Enterprises typically defeaturize at the following granularities: (i) *The entire web application along with all the features are retained in the mobile app.* Considering the desktop application as a large collection of pages, the structure of the pages within the application is largely maintained. This granularity is chosen when all the features within the application are heavily used; (ii) *A subset of features from the original application, carefully chosen either by the enterprise or the vendor, are mobilized.* The features to be mobilized are chosen based on how heavily they are used and the requirements of the user's job functions. With this strategy, the structure of the pages among the application is largely maintained, while reducing the number of features on any given page; (iii) *A mobile-first approach that uses APIs provided by the application to build the mobile app ground up.* This approach can only mobilize those features that have been exposed as APIs; (iv) *A sequence of features that constitute different steps of a single workflow are mobilized.* In this case, once the user starts the workflow on the mobile device, only the features relating to this workflow are presented, thereby decreasing the effort of finding a feature.

[2]For example, Salesforce has about 180K navigational workflows with just 4 navigational steps

2.2 Spot Tasks

In this paper, we identify another potential defeaturization granularity - *Spot Tasks*. A spot task is a simple linear workflow within an enterprise application where-in all the user interactions are only performed on one page of the application. However, this page can be buried deep within the complex application and the navigational effort required to reach that particular page may be high.

UI elements within an application page can be classified as: (i) READ: elements that carry content that is only consumed by the user (e.g., text content of an article); (ii) ACT: elements through which the user writes some parameters in the web application (e.g., text boxes to enter values, dropdown lists, etc.); and (iii) NAV: elements that progress the workflow to the next stage (e.g., links, submit buttons, etc.); For a spot task, each stage of the workflow, except the last stage, has only one NAV element and the final stage of the workflow can have READ/ACT/NAV elements. In other words, if the presence of READ, ACT, and NAV elements in a stage is denoted as R, A, and N, respectively, and the end of a stage is denoted as X, the spot task can be described using a regular expression as follows:

$$ST = [NX] * R?A?N?X$$

Note that even such a constraining definition of spot tasks still covers a substantial number of workflows within enterprise applications. We identify 45 spot tasks within 9 enterprise applications in Section 4. For example, checking the revenue on Salesforce, adding a vendor on Quickbooks, and viewing the available vacation days on Oracle Peoplesoft are all spot tasks (assuming the user is logged in).

Spot task variations: In this paper, we further expand the definition of spot tasks to also account for workflows with fixed (non-variable) inputs along all the stages except the last stage. The non-variable inputs allow for the hard coding of the ACT actions needed to reach the final screen where the user actions are performed. If the user is required to enter a username and password before executing a workflow, then all of the previous examples are still spot tasks under this definition (username and password are fixed values) [3].

2.3 Mobilizing Spot Tasks

The granularities at which mobilization has traditionally been performed necessitate the enterprises to invest significant resources and employ developers with specialized skill sets. Further, the resultant mobile apps are constructed in a one-size-fits-all fashion and are unlikely to address the needs of the entire user base within an enterprise. Thus, for many users, there will exist workflows that the resultant mobile app (i) will not support at all; or (ii) have a considerably increased task burden to perform.

[3]We provide more examples of spot tasks in Section 4

However, if there exists a mobilization solution that the users themselves rely on to create an app that is custom built for their workflows, these issues could indeed be addressed. The challenge though is how to enable such configuration of the mobile app regardless of the skills possessed by the user, and also, how the resultant mobile app can be made user-friendly. In this paper, the only skill that we assume from the user is the ability to *perform the workflows (to be mobilized) on the desktop*. Since the user performs the workflows on the desktop anyway, this is a reliable assumption.

The simplicity of the spot tasks allows for the design of such a mobilization solution to be possible. Since the spot tasks have a limited number of UI elements from within only one screen of the application, easy configuration of the apps (and the layouts) can be achieved, without requiring the user to have coding and design skills. Also, the linear non-parametric nature of spot tasks allows for the creation of robust mobile apps. Since the value of ACT and NAV elements are fixed for spot tasks, the sequence of stages in the workflow will always be the same. This eliminates the need for the user to anticipate any branches that may depend on the value of ACT/NAV elements and configure them. Furthermore, even if these tasks are already mobilized under other granularities, the users still might have to experience navigational burden just to perform these simple tasks.

Scope and Goals: The scope of our work is limited to the mobilization of spot tasks within enterprise web applications. We primarily consider HTML/JS compatible web applications due to their dominance [5]. The solution needs to support all major smartphone OSs (Android, iOS, Microsoft). The solution also needs to be usable by all users regardless of their skills.

3 *TASKR*: A DO-IT-YOURSELF APPROACH TO SPOT TASK MOBILIZATION

In this section, we present *Taskr*, a framework that allows for mobilization of spot tasks within enterprise applications by all users. The *Taskr* infrastructure consists of three components - *Taskr-recorder*, *Taskr-server* and *Taskr-client* (Figure 1). The *Taskr-server* is hosted on a cloud platform. When the enterprise wants to allow DIY mobilization for a particular application, it hosts the corresponding application client (for web applications, this would mean a browser pointing to the appropriate URL) on the infrastructure. When a user wants to mobilize her workflows, she uses the *Taskr-recorder* configuration tool to configure the mobile app simply by *performing the workflow that needs to be mobilized*. The infrastructure generates a *Taskr-client* mobile app (.ipa, .apk, and URL) for the user to download and install onto her smartphone. When the user launches the *Taskr-client* app, a computing slice is set-up on the fly to service that specific user session. The slice automatically loads the corresponding desktop application and user configuration. The infrastructure delivers the mobile view as configured to the smartphone. The user interacts with the *Taskr-client* app, and the actions are shipped to the cloud infrastructure where they are performed on the desktop client. In addition to the mobile app, the user can also start the spot tasks by sending a command to the *Taskr* over Email, Twitter, SMS, Slack, etc. The server replies to the user with any configured READ elements and asks the user to send the values of the configured

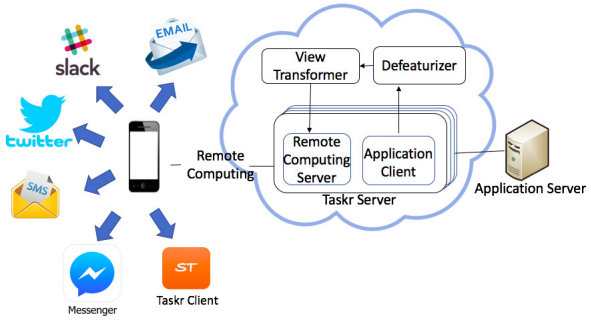


Figure 2: *Taskr* architecture

ACT elements. The user can then reply to this message with the ACT values. We now delve into the key design elements of *Taskr*.

3.1 Key Design Elements

Remote Computing with Refactoring: *Taskr* uses remote computing [8, 17] to mobilize applications while requiring no development and minimal deployment effort from the enterprise or the end-user. To mobilize any given application, enterprises can host a remote computing server and the application client on a Virtual Machine in the cloud. The application client's view is then streamed to the remote computing client on the user's smartphone. The user interacts with the application locally on her smartphone. It is indeed an interesting candidate to solve the mobilization problem. However, the key limitation of remote computing is that the entire application is streamed to the smartphone as-is.

Taskr optimizes the remote view for the client device through *Application Refactoring*, wherein the desktop application UI is dynamically transformed into an appropriate UI for the smartphone. Refactoring restructures the view for the target platform without changing the underlying application behavior via two steps - (i) reducing the number of features available (*Defeaturization*) and (ii) optimizing the application view (*Transformation*). The benefit of this approach stems from the fact that the UI elements of the desktop application can be selectively chosen and transformed into highly optimized versions for usability on the smartphone.

Do-It-Yourself Configuration: Users of an enterprise application best know what features are required to be present in the mobile app, in order to perform their job functions easily. *Taskr* leverages this fact and allows the users to configure defeaturization and transformation within remote computing themselves. For configuration, the users are only required to perform the workflows on the Desktop application in the presence of a configuration tool - *Taskr-recorder*. This tool observes the user's interactions with the application to know what UI elements are necessary for the completion of the task and defeaturizes the application to include only these elements. The tool also allows the users to fine-tune the configuration through an intuitive user interface.

Flexible Mobile Delivery: The result of the configuration process is a mobile app through which the users can view all their spot tasks and execute them. Note that a key goal of *Taskr* is to reduce the task burden of performing the tasks for all users irrespective of their skill levels. Therefore, *Taskr* does not restrict the users to use the mobile app to execute the tasks. Smartphone users use certain apps extensively throughout their day (e.g., Twitter, SMS, Email, Slack,

Messenger, etc.). *Taskr* leverages the users' familiarity with these modalities and allows them to execute their tasks within them. This saves the user the burden of learning to use the interface of a new mobile app - *Taskr* client. *Taskr* transforms the UI of the desktop application to suit these usage modalities i.e. smartphone native UI for the *Taskr-client* app and text blurbs for the other modalities.

Single Screen Transaction: The ideal candidates for DIY mobilization are the workflows that can be performed easily with the limited screen real estate of a smartphone. The workflows should not only require little user interaction but also be simple enough to be configurable by users of all skill ranges. Therefore, in order to maintain usability while at the same time requiring minimal intervention from the user, *Taskr* restricts the users to configure only a limited number of UI elements within one spot task. In this paper, we set the limit to 140 characters each for the total character count of READ elements and the labels of ACT elements[4].

3.2 Challenges and Design Choices

How is the configuration done? The user configures a spot task by simply performing that particular task in the presence of *Taskr-recorder*. For all the stages except the last stage of the task, the tool automatically tracks the UI elements that are acted upon by the user and records the action parameters - ACT elements, their values and NAV elements. For the last stage, the tool has an interface through which users can select any elements that may have been missed and assign a category to them - READ/ACT/NAV. As the user is selecting the elements, the tool records the number of characters of READ elements and the labels of ACT elements. Once the total number of characters in each category reaches the limit defined in Section 3.1, the user is notified and a further selection of elements is disabled.

How are UI elements identified reliably? If the actions performed by the user on the refactored view have to be correctly executed by the server, the UI elements involved in a workflow need to be reliably identified among all the other elements in that application, even when the application changes. Identification of UI element involves extracting a set of parameters (say, the *fingerprint*) unique to that element in the entire application view. Graphical coordinates cannot be used as a fingerprint, as minor changes on the page can easily break the element's fingerprint. Given that web applications are structured as a tree, called the document object model (DOM), the position of the element from an anchor element (nearest ancestor with an HTML attribute *id*) in the DOM can be considered as a fingerprint. However, it is susceptible to failure due to changes along the path from the anchor to the element. Therefore *Taskr*, instead of statically extracting an element's fingerprint, observes an element's features across multiple instances of the application over a period of time to determine what features remain stable and uses only these features as a fingerprint. Specifically, the features tracked by *Taskr* are: (i) Tag name, (ii) All HTML tag attributes, (iii) Path from the root node in the DOM , (iv) Path from the nearest ancestor on the DOM with an *id*, (v) Path from the nearest ancestor on the DOM with more than one children, (vi) Graphical coordinates with respect to the top left corner of the page, (vii) Graphical coordinates from the nearest ancestor with an *id*, and (viii) Graphical coordinates from the nearest ancestor with more than one children. A subset of features that are the most stable (the same in at least 80% of all instances) is then used as the fingerprint.

How is data extraction done? Once the UI elements are identified, extracting the (i) nature of the UI element (e.g., textbox/button etc.) and any (ii) associated context (e.g., label) is crucial so that the user can understand and execute its function on the mobile device as intended. This information cannot always be inferred from the HTML source of the element. This problem is further aggravated by the presence of complex third-party UI frameworks. For example, a button drop-down menu from Bootstrap with source < a class='btn' > would be incorrectly classified as a link (from the 'a' tag). Therefore, *Taskr* uses a hybrid approach that not only obtains data from the source but also from the other surrounding tags, and by taking the user's help where such extraction is not possible. Using tag and attribute definitions from the HTML5 standard and from the complex UI frameworks, a list of rules for extraction is first created manually. For e.g., to get a label for an <input> element, the text within that element's tags is processed. When no text is found, the page source is be parsed to see if a 'label' tag for that input is present. At the configuration step, the extracted nature and context are displayed to the user. Whenever extraction using rules fails, the user is prompted to specify the nature and the context. Note that this is tractable as it only needs to be done once for every new UI element encountered.

Translation to a mobile view: Every UI element in the workflow selected by the user needs to be translated into the desired usage modality on the smartphone - native UI element for the smartphone app client and text for email, twitter, SMS, slack, etc. *Taskr* uses a translation table that maps each UI element (including the ones from the third-party UI frameworks) to a corresponding native UI element (for the app) and also a text version for the other modalities. The result of the translation is presented to the user during configuration. When the translation table does not contain a mapping for the selected UI element or if the result of the translation is not satisfactory, the user can manually specify the translation by selecting a type (e.g., text box, radio button, etc.) and a corresponding label. For every new element encountered this step needs to be done only once[5].

Mobile delivery and presentation: For every workflow stage, the translated versions of these elements have to be displayed on the mobile screen in a manner that enables the user to finish the task with minimal effort. Taking into account the simplicity of spot tasks and the inherent limits on the number of characters allowed in the final stage of the task, *Taskr* follows a fixed display template for every spot task. For the mobile app modality, *Taskr* divides the screen into three panes, and populates the READ elements in the first pane, the translated versions of the ACT elements in the second pane and two buttons 'SUBMIT' and 'CANCEL' in the final pane. The elements are displayed in a list within the respective panes and in the order of their selection during the configuration phase to preserve the logical sequence of actions in the workflow. For the

[4]This restriction is arbitrary and is imposed to allow all transactions to mostly fit within a few text messages

[5]Note that, the current version of the translation table covers most of the input elements from HTML5 standard.

Application	Workflows	Application	Workflows
AWS	1. Create a security group 2. View service status 3.View instance status 4. View account balance 5. Create new volume	Peoplesoft	1. View the latest salary amount 2. Add direct deposit account 3. View year to date earnings 4. Get balance vacation hours 5. Update contact information
Sharepoint	1. Get the next task deadline 2. Create a task and assign it 3. Edit a wiki page 4. Sync the website 5. Share a project	Salesforce	1. Get Quarterly net performance 2. Create a poll for followers 3. Get information on the top deal 4. Create a new campaign 5. Create an open lead

Figure 3: List of Worflows configured on enterprise applications

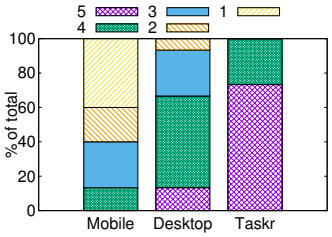


Figure 4: Likert responses from volunteers

other usage modalities, a text blurb is constructed with the text version of the READ elements in the final stage followed by the labels of the ACT elements (one in each line) and sent to the user. To execute the workflow, the user can reply to this blurb with values for the ACT elements (one in each line and in the same order).

4 EVALUATION

Prototype: We implement a proof of concept prototype of *Taskr* with which users can easily mobilize spot tasks and execute them through three different usage modalities - app, Twitter and Email (see Figure 4). Within this prototype, the *Taskr-recorder* is a Javascript browser extension for Google Chrome. The *Taskr-server* is written in python and deployed in the Amazon EC2 cloud. When the user selects a spot task, it instantiates a headless Chrome browser and attaches a Selenium automation driver to it. Upon receiving any user actions performed on the *Taskr-client*, it executes them on the browser through selenium. For the Twitter usage modality, the server uses Twitter Direct Messaging APIs to filter out appropriate commands from its Twitter stream and to send responses to the user. For the Email usage modality, the server monitors its email mailbox for any emails with commands using Python's imaplib. Any response to be sent to the user is handled by smtplib. Finally, the *Taskr-client* is implemented as an app for Android OS.

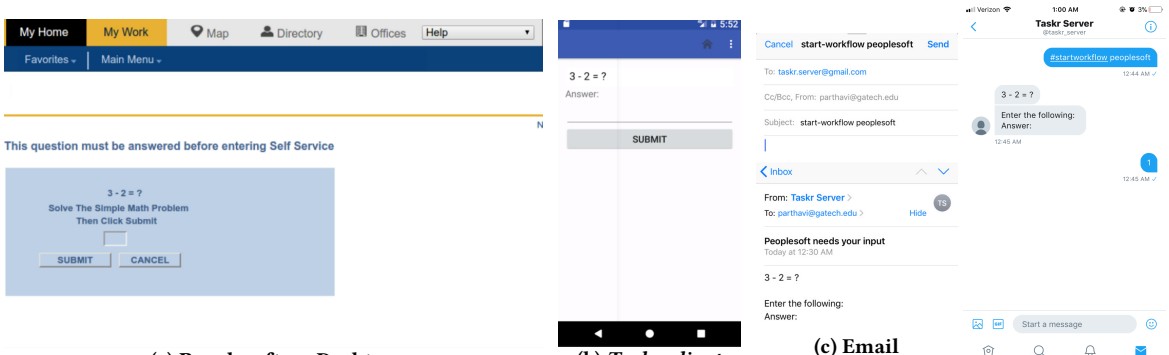
User Study: We mobilize spot tasks in 9 enterprise applications using *Taskr* in the following categories - Learning Management System (Sakai [9]), Human Resources Management (Oracle Peoplesoft), Collaboration (Sharepoint), Customer Relationship Management (Salesforce CRM), Accounting (Quickbooks [6]), Cloud Management (Amazon Web Services), Billing portal (A utility company website - name anonymized), Electronic Health Record (AtlasMD) and Fleet Management (Element Fleet). We configure five workflows from each of these applications representing typical daily usage patterns of employees. For brevity, we only show workflows from four of these applications in Table 3. We then start the *Taskr-client* on a Google Pixel smartphone (Android 7 Nougat) and the *Taskr-server* on an Ubuntu Server hosted on Amazon EC2 cloud instance. We subsequently execute each of the workflows on the *Taskr-client*, Chrome browser on the smartphone, and a Chrome browser on a desktop. Whenever the workflow cannot be performed using the mobile web version of the application, we load the desktop page of the application on the mobile browser to complete the workflow. We observed that, on an average, the workflows on *Taskr-client* take 40.67% fewer actions compared to the desktop browser and 38.19% fewer actions compared to the mobile browser.

We also evaluate *Taskr* using subjective experiments on 15 volunteers[6]. We selected the following 5 workflows from 3 applications - Sakai (editing a wiki page, changing permissions of a site and adding a participant to a site), Amazon AWS (#4), and Peoplesoft (#2). Each volunteer performed the workflows on three platforms (*Taskr* client, Desktop and Mobile browser) in a random order. The volunteers were then asked to answer 7 questions rating each of the platforms. Each question had 5 Likert-type [13] responses from which the user could choose one. Each option has a score corresponding to it (from 1:worst to 5:best). Figure 4 shows the % of total responses across the scores from the users in a stacked graph for one of the questions - *How satisfied are you with the application?* The responses to other questions follow similar trends. The users consistently rated *Taskr-client* better than the other two platforms for all the questions. For example, 100% of the users were satisfied (score > 3) for *Taskr*. On the other hand, only 66.67% of users were satisfied with the desktop experience and 13.33% with the mobile experience. The desktop was rated the better in general than the mobile, due to the user's familiarity with the application on the desktop.

5 RELATED WORK

In [3, 19], the content of a website is rearranged or enlarged to improve the readability on a small screen . [15, 16] allow users to access web applications from mobile phones. However, these solutions are designed for static web applications; [18] is a remote computing solution that reduces the task burden by creating macros for repeated tasks. However, this solution does not rely on smartphone native UI, and rather relies on the Desktop UI as-is and does not defeaturize. [11, 14] create application mashups that allow users to define a smaller subset of UI elements to be visible from a smartphone. However, complex image recognition is required to identify the user intent automatically. PageTailor [10] introduces reusable customization wherein the users can select the components of a web page to be rendered on the smartphone. However, this is only applicable to read content-centric web pages and not complex enterprise applications that require user interactions. [12] deconstructs PC applications to graphical primitives and reconstructs them on the mobile browser. However, it retains all features of the original

[6]The volunteers were mostly university students within 22-30 year age group



(a) Peoplesoft on Desktop (b) *Taskr-client* (c) Email (d) Twitter

Figure 5: *Taskr* prototype for a test workflow on Oracle Peoplesoft

application, thereby reducing the usability when rendered on a small screen.

6 DISCUSSION

Security: Most enterprise applications require the user to log in (either explicitly or through a single sign-on service) before any workflow can be executed. The requirement of log in usually does not restrict the number of workflows that qualify as spot tasks as the username and password can be treated as fixed parameters. The login username and password required are required by *Taskr* to execute workflows on most enterprise applications. The login parameters constitute sensitive information and can be encrypted and stored on the local device using services like keychain API for iOS. When the spot task has to be executed, these parameters can be encrypted and sent to the server using transport security such as SSL. Alternately, this sensitive data can be stored in the cloud isolated within enterprise network and hence be protected by enterprise firewalls. The user can then be restricted to using *Taskr* within the enterprise network. If the application server allows it, a continuous login session can be maintained at the *Taskr-server* using the stored username and password.

Evaluation: *Taskr* requires accurate fingerprinting of UI elements to execute the workflow. While we discuss the fingerprint technique used by *Taskr* in Section 3 and implement it in the prototype, we do not evaluate it for correctness. We plan to investigate this in the future. We implemented *Taskr-client* and server for twitter, email and native app usage modalities. However, we only conduct subjective tests on the native mobile app modality. We plan to implement a few other modalities and extend the testing in the future.

Extraction rules and Translation tables: *Taskr* relies on manually constructed rules for information extraction and fixed translation tables. For the prototype, we constructed these rules for most elements defined by the HTML5 standard. However, many web applications use elements defined by third party UI frameworks. We plan to extend these rules for some popular UI frameworks used by web applications.

Extension to other workflows: *Taskr* helps users mobilize simple workflows that can be described as spot tasks. This restriction limits the number of workflows that can be mobilized. We plan to relax these restrictions to include workflows that can be described as a sequence of spot tasks, and also other general workflows in the future.

7 CONCLUSION

In this paper, we identify a new granularity of mobilization - spot tasks, and argue that it empowers the users to drive the mobilization efforts themselves. We present *Taskr* a do-it-yourself mobilization infrastructure and implement a prototype through which users can mobilize spot tasks and execute them through a mobile app, Twitter or Email. We then evaluate it with users and show its benefits.

REFERENCES

[1] 2017 trends in enterprise mobility. https://goo.gl/3M2Ruv.
[2] Employees say smartphones boost productivity by 34 percent. https://goo.gl/PmEUys.
[3] Feed circuit. http://feedcircuit.garage.maemo.org/.
[4] Gartner survey shows that mobile device adoption in the workplace is not yet mature. https://www.gartner.com/newsroom/id/3528217.
[5] Google trends on web platforms. https://goo.gl/qqy558.
[6] Intuit quickbooks. https://quickbooks.intuit.com/.
[7] Mobile workforce to drive further enterprise change in 2017. https://goo.gl/uWHqGm.
[8] Remote desktop protocol. http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx.
[9] Sakai. https://sakaiproject.org/.
[10] N. Bila, T. Ronda, I. Mohomed, K. N. Truong, and E. de Lara. Pagetailor: Reusable end-user customization for the mobile web. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, New York, NY, USA, 2007.
[11] F. Lamberti and A. Sanna. Extensible guis for remote application control on mobile devices. *IEEE Computer Graphics and Applications*, 28(4):50–57, July 2008.
[12] H. Li, P. Li, S. Guo, X. Liao, and H. Jin. Modeap: Moving desktop application to mobile cloud service. *Mobile Networks and Applications*, 19(4):563–571, 2014.
[13] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.
[14] I. Mohomed. Enabling mobile application mashups with merlion. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10, New York, NY, USA, 2010.
[15] A. Moshchuk, S. D. Gribble, and H. M. Levy. Flashproxy: Transparently enabling rich web content via remote execution. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 81–93, New York, NY, USA, 2008.
[16] J. Nichols and T. Lau. Mobilization by demonstration: Using traces to re-author existing web sites. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, pages 149–158, New York, NY, USA, 2008. ACM.
[17] T. Richardson and J. Levine. The remote framebuffer protocol. 2011.
[18] C.-L. Tsao, S. Kakumanu, and R. Sivakumar. Smartvnc: An effective remote computing solution for smartphones. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, pages 13–24, New York, NY, USA, 2011.
[19] D. Zhang. Web content adaptation for mobile handheld devices. *Commun. ACM*, 50(2):75–79, Feb. 2007.