A Plug-in Architecture for Connecting to New Data Sources on Mobile Devices

Kerry Shih-Ping Chang, Brad A. Myers
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{ kerrychang, bam }@cs.cmu.edu

Gene M. Cahill, Soumya Simanta, Edwin Morris, Grace Lewis
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{ gmcahill, ssimanta, ejm, glewis }@sei.cmu.edu

***Abstract*—A key use for mobile devices is to search and view online information while on the go. As a result, many mobile applications serve as front ends for online databases. While there are many thousands of data sources that provide web service APIs giving access to their databases, creating mobile applications to use those sources requires significant mobile programming knowledge and a significant amount of time. We introduce Spinel, a plug-in architecture for Android, and a set of web-based configuration tools that together enable users to connect mobile applications to new data sources without programming. Spinel also provides APIs that make it easy for developers to create new applications that use those data sources. We provide three demonstration Android applications that use such data: Listpad for entering personal lists, Listviewer for viewing results of data queries, and Mapviewer for displaying query results on a map. An informal usability study showed that users could successfully attach new data sources to those applications.**

Keywords—end-user programming; mobile devices; plug-ins; web APIs; mashups

I. INTRODUCTION

A key use for mobile devices is to search and view information [1]. The need to access information while on the go has directly impacted the mobile application market, as many mobile applications are front-end user interfaces for online databases. The need for information in general has also led companies to make their databases publicly available through web application programming interfaces (APIs) to allow third-party developers to use the data in customized ways. Examples include websites that aggregate data from multiple sources and present them in an integrated interface, sometimes known as "mashups." According to ProgrammableWeb.com, today there are over 8800 web APIs available.

Despite the desirability of using these data sources, the majority of web API users must be tech-savvy developers who are experts in web programming [2]. It is even more difficult if one wants to use web APIs in a native mobile application. There are many advantages to running a native application instead of just a web-based application on mobile devices, such as a better look and feel of the user interface and faster performance. However, creating a mobile application is generally more difficult and time consuming than creating a web application. Consequently, there are many fewer data sources available on mobile devices than on the web. Even if there were applications pre-made for each user's needs, the user would then be faced with having to deal with each application's different user interface and multiple, separate data displays. The research on end-user programming suggests that there can never be sufficient software applications to satisfy every individual's needs, unless end users are given the ability to customize the programs [3].

Another motivation for providing dynamic connection to new data sources from mobile devices comes from emergency and first responders, who need to be able to quickly and easily view and combine information from new data sources relevant to the immediate issues. For example, the Homeland Security Infrastructure Program provides approximately 450 geo-coded data sets for use by the homeland security and emergency response communities, and the Homeland-Defense Operational Planning System from Lawrence Livermore National Laboratory provides online access to 500 other sources [4].

We present Spinel,[1] a plug-in architecture for Android that allows users to connect mobile applications to new data sources dynamically without programming. We have observed that while there are thousands of data sources available, most mobile applications have similar user interfaces with which they display such data, for example, using lists, maps (for geo-coded location data), and grids (for images). Therefore, a single, consistent user interface design can be used to display many different kinds of data. For example, the same list interface used to display contact information can also be used to show movie data, and a generic map-based display program can support a wide variety of geo-coded data sets.

This motivated the design of Spinel, which allows developers to write applications that access data sources using plug-ins. The Spinel architecture provides two levels of connectivity for data sources. Most data sources can be utilized by plug-ins comprised of one or more files in JavaScript Object Notation (JSON)[2] format, which are lightweight, sharable, and easily created in any text editor for people who are familiar with the syntax. Spinel also provides an interactive web-based configuration tool to create these plug-ins without writing any code. These plug-ins can then be

[1] Spinel is a gemstone, and here stands for Smartphones Providing Information Needing External Lists.
[2] http://www.json.org/

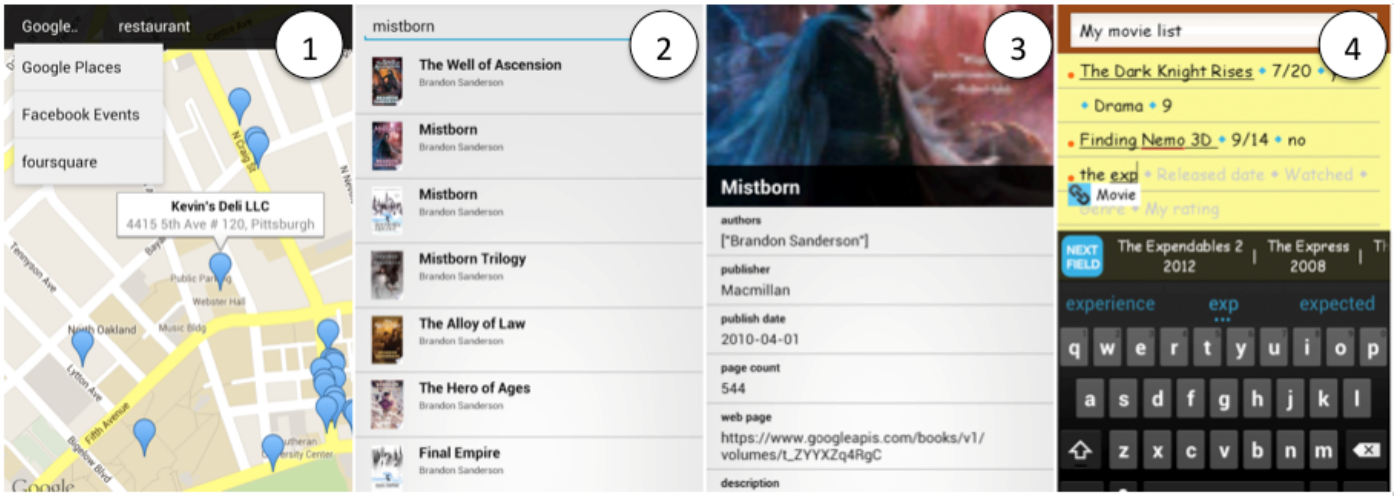

Fig. 1. Examples from the three Spinel mobile applications running on an Android phone: (1) Mapviewer showing a popup menu with three data sources loaded, (2) Listviewer's list view and (3) its details view, and (4) Listpad for entering personal lists. Listpad lets users type their own list information but uses external data sources to provide auto-complete suggestions, such as the movies shown to the right of the blue "Next Field" button.

dynamically added to a Spinel mobile application. In this way, users can easily extend the application by adding new data sources based on their needs. For data sources whose APIs cannot be supported by the JSON file, the Spinel architecture provides the ability for developers to write Java code to create a reusable plug-in, which can then can be used to connect to this web data source from multiple mobile applications. However, so far, we have not found this capability to be necessary – the JSON file has been sufficient for all the data sources described in this paper.

The contributions of this work include the plug-in architecture for mobile applications and a web-based interactive tool so that users can extend existing applications by adding new data sources without programming. This architecture also allows developers to easily create new applications that can make use of these data sources through an Android library. We evaluated Spinel in two ways: we built three demonstration applications (Listpad, Listviewer, and Mapviewer) to validate the architecture and connected them to more than a dozen different data sources; and we conducted a small usability evaluation to see if the web-based plug-in configuration tool was able to be used by developers.

In the rest of the paper, we first describe two usage scenarios and then present related work. After explaining Spinel in detail, we describe the usability evaluation and discuss our findings. We conclude the paper with future work.

II. Scenarios

A. End-User Programmer

Tim is an end-user programmer who wants to keep a list of songs he likes on his smartphone. Tim already uses Listpad, a general-purpose application built using the Spinel architecture to enter his other lists on the phone, such as a list of movies (see Figure 1 at 4). Listpad lets users enter lists of items and uses external data sources to autocomplete entries to make it easier for users to enter the information [5]. Here is how Tim could extend the Listpad to autocomplete music information by creating a Spinel data source plug-in for music.

Tim goes online to Spinel's web-based plug-in configuration tool (see Figures 4, 5, and 6). From a list of Spinel applications, he chooses to create a plug-in for Listpad. The configuration tool pops up a small dialog displaying instructions written by Listpad's developers (Figure 5), explaining that in order to add a new data source for autocomplete suggestions, Listpad needs two APIs, one that searches the data source and the other that retrieves the full record details of a selected item. The configuration tool prompts Tim to add these two APIs and asks him to start by providing examples of their usage. Tim searches "music web API" on Google, and in the top five search results he finds that Last.fm provides many APIs to access its music database. Tim quickly signs up on Last.fm to use its APIs, which gives him the needed "API key" to authenticate his data queries. From the documentation, Tim finds examples of the Album.search API and the Album.getinfo API, which are the two APIs he wants. Tim adds these APIs by first pasting an example API request into the tool. Following the instructions from Listpad developers shown in the tool, he then enters his new API key, configures the API, and selects the desired fields from the return data as the outputs to use in Listpad. When he finishes, the configuration tool provides a link for him to download the plug-in he just created. Tim downloads the plug-in, which is a single JSON file, and puts it in the Listpad folder on his phone. Tim starts Listpad and he can now choose Last.fm, the source he just added, to be the source of autocomplete suggestions, and Tim can quickly create a list of songs.

B. Developer

Ann is a developer who wants to make a new application for displaying data on a map (for example, see Figure 1 at 1, which was created in this way). While she originally wants her app to show data from Google Places, she also wants users to be able to extend the application to display data from any other sources that have a location property. So she builds her

application using the Spinel Android Library. The Spinel Library (Figure 3) provides APIs that let Ann query the data source and use the returned results. Ann uses the web-based developer tool to describe the desired kinds of APIs and the names of the fields that her code will use for the properties. For example, her app will use a search API that takes a string and returns a list of items, each of which should have a field called "location," which is a geo-coded location. She then uses conventional Android programming tools to create her application. Spinel's library makes it easy to access the data through the plug-ins from any remote data source. To test her app, she uses the web-based configuration tool (that Tim used) to create a plugin for Google Places, which will be shipped as the default plugin with her app.

III. Related Work

A. Mashups

Wong and Hong's survey [6] discusses several popular categories of mashups, including not only applications that combine data from multiple sites but also applications that provide new ways to interact with the data or present the data in a personalized way. Data in a mashup could be retrieved using web APIs, by scraping web pages, or even from local Excel spreadsheets. Tools like Karma [7] and Vegemite [8] allow users to extract and combine data from existing web pages or spreadsheets by demonstration. Listpad is different from these tools as it provides an architecture for using web APIs on mobile devices.

Studies have found that correctly using web APIs is one of the most difficult challenges in creating mashups [2], and many research projects have focused on allowing end users to interact with web services without programming. Marmite [9] provides a graphical user interface with a set of widgets that lets users extract and filter data on the web. Users interact with web services using forms that allow them to configure the inputs and outputs of an API. Users can select multiple widgets, execute them in order, and view the output. d.mix [10] facilitates the use of web APIs by integrating services and call sites. It allows users to inspect elements in a pre-annotated web page and sample the API calls used on those web pages in their own mashups. However, a big drawback of both Marmite and d.mix is that extending the system to support a new web API requires significant effort from the developers – Marmite requires developers to program a new widget that connects to the data service, and d.mix requires developers to write new annotations. This seems unrealistic given the large number of APIs available today. In contrast, applications using the Spinel architecture do not need to be changed when new data sources are added. Spinel developers write the application once and users can add new data sources based on their needs. This idea of letting users extend an application by providing new data sources was also discussed in Reform [11], in which developers write site-independent web enhancements and then lets end users attach them to existing web pages. Spinel is fundamentally different from Reform, however, as Spinel focuses on allowing end users to integrate web APIs into mobile applications.

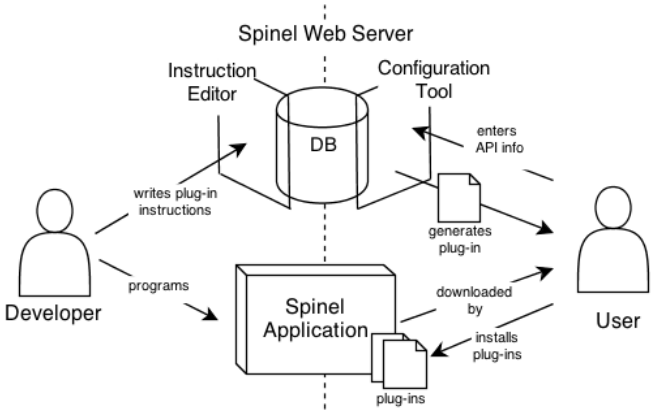

Fig. 2. Spinel's architecture and modules

As for commercial tools, Yahoo Pipes [12] is a graphical programming interface that provides a comprehensive set of widgets for creating mashups, such as fetching RSS feeds and manipulating strings. IBM's Many Eyes [13] is a visualization tool that lets users interact with the data they uploaded in new visualizations. Neither helps users attach data to mobile applications or provides an architecture for developers to create extensible applications.

B. End-User Programming for Mobile Applications

Microsoft's TouchDevelop [14] is a programming environment designed to let people write mobile applications directly on their mobile devices. It uses a custom scripting language and allows users to enter code by tapping on icons that represent different functions. The programs created in TouchDevelop can access certain data, but users of TouchDevelop still need to write code. Several systems help people create data collection applications on mobile devices without programming, such as Open Data Kit [15] and Sensr [16]. They provide a web-based graphical interface builder that lets people create data entry interfaces by drag-and-drop. However, these are quite different from Spinel because they do not support connecting to multiple data sources.

IV. Spinel

Figure 2 shows the various modules that are used in the Spinel architecture. From the developer's point of view (Ann in the second scenario), the application is programmed using the normal Android tools, but also with the Spinel Android Library to connect to data sources. When the application is finished, it is published as usual to the Android marketplace. Another requirement is to use the web-based developer tool on Spinel's website to write a short description of what APIs are used in the application and how they are used. Figure 2 shows this as the Instruction Editor. The description is stored in the database (DB) on the Spinel web server. From the user's point of view, applications can be downloaded and used as usual, if the built-in data sources are sufficient. To add a new data source (as Tim does in the first scenario), the user goes to Spinel's website and uses the Configuration Tool to create a new plug-in that meets the requirements expressed previously by the developer using the Instruction Editor. With the configuration tool, the user enters the information about the

data source and selects the desired fields from the returned data as outputs based on the developer's instructions, as described in the scenario.

A. Spinel Plug-in

Creating the more complex form of the Spinel plug-ins requires programming in Java to Spinel's API and is therefore not very interesting. The rest of this section will focus instead on the plug-ins that can be described by one or more files in JSON format. The first part of the JSON file describes the data sources that provide the APIs, including their names and the authentication methods they use. Although there are web services that do not require any user identity, most data sources require people to register the application first and provide keys that grant access to the web APIs. Spinel supports the two most common authentication methods used in web APIs: an API key and OAuth 2.0. For API keys, the user gets a single key from the data source, which can then be used for all future queries (as shown in the scenario). OAuth [17] is an authentication protocol that lets end users log in to the mobile device and grant permission to the applications to access remote data sources on their behalf, for example, to read personal data such as friend lists from Facebook. OAuth has become a standard in web services and is used by major websites such as Facebook and Twitter. The JSON file stores the parameters required by these two methods: for API keys, the file contains the value of the key; for OAuth 2.0, the file stores the required parameters such as the client ID, the secret, and the authentication URLs. All this information should be documented in the API's official web page.

The second part of the JSON file describes the web APIs used in the application and their parameters. For each API, the file includes the name of the API, the request URL, and the output format. The JSON file plug-in supports REST APIs. REST is the predominant architecture style used by most web APIs [18]. Following the REST design guidelines, an API request in the plug-in is divided into a base URL and a set of parameters. Parts of the API request that are affected by user inputs are represented by special keywords and will later be replaced with the real values in the application.

The JSON file also describes which fields in the data returned by this API call are used by the application, and what they should be called. Currently, the JSON file plug-ins support only APIs that return data in JSON format. Each desired field is represented by a name given by the plug-in developer (so the application can access it) and a path that gets to the correct node in the returned JSON model. Spinel uses JSONPath [19], a language similar to Xpath, to extract data out of the returned JSON data. In the future, we will make Spinel support data returned by APIs in XML format as well.

JSON file plug-ins are usually created by the web-based configuration tool. However, people who are familiar with Spinel and JSON can choose to write a plug-in using any text editor. This may be useful for some data sources. As discussed below, the web-based tool can describe outputs with specific names or at a specific index in a list, but the JSONPath language is somewhat more powerful and can express multi-level nested indexing, such as the second item in the third item. We provide empty plug-in templates for people to fill in, or they can specify most of the format using the web-based tool and fix the indexing by hand using a text editor.

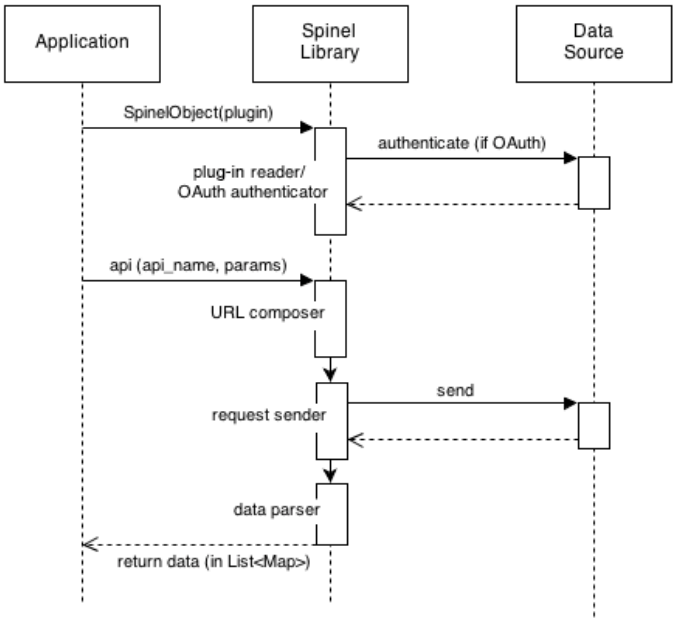

Fig. 3. UML sequence diagram showing a use of the Spinel Android Library. Time goes down from the top.

B. Spinel Android Library and the Instruction Editor

As discussed in the scenario for Ann, Spinel provides a Java library for developers to use in applications. To make the development easier, this library handles the entire process of making a web API call, including establishing the internet connection, handling the appropriate authentication protocols, sending the request, and parsing the returned data, based on the information specified in the plug-in.

Figure 3 shows a UML sequence diagram of the process that an application uses to interact with the remote data source. We use a new scenario here to describe how the library works. Suppose the developer is writing a common search-and-view application that searches the remote database based on the user's inputs and presents the results in a list. The user can click on any item of the list to view the more detailed information about the item (see Figure 1 at 2 and 3 for examples). The developer first creates a SpinelObject for each plug-in file. SpinelObject initializes all variables based on information in the plug-in. If any data source in the plug-in uses OAuth as the authentication method, the OAuth authenticator will be invoked. It lets users login to the source in a popup window and retrieves an access token that is later used to make API calls. The developer needs to write only a single line of code (calling the constructor) to perform all of these initialization steps.

When the user types in a value and then presses the search key in the interface, an API is called to search the remote database. To do so, the developer puts the query string in a Java hashmap and gives it the appropriate name, for example, "query-term." Then this is passed to a method in SpinelObject with the name of the API, for example, "search." The method

will use the information for the "search" API specified in the JSON plug-in file and will compose a complete request URL by adding the value that the user specified for the "query-term." Spinel then connects to the remote data source using the URL to retrieve the data from the source. Finally, the returned data is sent to a parser that extracts the desired fields based on the output model specified in the plug-in. Each value extracted is given a name specified in the plug-in and put into a hashmap data structure for the application.

In our example here, the data returned from the search API will be a list of items, so a simple loop is used to take the value of the "title" field as the title of the list item in the interface, the value of the "description" field as the subtitle, and the value of the "image" field as the source of the image for each list item. Later, if the user taps on an item in the displayed list, a similar process is used to call the API that returns the detailed information.

We can see from these examples that the design of the Spinel architecture requires consistent naming of the APIs and the input and output data between the application and the plug-in so they can communicate properly. To do so, the web-based instruction editor allows developers to enter these requirements and give some general descriptions of the APIs to help end users create the plug-in correctly. The developer's instructions are integrated into the web-based plug-in configuration tool for end users through the Spinel web server's database.

C. *The Plug-in Configuration Tool*

The final piece of the Spinel architecture is the web-based configuration tool, which allows developers and end-user programmers to quickly create the JSON plug-in files. We designed this tool with several goals in mind. First, because web APIs are URLs that are usually long and hard to read, especially for people not familiar with the format, we wanted to minimize the requirement of manually entering the API information to avoid likely typos and confusion. Second, we wanted to make selecting and naming the input and output fields of an API clear and easy to do in the tool, since that is a critical requirement for a plug-in to work in an application, as discussed in the previous section.

To address the first goal, we designed the configuration tool to let users start adding an API by copying and pasting an example API request from the documentation. The documentation for virtually all of the APIs that we have investigated contains multiple examples, so users will often be able to find such examples easily. Based on the example, the system can prefill many of the fields that users otherwise would have to fill in manually, such as the base URL and the parameters (see Figure 6 at 1). To address the second goal, we carefully integrate the developer's instructions into the tool (see Figure 5 and Figure 6 at 2) and let people specify the output fields directly from the return data using checkboxes (Figure 6 at 4).

Figure 4 is a state diagram showing the flow when using the web-based configuration tool. The first page of the tool asks the user to select which Spinel application she wants to create a plug-in for. This is required since different

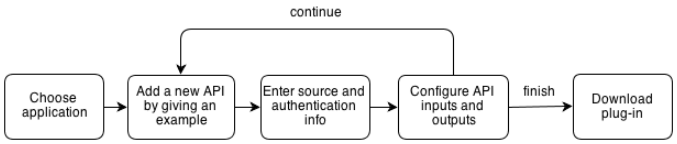

Fig. 4. A state diagram showing the interaction flow of the configuration tool

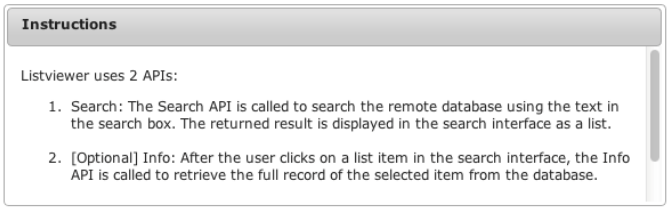

Fig. 5. Developer's instructions about the APIs that the plug-in uses, shown in a small, draggable dialog box

applications have different requirements for their plug-ins. The tool then pulls out the developer's instructions for the selected application from the server and shows them in a small dialog box (Figure 5) as a reference for the user. On the second page, the tool asks the user to start adding an API by giving an example API request. Although giving the system an example first is strongly recommended, it is not a requirement. The user can still proceed without providing an example. Next, the tool asks users to provide the authentication information. As described in the first scenario, all of the information should be documented in the API's official web page, and the user should be able to just copy and paste it.

After the user enters an example API request and the authentication information, the tool is now ready to let the user configure what she wants as inputs and outputs. Figure 6 shows this page, which is the main editing interface. In the configuration tool, an API is divided into a base URL and a set of parameters to increase readability. The fields will be prefilled if the user had given an example, or the user can manually type them in. For example, in Figure 6, Spinel has identified three parameters in the example query that the user pasted in. The Spinel configuration tool has guessed the first parameter to be the API key based on its name and therefore has replaced the value with the API key that the user entered in the previous step. Spinel has identified two other parameters in the example query: "q" and "page_limit," which in this API means the query string and the number of returned items. In this case, the user wants to allow the query string to be entered in the mobile application, so the "q" parameter is marked as the "input" that will be entered. All the other parameters will retain their constant values as entered here (such as the number of returned items). If the API has additional parameters that were not in the example API request, the user can add them manually by pressing the "Add a new parameter" button and then typing the parameter name and value.

In rare cases, we have seen APIs in which the value to be used from the application (like the query string) is not provided as a parameter but instead is part of the base URL. Spinel supports this. The user can select that text in the base URL and press the "Make the selected text …" button below

the base URL box (see Figure 6 at 1 and 2) to make the value of the selected text be filled in at run time.

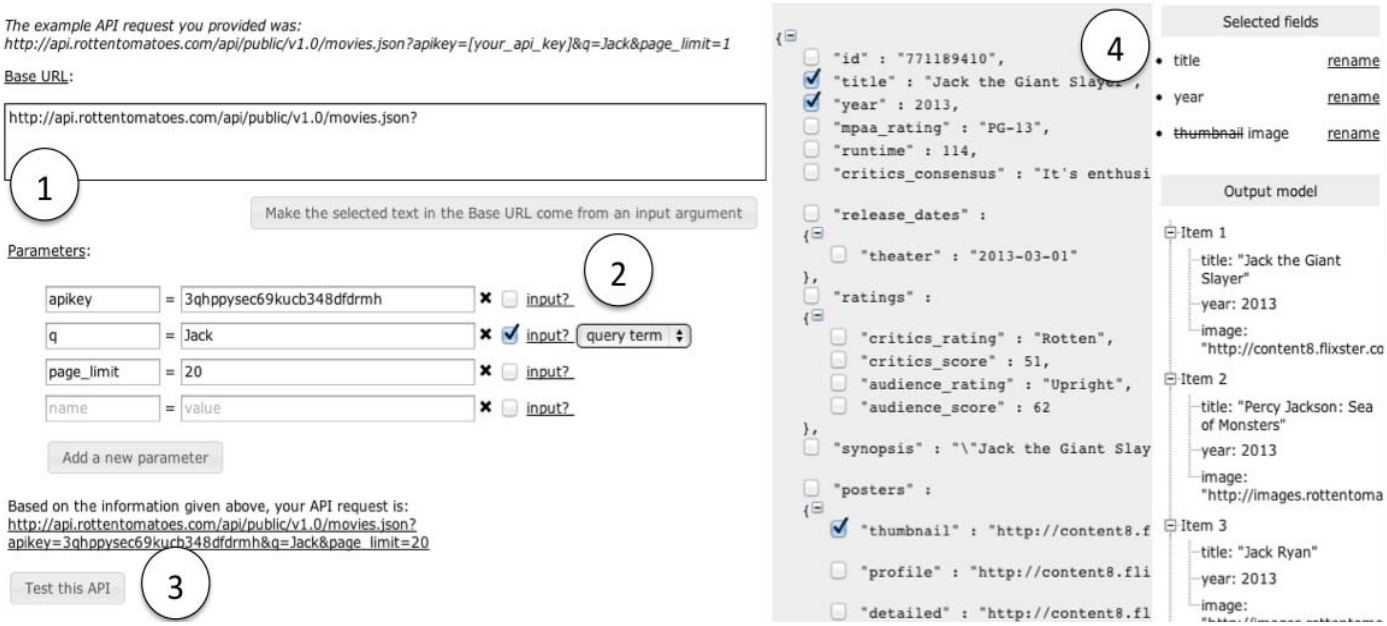

Fig. 6. The main API editing page in the Spinel web-based configuration tool. (1) An API is divided into a base URL and a set of parameters, which are filled in by the tool if the user provided an example API request on the previous page (not shown). (2) The user selects a parameter to be an input field and gives it an appropriate name using the drop-down menu that is generated based on the developer's instructions. (3) The user can click on the "Test this API" button to see the return data at (4, with the gray background). The user can select desired fields as the final output of this API call using the checkboxes. The tool shows all selected field names and visualizes the output model that will be sent to the application in a tree map at the far right, to help the user confirm that her selections are correct.

After entering all of the values, the user can execute the API to see the data it returns using the "Test this API" button (Figure 6 at 3). The return data is shown in the right half of the screen (Figure 6 at 4, with the gray background). For each field in the return data, there is a checkbox to let the user select it as an output. Every time that a checkbox is checked or unchecked, both the Selected Fields panel and the Output Model panel (Figure 6 at 4, on the far right) will update to provide feedback. The Selected Fields panel shows the names of the fields that the user has selected and lets the user rename them to be what the developer specified. For example, in Figure 6 at 4, the user selects a "thumbnail" field from the return data and renames it to "image," as this is what the developer has specified that this application is expecting. The Output Model panel (bottom right of Figure 6) provides a clean view of what the user has selected to be sent to the application. It visualizes the user's selection as a tree structure for better readability. For example, in Figure 6 the user selects the "title," "year," and "thumbnail" (renamed to "image") fields for all items returned as the search results. While in the return data panel, the user cannot really see all her selections because it is quite long and contains much unwanted data. Using the Output Model panel she can quickly check whether her selections are correct.

After the user selects all the output fields that she needs, she can press the finish button to complete editing this API. If the application requires more than one API to be added, the user can follow the same procedure to add another API. When the user finishes adding all the APIs, the configuration tool will generate a JSON plug-in file and let the user download it to the smartphone through a link. The user can then put the file in the appropriate place on her mobile device and use this new data source in the application.

D. Limitations

The configuration tool has several limitations. As mentioned in Section A, the Spinel JSON plug-in uses JSONPath [19] to describe which nodes in the returned JSON model should be extracted as output fields. Currently, the configuration tool supports generating only JSONPath expressions that have the wildcard character (*) in the top-level array. This means that if the API returns a list of items, such as the search results, the configuration tool can handle the items returned, as shown in Figure 6. However, if one of the items itself contains a list, the configuration tool will handle selecting only a particular item out of the second-level list (i.e., the first item of the second-level list, and not the entire list). It also does not support generating any expressions that do filtering (for example, to select all elements with a certain prefix), recursive expressions (such as to select all array elements that have an even index), or relative relations (such as to select the last field of each of the elements in the array). This is mainly because the configuration tool generates the JSONPath expressions purely based on which checkboxes are selected. However, the Spinel libraries are sufficiently general to handle these kinds of queries, so users who know how could express complicated queries like these by hand-editing the generated JSON files in a text editor.

Currently, the Spinel architecture supports only REST Web APIs that return data in JSON format. It does not support

APIs using other protocols, such as SOAP APIs. It also does not support APIs that return other kinds of data formats, though we plan to make it support XML in the future.

V. Example Applications

In order to demonstrate that the Spinel architecture can be used for multiple purposes, we created three different Android applications. The first, Listpad, was described briefly in Tim's scenario and has been extensively described elsewhere [5].

The other two applications were created specifically to test Spinel. The first, Listviewer, displays the results in a list format and allows users to tap on any item for more information. The user who creates the plugin using the Spinel web-based tool can select any fields to be displayed in the search results list, and any fields to be displayed in the detailed view (see Figure 1 at 2 and 3).

The second new application is Mapviewer, described in Ann's scenario, which displays the results of the original query as pins on a map. If the user taps on any pin, then a small popup appears with further information about that item (see Figure 1 at 1). Tapping on the popup brings up a different screen showing a detailed view similar to the one in Listviewer (like Figure 1 at 3). As with Listviewer, the user who creates the plug-in has complete control over what information is displayed on top of the pins and what information is shown in the detailed view.

VI. Usability Study

A. Participants, Tasks, and Procedures

We conducted a preliminary usability study to evaluate the design of our web-based configuration tool. Five participants were asked to use the tool to attach new data sources to two of the example applications. All of our participants were graduate students with programming experience, though none of them were familiar with using web APIs. In real life, end-user programmers vary in programming experience. Our study was designed based on the scenario discussed in [20], where often the more experienced users would create customizations (in our case, the plugins using the web configuration tool), and people with little or no programming skills would find them on the server and use them. Our study did not measure if end users at all programming levels could successfully use our tool. However, it did evaluate the usability of our tool and show whether it is *possible* for new data sources to be attached to mobile applications without programming, which is a key contribution of Spinel.

Before the participants started any tasks, the experimenter first introduced Listviewer and Mapviewer and then gave a short tutorial of the configuration tool by demonstrating how to add a new source to Listviewer. Participants were then given two tasks. For the first ask, they were asked to add a movie database (Rotten Tomatoes) to Listviewer, which lets users search a remote database and displays results as a list, but when the user clicks on a list item, instead of going to the detailed view, the application opens the web page of the selected item. The plug-in for this application needs only a search API, where one of the selected fields must be a URL for the item. For the second task, participants were asked to add a place database (Google Places) to Mapviewer. Mapviewer takes two APIs; one is a search API, and the other is a detail API used for the detailed view to retrieve the full information about an object given its unique identifier (see Table 1).

TABLE I. The APIs used in each task and their input and output parameters

Task	API		
	Name	***Input***	***Output***
Task 1	"search"	"query"	"title," "description," "image," "webpage"
Task 2	"search"	"query"	"title," "description," "latitude," "longitude," "id"
	"detail"	"id"	"title" and any other fields that the participant wants

In both tasks, the participants were given the documentation pages for the APIs that they were using. Participants were also given the authentication information used for the data sources, so they did not need to register with the data sources just for this study.

Since there was no obvious comparison system to Spinel, we felt the most appropriate evaluation would be for usability, so we used a conventional usability evaluation protocol, in which the participants were asked to think aloud during the whole study. They were paid for their time.

B. Results

All of our participants successfully finished the tasks. The average time they spent on the first task was seven minutes, and the average time they spent on the second task was fifteen minutes (which had twice the number of APIs to configure). The real-life times are likely to be even shorter since our participants were doing think-alouds. We identify the time to completion as another big advantage of using the Spinel architecture to add new data sources in addition to the fact that it does not require users to write code. Participants were excited to see the plug-ins they had spent a few minutes creating working in the application. One participant said that it would take him many hours to figure out how to write these in Java since he had never programmed in Android before.

In general, participants found that the configuration tool was easy to use and the flow was easy to follow. Participants did express concerns that in real life they might not know where to look for the right APIs to use before starting to use the tool. In fact, this might be even more of a barrier for people with no programming experience at all. They might even have trouble reading the documentation that comes with APIs, since the documentation is generally targeted at web programmers. Participants suggested that a short tutorial video should be shipped with the Spinel applications to explain some background about web APIs, what they look like, and how to look for them in general, in case the user is not aware of the many existing web APIs.

One of the participants spent a significantly longer time on the first task compared with others, because he copied the

wrong API example to use. He discovered this when the output of the API looked wrong. He tried to fix it by directly changing the text in the base URL and parameter textboxes, but he finally gave up this approach because he did not want to read the whole documentation. Eventually, he went back to the previous page of the configuration tool to start over by giving a new example. This suggests that our example-driven approach is really helpful for people who do not know web APIs well, but it also suggests that with our design it might be hard to recover from a bad example. The participant suggested providing "examples of example API requests" in the tool to help users identify the right kinds of examples to copy from the documentation. We have now added this to the tool.

Participants suggested other usability issues to improve in the tool, including making any value that is a URL link in the return data clickable and using color coding to make the structure of the JSON data more obvious. We will make these changes to the next version of the configuration tool.

VII. Conclusions and Future Work

We have presented the design of Spinel, a plug-in architecture that allows new data sources to be easily added to mobile applications. The Spinel plug-ins are lightweight JSON files that describe the usage of web APIs and can be quickly generated by the Spinel plug-in configuration tool without programming. Spinel comes with an Android library that reads the plug-ins and allows developers to use them easily in the code of their mobile applications.

Future work could be in many directions. First, we plan to extend the Spinel Android library to accept more return formats from remote data sources, including XML and CSV. Second, we could allow plug-ins to be downloaded and installed directly from the web server instead of requiring users to manually move the file to the phone. Third, we could improve the error handling in the library so that it returns useful error messages to explain why a plug-in is not working for both developers and end users as a means of debugging. Fourth, we could extend the Spinel architecture to make it available for applications on other platforms besides mobile, such as for web applications, since the plug-ins are JSON files that can be easily interpreted across platforms. Last, we could create more applications that demonstrate the usefulness of the Spinel architecture, such as applications that provide customizable visualizations or data merging. Our final goal is to make the Spinel web server a community shared space that hosts many different kinds of Spinel applications created by developers, along with many different Spinel data source plug-ins created by developers and end-user programmers, for people around the world to download and use based on their needs.

Acknowledgements

This material is based upon work funded and supported in part by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0000249. This work was also funding in part by NSF grant IIS-1116724. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of CMU, the SEI, or the NSF.

References

[1] K. Church and B. Smyth, "Understanding the intent behind mobile information needs," *Proceedings of the 14th International Conference on Intelligent User Interfaces*, pp. 247–256, 2009.

[2] N. Zang, M. B. Rosson, and V. Nasser, "Mashups: who? what? why?," in *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, 2008, pp. 3171–3176.

[3] B. A. Myers, A. J. Ko, and M. M. Burnett, "Invited research overview: end-user programming," in *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 2006, pp. 75–80.

[4] S. P. Wampler, "Hops Assists Emergency Responders," *LLNL*, 2012. [Online]. Available: https://www.llnl.gov/news/newsreleases/2012/Oct/NR-12-10-12.html.

[5] K. S.-P. Chang, B. A. Myers, G. M. Cahill, S. Simanta, E. Morris, and G. Lewis, "Improving structured data entry on mobile devices," in *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, 2013, to appear.

[6] J. Wong and J. Hong, "What do we 'mashup' when we make mashups?," in *Proceedings of the 4th International Workshop on End-User Software Engineering*, 2008, pp. 35–39.

[7] R. Tuchinda, P. Szekely, and C. A. Knoblock, "Building mashups by example," in *Proceedings of the 13th International Conference on Intelligent User Interfaces*, 2008, pp. 139–148.

[8] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau, "End-user programming of mashups with vegemite," in *Proceedings of the 14th International Conference on Intelligent User Interfaces*, 2009, pp. 97–106.

[9] J. Wong and J. I. Hong, "Making mashups with Marmite: towards end-user programming for the web," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2007, pp. 1435–1444.

[10] B. Hartmann, L. Wu, K. Collins, and S. R. Klemmer, "Programming by a sample: rapidly creating web applications with d.mix," in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 2007, pp. 241–250.

[11] M. Toomim, S. M. Drucker, M. Dontcheva, A. Rahimi, B. Thomson, and J. A. Landay, "Attaching UI enhancements to websites with end users," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1859–1868.

[12] Yahoo, "Yahoo Pipes," 2012. [Online]. Available: http://pipes.yahoo.com/.

[13] IBM, "Many Eyes." [Online]. Available: http://www-958.ibm.com/software/analytics/manyeyes/.

[14] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich, "TouchDevelop: programming cloud-connected mobile devices via touchscreen," in *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2011, pp. 49–60.

[15] "Open Data Kit." [Online]. Available: http://opendatakit.org/.

[16] S. Kim, J. Mankoff, and E. Paulos, "Sensr: evaluating a flexible framework for authoring mobile data-collection tools for citizen science," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, 2013, pp. 1453–1462.

[17] OAuth, "OAuth 2.0." [Online]. Available: http://oauth.net/2/.

[18] D. Benslimane, S. Dustdar, and A. Sheth, "Services mashups: the new generation of web applications," *Internet Computing, IEEE*, vol. 12, no. 5, pp. 13–15.

[19] "JSONPath - XPath for JSON." [Online]. Available: goessner.net/articles/JsonPath/.

[20] B. A. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*, 1st ed. Cambridge, MA, USA: MIT Press, 1993.