Smart Programming Playgrounds

Rohan Padhye, Pankaj Dhoolia, Senthil Mani and Vibha Singhal Sinha
IBM Research
{ropadhye, pdhoolia, sentmani, vibha.sinha}@in.ibm.com

***Abstract*—Modern IDEs contain sophisticated components for inferring missing types, correcting bad syntax and completing partial expressions in code, but they are limited to the context that is explicitly defined in a project's configuration. These tools are ill-suited for quick prototyping of incomplete code snippets, such as those found on the Web in Q&A forums or walk-through tutorials, since such code snippets often assume the availability of external dependencies and may even contain implicit references to an execution environment that provides data or compute services.**

We propose an architecture for smart programming playgrounds that can facilitate rapid prototyping of incomplete code snippets through a semi-automatic context resolution that involves identifying static dependencies, provisioning external resources on the cloud and injecting resource bindings to handles in the original code fragment.

Such a system could be potentially useful in a range of different scenarios, from sharing code snippets on the Web to experimenting with new ideas during traditional software development.

I. Introduction

Programmers often rely on code snippets found on the Web to learn the use of an API or language feature. In fact, Q&A sites such as StackOverflow can contain a vast coverage of an API [1] and thus can be used to provide de-facto usage examples in addition to the official API documentation [2].

However, the most reliable or popular posts often contain concise code snippets with implicit references to external dependencies and in-line placeholders for surrounding program context [3]. If a user wishes to try out the code snippet for themselves, they must first fill in these gaps before the code becomes executable.

Modern integrated development environments (IDEs) contain sophisticated components for inferring missing types, correcting bad syntax and completing partial expressions in code fragments, but they are limited to the context which has been configured for a project. Hence, users need to manually ensure that the required APIs are available before expecting IDEs to perform auto-completion.

Further, if the code fragment involves interaction with an external service, such as a file-system or running database server, the details of acquiring handles to such resources are also usually left for the users to supply. If the person reading the snippet – the consumer – is trying out new functionality, they may not have such services already running. On the other hand, if the consumer is a person asking a question on a Q&A forum to solve some immediate problem, then they may already have their environment set-up; though in this case the person answering the question with a code example – the producer – may not have the same set-up as the consumer.

Q. [JDBC] How can I get all values of a column in an SQL table into a List?

A. Try using `commons-dbutils` from Apache:

```
QueryRunner runner = new QueryRunner(dataSource);
List<String> strings =
  runner.query("SELECT * FROM my_table",
    new ColumnListHandler<String>(columnIndex));
```

Fig. 1. An example post on a Q&A site containing a Java code snippet.

In many domains, cloud computing technologies have enabled the possibility of dynamically instantiating data and compute services and composing them to drive usable applications. The combination of programming playgrounds and cloud-based resource provisioning and binding can create a powerful new paradigm for creating, sharing and consuming code fragments along with their implied contexts and required environments in a seamless manner.

In this paper, we propose a smart programming playground, that takes as input an isolated code snippet and enables it to be executed in a meaningful way through a three-step semi-automated process:

1) **Context resolution**: Resolve static code dependencies (by downloading the appropriate libraries and adding them to a path from where they can be referenced) and syntactic incompleteness (such as unqualified API references or undefined variables) and place the code fragment in an appropriate program entry-point (such as a `main` method or an event handler).
2) **Environment configuration**: Determine the physical or virtual resources referenced in the code fragment and instantiate real or simulated services to provide such resources (e.g. a database server, a search engine, a file-system, a network interface) and if required placeholder data (e.g. a textbook schema, a sample search index or stock files and directories of a particular type).
3) **Value binding**: Enable the code fragment to access these resources by injecting appropriate handles or references (e.g. connections, file-paths, URLs).

Section II motivates the problem with the help of an example. We describe the design and architecture of the smart playground in Section III followed by a brief discussion about its implications and limitations in Section IV. Section V compares our approach to existing solutions and we conclude in Section VI.

II. A Motivating Example

Consider the example in Figure 1, which is typical of a post on a Q&A site that contains a code snippet[1]. In this example, the asker is looking for a way to read all values in a column of an SQL table and retrieve them as a list of strings in a Java program. Though the asker seems to be using a standard JDBC interface, the accepted answer suggests the use of a third-party library (`dbutils` from Apache Commons[2]) to achieve the same goal.

The best way that a reader can understand and verify this answer is by trying out the code themselves. However, merely copying this snippet into their own workspace does not enable such prototyping, since it poses many challenges.

Firstly, the snippet contains references to type names (such as `List`, `QueryRunner` and `ColumnListHandler`) which are unqualified and need to be properly imported. While modern IDEs can automatically insert `import` statements for types such as `java.util.List` that are in the standard library or elsewhere in a project's class-path, they cannot resolve types such as `QueryRunner` unless the required library APIs have been added in the project's build configuration. Hence, the first step that a user needs to do before trying this snippet is to download the `dbutils` library from Apache Commons and add it to their workspace.

Secondly, the code fragment contains undefined variables such as `dataSource` and `columnIndex` which need to be initialized. A modern IDE would be able to recognize, by looking at expected types of arguments to the constructors of `QueryRunner` and `ColumnHandler`, that that these variables could be of type `java.sql.DataSource` and `int`, respectively. However, it is up to the user to actually bind values to these variables. While a user could simply enter a numeric value for the `columnIndex` value, it is not as easy to provide a value to the `dataSource` variable, since it must reference an actual database connection source.

Hence, the third step that a user must take is to ensure that there is indeed a database server running to which they can connect, and create a `java.sql.DataSource` object by setting up the appropriate connections in their code manually. Only after all three of these actions – context resolution, environment configuration and value binding – have been performed, can a user actually test the code snippet and examine its output.

To compare this with our proposed solution, a smart playground would first detect the use of the `commons-dbutils` API due to the presence of the `QueryRunner`, and upon confirming this library and its version with the user in a brief dialog also download and include the required JARs in the build path as well as insert appropriate type imports. Similarly, the smart playground would detect the requirement of a `DataSource` object, and realize that it needs to instantiate a resource of type `sql-database` to resolve this requirement,

[1]Figure 1 is a slightly modified version of http://stackoverflow.com/questions/15162976/how-do-i-query-for-a-liststring-using-dbutils#15163415

[2]http://commons.apache.org/dbutils

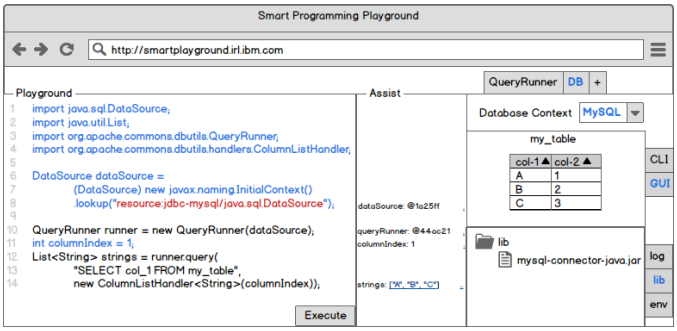

Fig. 2. A mock-up of our proposed smart playground user interface in the scenario where a user is trying to execute the code snippet found in Figure 1.

and it will thus prompt the user to choose from a list of available implementations (e.g. `mysql` or `postgresql`). These resources enable the dynamic provisioning of a database instance in some virtual machine on the cloud, followed by the injection of the appropriate handle to the `dataSource` variable in the original code.

The resource meta-data could also specify ways of manipulating its state outside of the code fragment, such as by exposing a graphical user interface (GUI). In our example, a user could quickly inspect and modify the schema and data in the provisioned database through a web-based UI.

The execution of the code fragment would then lead to a meaningful outcome of actually retrieving values from one column of an SQL table into a list of strings. The user could further play around with this code snippet to explore the usage of the API with real side-effects. A mock-up of the front-end of this system containing the resource UI, the transformed code and the output is shown in Figure 2.

III. Design and Architecture

A smart programming playground consists of several components as shown in Figure 3. A user may interact with three of these, namely the context resolution engine, the resource manager and the execution runtime. The meta-data for resolving dependencies and provisioning services is organized as a catalog of resource descriptors. The user-supplied code as well as dynamically instantiated services run in a stateful environment, represented by a cloud. The figure shows a sample work-flow for semi-automatic context resolution and resource instantiation represented by the solid lines with numbered labels indicating the order of user or component interactions.

We next describe each of the components of the smart playground briefly.

A. Resources (Descriptors and Catalog)

A resource is any external entity that is not directly specified in the code fragment but is required for its meaningful execution. Examples of resources are the `commons-dbutils` API resource and the `mysql` database resource. Each resource is described in a standardized descriptor that primarily needs to provide one or more of the following meta-data in a descriptor:

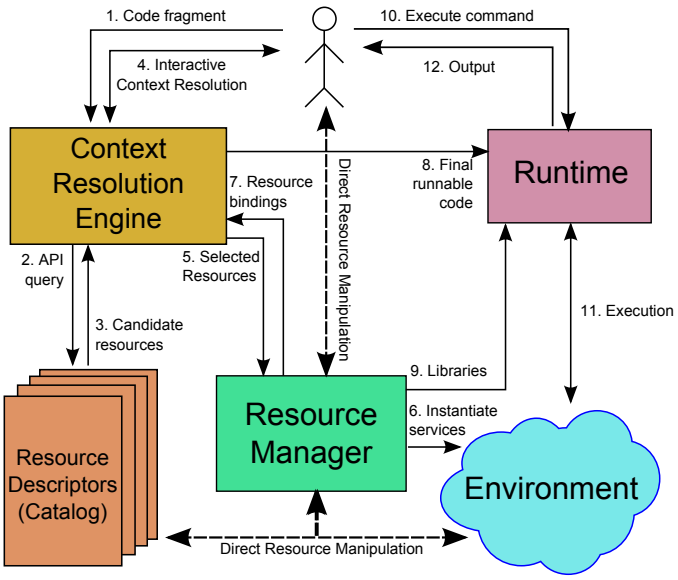

Fig. 3. Architecture of a smart programming playground, including a sample semi-automated work-flow (solid lines and numbered labels) as well as means for manual overriding (dashed lines and unnumbered labels).

1) Means to provision and de-provision the resource in the execution environment. These could be scripts or Web-service endpoints.
2) List of required application programming interfaces (APIs).
3) List of exposed value bindings for accessing provisioned resources.
4) List of resource contracts that this resource implements. This is similar to class inheritance in object-oriented programming.
5) List of other resource types that are required to be already instantiated. Dependencies may be resolved by a resource that extends the specified dependency.
6) A Web URL for inspecting and manipulating the state of a provisioned resource instance.

Figure 4 shows a sample resource catalog for six resources in the Java ecosystem, each of which declare one or more of the above meta-data declarations. In this example, API requirements are mentioned as Apache Maven [4] artifact identifiers. The *Requires* declaration indicates which other resource type must be instantiated as a pre-requisite. The *Provides* declaration indicates value bindings (e.g. Java objects) that the resource will expose once its dependencies are resolved and any provisioning scripts are complete.

The simplest resource, `commons-dbutils`, only requires an API to be downloaded and added to the code fragment's class-path to be usable. On the other hand, the `jdbc` resource requires a running database service at the back-end, which is represented by a dependency on an abstract `sql-database` resource, whose contract mentions that it must provide value bindings for a set of connection parameters after provisioning. Once the required `sql-database` resource is provisioned and its value bindings are available, the `jdbc` resource would

`commons-dbutils`
API: `commons-dbutils/commons-dbutils`

`jdbc`
Provides: `conn <- java.sql.Connection`
Provides: `ds <- javax.sql.DataSource`
Requires: `sql-database`

`@sql-database`
Provides: `url <- java.lang.String`
Provides: `port <- java.lang.Integer`
Provides: `dbname <- java.lang.String`
Provides: `user <- java.lang.String`
Provides: `pwd <- java.lang.String`

`mysql`
Extends: `sql-database`
API: `mysql/mysql-connector-java`
Provisioning/de-provisoning Script: (...)
Web GUI: (...)

`postgresql`
Extends: `sql-database`
API: `org.postgresql/postgresql`
Provisioning/de-provisoning Script: (...)
Web GUI: (...)

`sample-bank-data`
Requires: `sql-database`
Provisioning/de-provisoning Script: (...)

Fig. 4. A subset of a resource catalog containing a handful of inter-dependent resources. A prefix of `@` denotes an abstract resource.

use the exposed connection parameters to instantiate and expose `Connection` and `DataSource` objects as declared in its *Provides* declaration. The `sql-database` resource contract is implemented by two concrete resources in this example: `mysql` and `postgresql`, each requiring a different library to be downloaded as well as having scripts for provisioning ephemeral database instances. These resources also generate Web URLs to allow a user to inspect and manipulate their state via a graphical user interface (GUI); in this example since the resources are database instances the GUI would allow viewing or modifying the database schema and data. In addition to manually seeding data in the database, a user may also choose to instantiate the `sample-bank-data` resource, which only requires a running database server and contains scripts to populate the database with a sample schema for bank accounts. Such placeholder data may be useful for sharing code snippets that demonstrate the use of the API and are indifferent to any underlying data layout.

B. Context Resolution Engine

The context resolution engine analyzes the original source code fragment and identifies implicit references to APIs as well as notes all undefined variables and their types. It queries the catalog to find a candidate set of resources that match the APIs and value bindings required. Several approaches exist for determining the APIs referenced in a piece of un-compilable code [2], [5]. In case of multiple results from the catalog query, this engine can engage in a dialog with the user for choosing a particular resource type and version.

C. Resource Manager

The resource manager processes the meta-data in resource descriptors to (1) download required libraries for the runtime, (2) execute required scripts to instantiate services, (3) create value bindings for the code snippet that reference the instantiated service instances, and (4) allow monitoring and manipulation of the state of the service instances.

Users may choose to bypass the context-resolution engine completely and pick resources manually, as shown by the dashed lines in Figure 3 representing an alternative work-flow.

D. Runtime

The runtime is the engine that executes the final code, with all static dependencies resolved and dynamic resource handles bound to appropriate values. Several runtimes may exist to support different programming languages or application frameworks with different entry points.

This architecture of runtimes and resources can be naturally implemented on top of Platform-as-a-Service (PaaS) architectures comprising of containers and services, such as IBM Bluemix [6], which itself is based on Cloud Foundry [7].

IV. Discussion

It is important to note that our proposed system is not attempting to guarantee the automatic execution of *any* code snippet found the Web. For one thing many code snippets are syntactically broken or incomplete. While some instances of syntactic incompleteness can be automatically repaired, such as the use of ellipses "..." or comments as placeholder values, this is not always possible in general. Second, not all implicitly referenced APIs in a code snippet can be identified. The *partial program analysis* approach [5] works well only in the presence of explicit imports, while the *oracle*-based approach [2] is limited by the APIs indexed by its oracle. Third, even if the referenced code dependencies are identified, the resource catalog needs to be very large in order to identify all types of required external services and provide implementations for their provisioning. We do not expect a hand-made monolithic resource catalog to be available from the start, but envision an incremental approach where users can easily manually specify the required resources if they cannot find them in the catalog; this action would add the resource to the catalog for future users. Existing projects such as Apache Maven [4] and Cloud Foundry [7] have demonstrated the feasibility of this approach by crowd-sourcing large repositories of API artifacts and runtime environments respectively.

V. Related Work

To aid a developer working with an unfamiliar API, CodeHint [8] allows a user to execute a partial program up to some break-point where they are stuck, and proposes synthesized code expressions that satisfy some dynamic post-condition. This tool helps a developer who already has a context, but is looking for code to perform some function using that context. Smart playgrounds address the converse case, where a user has found some code that is using an API or service, but is missing the preceeding context.

Several tools have also been developed to enable users to try out code snippets found on the Web. However, these tools support a very limited notion of context. For example, Runnable.com [9] hosts many user-contributed snippets for performing common tasks in various languages, such as processing file-uploads in PHP. While such examples can actually be executed and tried out on the Web, other PHP snippets such as a user-login script using MySQL cannot be executed unless a database is separately provisioned.

Programming playgrounds for scripting languages have also become popular for their ability to provide instantaneous feedback of the results of a computation or for visualizing its side-effects. Two great examples of this paradigm are JSFiddle [10], a popular platform for sharing JavaScript + HTML snippets that shows the resulting dynamic Web page alongside the code editor, and Apple's Xcode playground for the Swift language [11], that allows dynamic visualization of data structures and UI elements. However, these platforms are also limited to the context specified in the code snippets and do not assist in creating or simulating external services.

The simulation of an external environment is commonly used in test-driven development through the use of mock objects [12]. Mock objects could be used to prototype code snippets as well, though the burden of mocking these objects again falls on the user. Also, since the goal of a playground is different from that of a test framework, it is possible that the user is actually trying to test out the service itself, rather than the code accessing it. Our approach solves both these issues by provisioning real service instances in the cloud.

VI. Conclusion

The advent of cloud technologies and the increasing use of the Web and social media to share code has made it possible to develop new paradigms of sharing not just code fragments, but executable code environments. In this paper we have proposed such a smart programming playground, that can help resolve programming context and instantiate dynamic services to enable rapid prototyping of shared code snippets. With such a system, we believe that developers who are exploring new technologies can forget worrying about environment configuration and resource provisioning and focus on the creative process of working with code.

References

[1] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, ser. Web2SE '11, 2011.

[2] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014.

[3] J. Sillito, F. Maurer, S. M. Nasehi, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, ser. ICSM '12, 2012.

[4] "Apache Maven," http://maven.apache.org, Accessed: February 2015.

[5] B. Dagenais and L. Hendren, "Enabling static analysis for partial Java programs," in *Proceedings of the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications*, ser. OOPSLA '08, 2008.

[6] "IBM Bluemix," http://www.ibm.com/software/bluemix, Accessed: February 2015.

[7] "Cloud Foundry," http://cloudfoundry.org, Accessed: February 2015.

[8] J. Galenson, P. Reames, R. Bodik, B. Hartmann, and K. Sen, "CodeHint: Dynamic and interactive synthesis of code snippets," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE'14, 2014.

[9] "Runnable.com," http://runnable.com, Accessed: February 2015.

[10] "JSFiddle," http://jsfiddle.net, Accessed: February 2015.

[11] "WWDC 2014 Session Videos - Swift Playgrounds," https://developer.apple.com/videos/wwdc/2014/?id=408, Accessed: February 2015.

[12] K. Beck, *Test-driven development by example*. Addison-Wesley Longman, 2002.