Static Checking of TouchDevelop Programs against Web Service Specifications

Master's Thesis Proposal

Pascal Zimmermann
Advisor: Lucas Brutschy

December 6, 2013

Background

TouchDevelop is a programming environment developed by Microsoft Research. TouchDevelop allows users to create applications for their mobile devices with a user interface that is optimized for development directly on the device using a touch screen; code can be built using predefined blocks which greatly reduces the need for keyboard inputs. TouchDevelop is especially targeting programmers without much programming experience, and – as the only requirement to write and run TouchDevelop programs is a web browser – it can be used on many different devices. Currently, all programs written in TouchDevelop are collected by Microsoft and are publicly available.

The barrier to publishing TouchDevelop scripts is very low, leading to potentially many publications by inexperienced programmers. The TouchBoost project aims to provide static program analysis tools for the TouchDevelop programming environment. The number of publications of defective scripts can be reduced by statically discovering software bugs. TouchBoost is based on a static analyzer called Sample (Static Analysis of Multiple LanguagEs) which is based on the abstract interpretation framework [4]. Sample is being developed by the chair of programming methodology at ETHZ.

Motivation

Many applications that are developed for mobile devices make use of web APIs. While the type system greatly helps to check validity of calls to objects used with traditional APIs, the arguments of web API methods are often string values carrying some well defined semantics. Consider the following example:

```
var textQuery := "http://maps.googleapis.com/maps/" ∥
     "api/geocode/xml?latlng=#loc&senses=true"
var currentLoc := senses→current location
textQuery := textQuery→replace("#loc", currentLoc→to string)
var xml := web→xml(web→download(textQuery))
data formattedAddress := xml→child("GeocodeResponse")
     →child("result")→child("formatted_address")→to string
```

The program in the example first initializes a string that represents a URL. The program then passes the URL string to the `download` method which returns a string that is parsed as an XML object. The program then reads a value of this XML object, simply assuming that this value exists. When verifying the program, we need to know whether the returned string of download can actually be parsed as an XML object, and whether this XML object has a child called GeocodeResponse. However, the specification of the download method is not strong enough and we need the specification provided by the web service that is called.

Most web APIs provide some sort of specification – either formal or informal. A few of them provide machine-readable specifications written in the Web Application Description Language (WADL) [5] which is an extensible specification language based on XML. There also exist other specification languages such as the JSON-based Swagger.

Core

The first objective of this Master's thesis is the extension of an existing string analysis such as [3], driven by the need to analyze URL strings. The analysis should deal with common string operations, e.g., concatenation and substring replacement. Furthermore, the analysis shall also track the relation between substrings and other values in the program. Other values can for example be numeric values that are converted into a string value, or the location in our motivating example that is explicitly converted using the to string method. The analysis should be kept as general as possible while being able to extract the features of URL strings.

The second task is to use the implemented string analysis to select the correct web API specification and to check the TouchDevelop programs against them. The checks include type checks as well as semantic checks. The listing below shows a simplified contract of the web API call in our motivating example.

```
<resources base="https://maps.googleapis.com/maps/api/geocode">
  <resource path="/{format}?latlng={lat},{lng}">
    <param name="format" required="true"
      default="xml" style="template">
      <option value="xml" mediaType="application/xml"/>
      <option value="json" mediaType="application/json"/>
    </param>
    <param name="lat" required="true"
      type="xsd:decimal" style="template">
      <minimum value="-90"/>
      <maximum value="90"/>
    </param>
    <param name="lng" required="true"
      type="xsd:decimal" style="template">
      <minimum value="-180"/>
      <maximum value="180"/>
    </param>
    <method id="inverse-geocoding" name="GET">
      <request>
        <param name="sensor" required="true"
          type="xsd:boolean" style="query"/>
      </request>
      <response>
        <representation mediaType="application/xml"
          element="GeocodeResponse"/>
        <representation mediaType="application/json"/>
      </response>
    </method>
  </resource>
</resources>
```

The developed analysis shall be implemented in Scala as part of the TouchBoost tools. The implementation will be tested and evaluated using existing TouchDevelop programs. Apart from the implementation, a report on the project is to be delivered and a final presentation concludes the thesis.

The following deliverables must be handed in by the end of the thesis:

- The code that implements the designed analysis
- A formalization of the abstract semantics on a minimal language
- The results of an evaluation against web application specification on the corpus of all available TouchDevelop programs
- A comprehensive test suite with expected results

Possible Extensions

After the completion of the core, some of the following extensions should be done depending on the time left:

- Further abstract the Bricks analysis to produce a unique representation of the URL. Commonly, the query part of a URL is a set of key-value pairs. We would like our analysis to find equivalent URLs by defining equivalence classes.
- Implement a data-flow analysis to dynamically disable the analysis for string variables that are not used in web API calls. String is a very common data type serving many purposes and analyzing all string variables is potentially wasteful.
- Extend the string analysis to find strings that do not form a syntactically well-defined URL, e.g., strings that contain characters that are prohibited by the URL syntax. There are three cases where we are not able to retrieve web service specifications: Either the string analysis is not powerful enough to deduce the targeted web API, we don't have specifications for the targeted web API available, or the URL is wrong.

Schedule

The whole thesis must be done within six months. It is planned to hold weekly meetings with the advisor to discuss the progress and ideas. In this section we describe a tentative schedule.

2 Weeks	The first two weeks are dedicated to get familiar with the technical environments, i.e., Scala, Sample and TouchDevelop. This time will also be used to get familiar with the implementation of the Bricks analysis as described in [3], and to do further research, i.e., to read papers and to find example programs written in TouchDevelop.
6 Weeks	The following six weeks are used to design, formalize and implement an extension to the existing string analysis.
3 Weeks	A period of three weeks is planned to implement the checking of the TouchDevelop programs against WADL-specifications.
8 Weeks	After completing the core, eight weeks are planned to work on extensions.
5 Weeks	A period of five weeks is dedicated to working on the report and to evaluating the analysis.
1 Week	One week is dedicated to preparing the presentation.

Related Work

Recently, researchers have developed various methods to statically analyze the values of strings. The general-purpose data type String is present in many programming languages and there are numerous applications of string analysis such as statically checking the syntax or semantics of dynamically generated SQL queries.

Christensen et al. [2] describe a string analysis that abstracts string values using a regular language. Their analysis has been implemented to analyze Java strings. However, they do not fit their algorithm into the abstract interpretation framework and the analysis has difficulties dealing with strings stored in heap variables.

Choi et al. [1] solve the problem of dealing with heap variables and integrate constant propagation by designing a string analyzer that uses abstract interpretation techniques. One of their main contributions is the design of a widening operator to deal with lattices of infinite height.

Kim and Choe [6] have developed a string analysis that can analyze context-free languages. The analysis uses abstract interpretation techniques. The abstract domain they propose is a predicate domain on transitions of pushdown automata that read the string. However, this analysis can only check whether a string belongs to a certain grammar which only allows some syntactic checks.

Constantini et al. [3] also build their analysis in the abstract interpretation framework. They develop different abstract domains with different tradeoffs in complexity and expressiveness. Their two most expressive abstract domains are called the Bricks domain and the String Graph domain. Both domains capture character inclusion and order. Their approach is generic and – thanks to abstract interpretation – modifiable. The Bricks abstract string domain has been implemented as part of the Sample analyzer. The abstract semantics are only defined for pure string operations; we want to extend the existing implementation to track other values related to the string variables.

References

[1] Tae-Hyoung Choi, Ouksen Lee, Hyunha Kim, and Kyung-Goo Doh. A practical string analyzer by the widening approach. In *APLAS*, pages 374–388, 2006.

[2] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Precise analysis of string expressions. In *SAS'03*, pages 1–18, 2003.

[3] Giulia Costantini, Pietro Ferrara, and Agostino Cortesi. Static analysis of string values. In *ICFEM'11*, 2011.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252, 1977.

[5] Marc Hadley. Web application description language. http://www.w3.org/Submission/wadl/, 2009.

[6] Se-Won Kim and Kwang-Moo Choe. String analysis as an abstract interpretation. In *VMCAI'11*, pages 294–308, 2011.