LIMO: Learning Programming using Interactive Map Activities (Demo Paper)*

Ruby Y. Tahboub Jaewoo Shin Aya Abdelsalam Jalaleldeen W. Aref
Walid G. Aref Sunil Prabhakar
Purdue University, West Lafayette, Indiana
{rtahboub,shin152}@cs.purdue.edu, aya.abdelsalam.91@gmail.com,
{jaref,aref,sunil}@cs.purdue.edu

ABSTRACT

Advances in geographic information, interactive two- and three-dimensional map visualization accompanied with the proliferation of mobile devices and location data have tremendously benefited the development of geo-educational applications. We demonstrate LIMO; a web-based programming environment that is centered around operations on interactive geographical maps, location-oriented data, and the operations of synthetic objects that move on the maps. LIMO materializes a low-cost open-ended environment that integrates interactive maps and spatial data (e.g., OpenStreetMap). The unique advantage of LIMO is that it relates programming concepts to interactive geographical maps and location data. LIMO offers an environment for students to learn how to program by providing: 1. An easy-to-program library of map and spatial operations, 2. High-quality interactive map graphics, and 3. Example programs that introduce users to writing programs in the LIMO environment.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education; D.2.6 [**Programming Environment**]: Map-integrated Environment.

General Terms

Design.

1. INTRODUCTION

In recent years, the interest in learning programming and computer science classes among undergraduate students has been modestly growing in comparison with STEM[1] [11]. The key to inspire students to learn computer science lies in adopting an active learning approach. Many academic institutions have redesigned the introductory computer science course to include a visual component e.g., Alice [8] and Greenfoot [9], a hardware component e.g., Finch robot [10], or a context, e.g., media computation [7]. Although these approaches have successfully raised the interest in computing and made programming accessible, none of the environments provide a platform for maps as a pedagogical tool for learning programming. Interactive maps (e.g., based on OpenStreetMap, Google Maps, Bing Maps, or Google Earth) and location-based services provide a very viable alternative and a unique opportunity to add excitement to students while learning programming by relating programming concepts to locations and places that students are familiar with. Moreover, map activities, e.g., finding directions with certain properties, scale conversion, coordinate systems, distance computations, shortest paths in road networks, and spatial analysis are rich in semantics and help students develop strong computational thinking skills.

*Walid G. Aref's research is partially supported by the National Science Foundation under Grants IIS 1117766 and IIS 0964639.

[1]Science, Technology, Engineering and Mathematics.

SIGSPATIAL '15, November 03-06, 2015, Bellevue, WA, USA

http://dx.doi.org/10.1145/2820783.2820796 ...$15.00.

2. THE LIMO PROGRAMMING ENVIRONMENT

We have realized a prototype for LIMO (Please see http://ibnkhaldun.cs.purdue.edu:8181/limo/). In this section, we present a first view of the LIMO environment including its user interface and programming library. Also, this section covers an overview of the LIMO system design.

2.1 LIMO User Interface

LIMO provides a web-based environment that integrates Python scripting and interactive maps to facilitate creating map-visualized programs. The LIMO user interface, illustrated in Figure 1, is comprised of three main parts: a program *scripting area*, a *program output area* and an *interactive map*. Users can create programs by writing Python scripts that utilize the LIMO library to write map programs. Furthermore, the LIMO interface integrates OpenStreetMap that is used to explore map features, e.g., roads and parks, and visualizes the output of the executed programs, e.g., an animation of the *Commuter* as it follows moving directions on the map. Finally, the program output area is dedicated for displaying textual output, e.g., computational results performed by the executing program.

Writing programs in LIMO is simple. First, the program scene is set as a map that is zoomed at a default location. Next, the user writes a standard Python script that utilizes the LIMO library to encode the actions of the program's

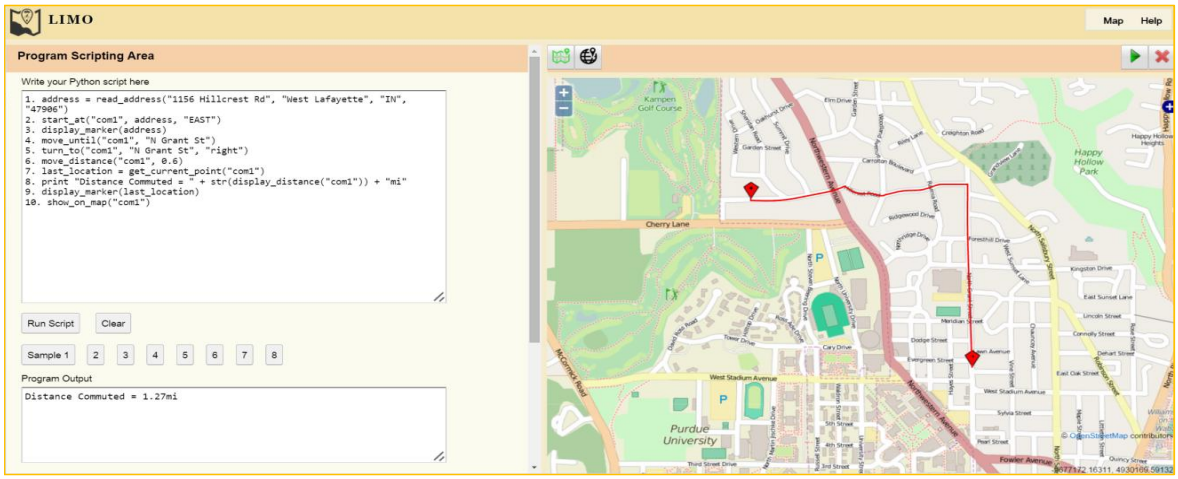

Figure 1: The LIMO programming environment consists of *program scripting area*, *a program output area* and an *interactive map*.

moving object that is referred to by the *Commuter*. Finally, on running the program, the animation of the movements of the *Commuter* is displayed on the map. Figure 1 gives an example program that provides the commuting directions between two locations. Function *start_at* (Lines 2) specifies the start location and direction of *Commuter*. Next, Function *display_marker* (Line 3) adds a marker at the current location of *Commuter*. Constructs *move_until*, *move_distance* and *turn* (Lines 4-7) control *Commuter's* movements on the map. Function *display_distance* (Line 8) prints in the program output area the *total distance* commuted so far. Finally, Function *show_on_map* displays an animation of Commuter's movements on the map.

2.2 The LIMO Programming Library

LIMO integrates interactive maps and rich location-oriented data and spatial data types into an easy-to-use programming library. The LIMO library functions are designed to enable users to process, customize, and visualize location data on the map. The LIMO library offers two categories of functions: *map* basics, and *spatial*.

The *map* functions enable users to interact with the underlying map and perform primitive activities, e.g., displaying a message or adding a marker on map. Location in the basic constructs is represented as an actual address that can either be explicit, e.g., the intersection of two streets, or implicit, e.g., the *commuter's* current location. Moreover, the basic map functions are suitable for beginners with no map programming experience. For instance, the semantics of *display*, *move* and *turn* are intuitive and have basis in reality. Hence, a wide variety of programs can be written to describe the various movements of *commuter* while keeping track of time and distance.

The *spatial* functions encapsulate complex location-based data and operations into easy-to-use functions. Location is represented as a geo-coordinate point, e.g., using the latitude and the longitude. Spatial functions enable creating programs that incorporate real-world location data and apply.For instance, *get_location* seamlessly converts a textual address to its equivalent geo-coordinates, *get_all* is a spatial function that provides a list of locations (in the form of geo-coordinates or shapes) that match a textual place *description* parameter (e.g., all airports in Indiana). Consequently, given the current location of *Commuter*, one program may iterate over the list of interesting locations and finds the

Table 1: Sample operations of the LIMO programming library

Category	Function Name	Description/ Options
Map (Basics)	start_at(name, address, direction)	Sets Commuter's start location to address with heading direction
	move_distance(name, distance), move_until(name, street), move_to_next_intersection (name)	Move Commuter for certain distance or until a certain street or next intersection
	turn_to(name, streetName, direction)	Re-orient Commuter towards a new street and direction, e.g., 123 University St. and South
	display_message (message, address|location)	Place a text message, e.g., at a given address or geo-location
	display_marker (address| location)	Place a map marker at a given address or geo-location
	display_[distance| time](commuter)	Display total distance/time commuted so far
	compute_distance(add1, add2)	Return the distance between two addresses or geo-locations
Spatial	get_location(address)	Return a point that represents the geo-coordinates of address
	get(name, description, geometric shape)	Return the location (as geometric shape) of the place that matches the given name and discription
	get_all(description, geometric shape)	Return a list of locations (as geometric shape) for places that match discription
	overlaps| touches| intersects| contains (shape1, shape2)	Boolean operators that test whether two shapes overlap, touch, intersect, or contain one another
	display_shape (geometric shape)	Display geometric shape (e.g., lake boundary) on map

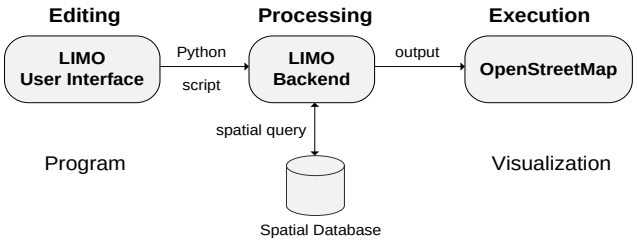

Figure 2: Process flow diagram of a program in LIMO.

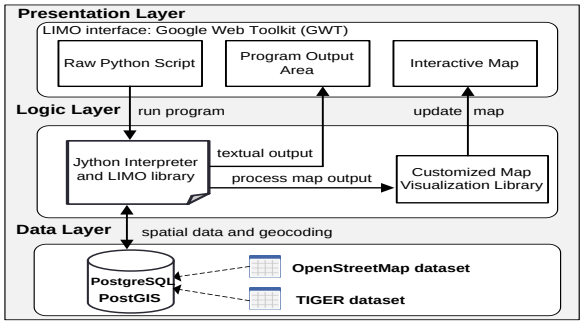

Figure 3: The architecture of the LIMO programming environment.

closest (or farthest) location to *Commuter*'s current location. Another program may simply visualize all the qualifying locations on a suitably scaled map. Spatial predicates, e.g., *intersect* or *contains* further enrich the programming library by enabling spatial tests, e.g., does the park contain a lake? Refer to Table 1 for a list of sample operations of the LIMO programming library.

2.3 System Design

Figure 2 gives the process flow of a LIMO program inside the LIMO programming environment. In the editing phase, the user creates and edits her program. Upon program execution, LIMO's backend analyzes the program script and invokes the proper LIMO library functions to carry out map and spatial operations. For example, the ***display_marker****("1001 Hillcrest Rd ...")* requires a spatial query that obtains the geo-coordinates of the address parameter. The geo-coordinates are used to visualize the construct on the map. Then, the backend performs the actions specified by the program. Finally, in the execution phase, the program output is visualized on the map.

The LIMO programming environment is a web application that follows a multi-tier architecture. Refer to Figure 3. In the Figure, LIMO's presentation layer embodies a web-based user interface that integrates a scripting area and an interactive map. The middle layer (i.e., Logic) consists of a Jython interpreter (a Java-based implementation of Python), the LIMO library of functions, and a customized map visualization library. Finally, the data layer uses a relational database that handles spatial datasets. We discuss the technical details of each layer next.

2.3.1 Presentation Layer

LIMO's user interface is built using Google Web Toolkit (GWT) [2]; an open-source web tool for developing JavaScript front-end applications. GWT is appealing for LIMO due to its cross-browser compatibility, efficient testing and debugging, and not mandating prior background in web languages (e.g., JavaScript or HTML).

An interactive map based on OpenStreetMap (OSM) is integrated into LIMO's user interface. OSM provides a free editable map of the world. Moreover, OSM offers open access to map datasets. Finally, LIMO uses the OpenLayers API [3]; an open-source library for map development, to realizw visualizations on LIMO's interactive map.

2.3.2 Logic Layer

The logic layer (also termed the Application Layer) plays a key role in executing programs in LIMO including performing computations and communicating with the Data Layer. The Jython interpreter processes the python script and utilizes the LIMO library to carry out spatial and map operations. On program execution, the textual output is displayed on the program output area and the map-related data is moved to a customized map visualization library to create object movement animation. Finally, the interactive map is updated with program visualizations.

2.3.3 Data Layer

The data layer is comprised of a relational database management system (DBMS) and stored datasets. LIMO deploys PostgreSQL [5]; an open-source relational DBMS along with PostGIS [4]; an open-source extension that adds support for spatial data to PostgreSQL.

The spatial database in LIMO stores TIGER [1] and OpenStreetMap datasets. TIGER is a public dataset administrated by the U.S. Census Bureau [6]. TIGER consists of geometric data that features roads, railroads, rivers, in addition to legal and statistical geographic areas. LIMO uses TIGER to support geocoding, e.g., translating between address and geo-coordinates. Geocoding is crucial for LIMO in order to visualize the actions of constructs on the map. OpenStreetMap contains rich datasets about the locations of points of interests (e.g., tourist attractions), lines (e.g., roads), and areas (e.g., lakes).

3. DEMONSTRATION

This section provides sample LIMO programs to demonstrate the various programming library functions and showcase how the LIMO environment can be used to create stimulating open-ended programming exercises.

3.1 Basic Programs

Program 1. This program presents an example of utilizing *display*, *compute-distance*, and conditional structure *If ... Else*. The goal of the program is to determine whether the user should walk or bike from home to the office. The user is willing to walk *only if* the distance to office is less than 3 miles. Otherwise, she prefers to bike. Moreover, the program adds a marker and a message on the locations of home and office.

```
1  Home = read_address("1001 Hillcrest Dr.",
   "WestLafayette", "IN", "47906")
2  Office = read_address("Airport Dr.",
   "West Lafayette", "IN", "47907")
3  display_marker(Home)
```

Figure 4: (a) The output of Program 2. (b) The output of Program 3.

```
4  display_message("Start", Home)
5  display_marker(Office)
6  display_message("Destination", Office)
7  distance = compute_distance(Home, Office)
8  if distance < 3 :
9    print "I'll be walking!"
10 else:
11   print "I'll be biking!"
```

A simple extension to Program 1 is to add a third option for driving in case the distance between home and office is greater than 5 miles.

Program 2. This program presents an example of utilizing *get-all*, *display-shape*, the spatial operator *touches*, conditional, and looping structures. The goal of the program is to iterate over a list of geometries that represent the boundaries of states. It is required to identify the states that share borders with *Indiana* and display a boundary around each one of them.

```
1  Ind_pol = get("Indiana", "STATE", "POLYGON")
2  Ind_pnt = ("Indiana", "STATE", "POINT")
3  display_message("Indiana", Ind_pnt)
4  display_shape(Ind_pol)
5  all_states = get_all("STATE", "POLYGON")
6  for state in all_states:
7    if touches (state, Ind_pol):
8      display_shape(state)
```

Program 3. This program presents an example of utilizing *get*, *display_shape*, the spatial operator *intersects*, conditional and looping structures. The goal of the program is to display the I-65 Highway segments that intersect the boundary of the city Indianapolis. The *get* constructs (Line 1) obtains a list termed *I65_segments* that contains all of the I-65 Highway segments. Refer to Figure 4b for illustration, the *I65_segments* spans multiple states (e.g., Michigan, Indiana, ...). Next, the boundary of Indianapolis city is obtained (Line 2) and the spatial operator *intersects* is used inside a looping structure to filter out the unqualified segments (Lines 4-6).

```
1  I65_segments= get("I- 65", "PRIMARY-ROAD",
   "POLYLINE")
2  Ind_pol= get("Indiana", "STATE", "POLYGON")
3  display_shape(Ind_pol)
4  for i in range(len(I65_segments)):
5    if intersects (I65_segments[i], Ind_pol):
6      display_shape(I65_segments[i])
```

3.2 Some Open-ended Problems

Interactive maps can be used to construct open-ended programming problems, i.e., ones that can have multiple potential solutions. In the following, we present ideas for some open-ended problems relevant to the LIMO environment.

Program 4. **The design of a 5K race route**. This problem has a restriction on the race distance in addition to the start and end locations. In contrast to the *Commute Directions* program, there might be multiple or even *no* possible solutions.

Program 5. **Planning a road trip**. This problem involves choosing the points of interest and allocating the gas budget that is determined by the total miles to be commuted. Many compelling questions can be addressed. For instance, finding a trip route that allows the user to visit as many places as the budget and miles restrictions allow.

Program 6. **Data visualization activities**. There are many real-world activities that involve visualizing data on maps, for instance, keeping track of social media friends and followers around the country. Friends' location data can be constructed by using Operation *get_location*. Another example is monitoring the local weather, e.g., storms, snow accumulation, and other relevant weather data.

Program 7. **The design of a map game**. Hunting for hidden treasures is a very intriguing game for all ages. Markers on the map can be used to represent treasures and the goal is to use the LIMO library to encode the path to each treasure. In this game, user-defined metrics (e.g., the distance or the number of turns) can be used as scoring criteria to determine favorable paths among others.

Acknowledgments

We would like to thank Susanne E. Hambrusch for her valuable comments and suggestions on how to improve LIMO Environment.

4. REFERENCES

[1] Census tiger. `http://www.census.gov/geo/maps-data/data/tiger.html`.
[2] Google web toolkit. `http://www.gwtproject.org/`.
[3] Open layers. `http://openlayers.org/`.
[4] Postgis. `http://postgis.org/`.
[5] Postgresql. `http://www.postgresql.org/`.
[6] Us census bureau. `http://www.census.gov/`.
[7] M. Guzdial. A media computation course for non-majors. In *ACM SIGCSE Bulletin*, volume 35, pages 104–108, 2003.
[8] C. Kelleher, R. Pausch, and S. Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *CHI*, pages 1455–1464. ACM, 2007.
[9] M. Kölling. The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):14, 2010.
[10] T. Lauwers and I. Nourbakhsh. Designing the finch: Creating a robot aligned to computer science concepts. 2010.
[11] S. Zweben and B. Bizot. 2014 taulbee survey. *COMPUTING*, 27(5), 2015.