Satsy Design Document

Team: May14_13
Date: October 4, 2013
Members: Carl Chapman, Cody Hoover, Cole Groff, Kaitlin McAbee, Trevor Lund
Advisor: Kathryn Stolee

Introduction

Project Definition

This project is a web-based search engine for source code called Satsy. Satsy is based on the fact that every program has one or more executable paths, and that Satisfiability Modulo Theory(SMT) solvers can be used to evaluate whether or not a particular executable path in a program can satisfy an input-output specification.

For testing, a small set of programs have had their paths encoded into the SMT-LIB2 [1] format. Satsy will combine user-defined input-output pairs with this set of translated source code methods and execute Microsoft's Z3 [2] SMT solver on each combination. Satsy will parallelize the execution of Z3 on these combinations so this search can be done more quickly. This project will run on Amazon Web Service's (AWS) Elastic Cloud Compute (EC2) servers because they provide the option of elastically increasing computational power as needed [3].

The results of Z3's execution on each program path, input-output specification pair are collected and returned to the user. The intention of this project is to rank these results based on the number of input-output pairs a particular program was able to satisfy, and other related metrics to be explored in the future. Once ranked results are displayed to the user, the user will be able to refine their search and search again as desired.

Project Goals

- Satsy should be able to return source code that satisfies the user-specified input-output pairs, ranked by its ability to satisfy the specification.
- Satsy should be able to conduct its search quickly and return a user's search results in less than 5 seconds per search.
- Multiple users should be able to run searches at the same time. Satsy should be able to support up to 100 users at a time until further expanded.
- Satsy should be designed to be able to scale to quickly search up to 10 GB of encoded source code.

Deliverables

- A web application supporting a search page and a results page running on an AWS server.
- Code containing search logic that parallelizes execution of Z3 and combines results.

- An algorithm to sort the results of the Z3 search in an order that brings the most relevant blocks of source code to the top of the results.
- Module-level unit tests.

System Level Design

System Requirements

Functional Requirements

- Satsy will be able to accept multiple input-output pairs as search terms from each user, up to 10 pairs.
- Satsy will be accessible from anywhere that has internet access.
- Satsy will return a set of source code snippets given a set of input-output pairs.
- The source code that Satsy returns should be able to take the given inputs as parameters and produce the given output as a returned result.
- The source code results nearer to the top of the search result page should be satisfiable by most or all of the input-output pairs.
- Source code results that are satisfiable by all of the input-output pairs will be sorted by other criteria to be specified later.
- Satsy needs to provide a clean interface to interact with the user, to be determined by user feedback.
- Satsy will provide feedback to the user if they didn't format their search terms correctly or if an error occurred.
- Satsy will display an input search box and an output search box.
- Users will be able to add additional sets of search boxes on the search page.
- The system will handle Java string, int, boolean, and character inputs, and allow extensions for further language support.

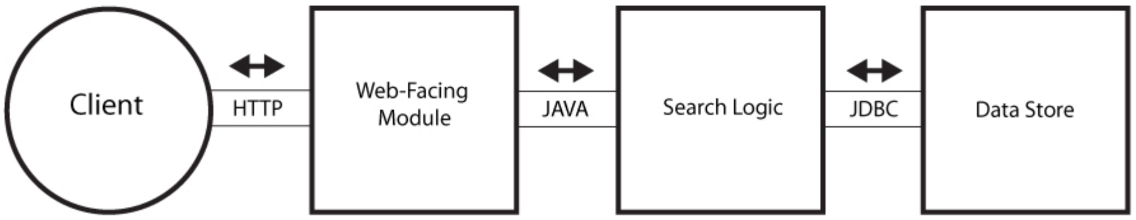
Non-Functional Requirements

- Users should quickly be shown the results of their search upon submitting their search criteria.
    - Metric: There shall be no more than 5 seconds between the user clicking the search button and the user seeing their search results.
- Searching with Satsy should be intuitive to the user
    - Metric: Less than 15% of users should express confusion when using Satsy.
    - Metric: 90% of users should be able to construct a valid search query within 1 minute of using Satsy.
- Satsy needs to support returning many different source code results.
    - Metric: Up to 1000 source code results can be returned.
- Satsy will be able to handle multiple user searches at once.

    - Metric: Up to 100 searches at once will be supported.
- Results should be accurate.
    - Metric: Results should return with a false positive rate of <15% and a false negative rate of <5%.

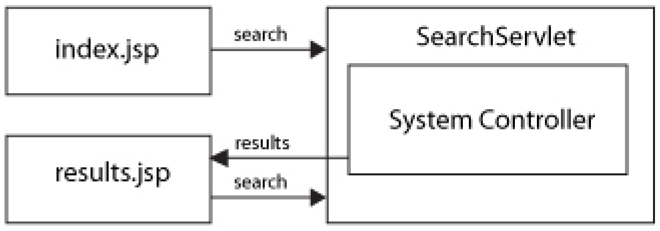
Functional Decomposition

High Level Decomposition



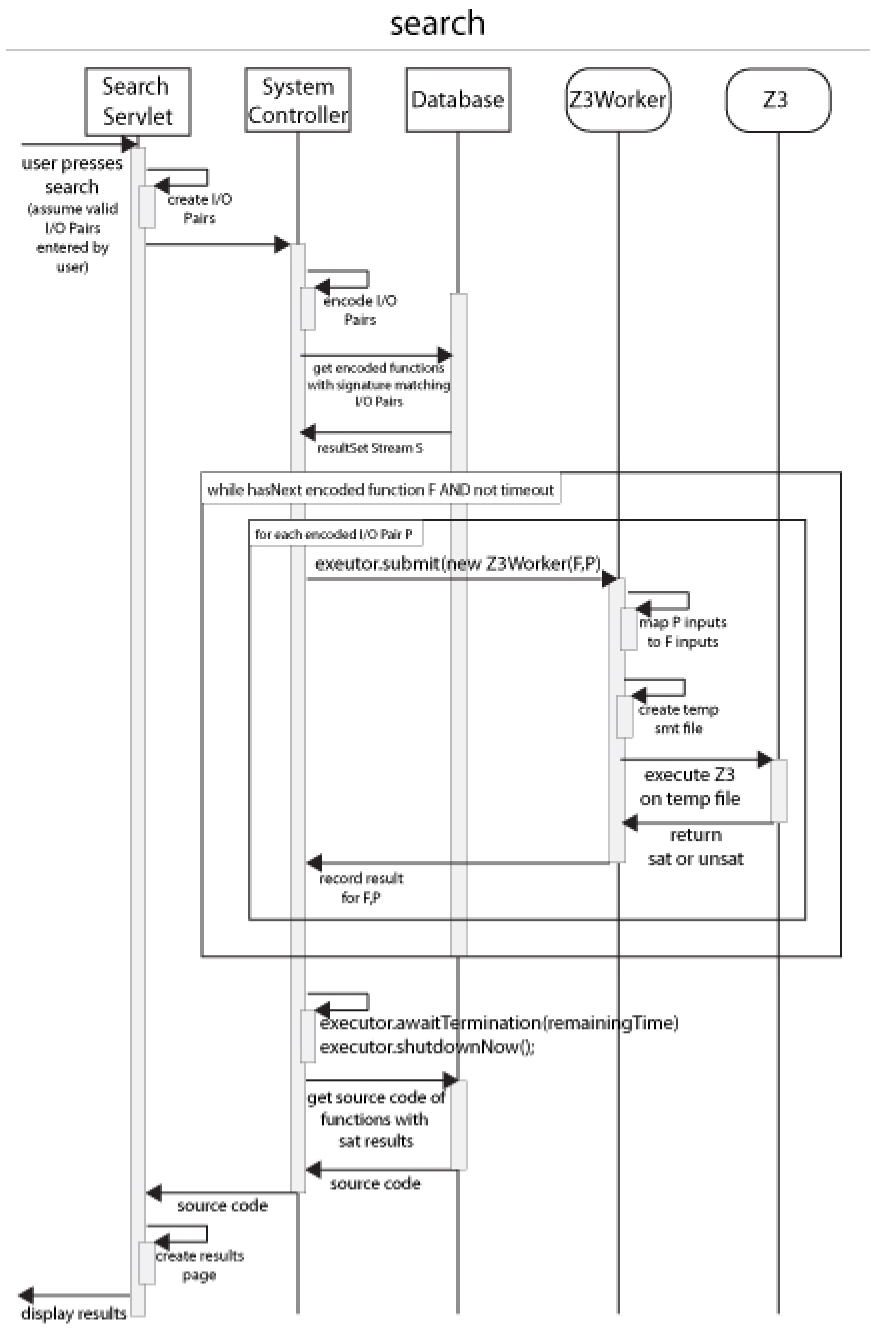
At a high level, Satsy can be decomposed into three modules. The web-facing module interacts with the client through HTTP, and communicates with the search logic module through java. The search logic module communicates with the web-facing module through java and with the data store module using JDBC. This architecture closely matches the ‘state-logic-display’ pattern.

Web-Facing Module



When the client fills out the form on the index.jsp page and presses the ‘search’ button, the web-facing module initiates a java search servlet. This servlet instantiates all the search logic and creates a results.jsp populated with relevant results. From this results page, the client can initiate more searches as desired.

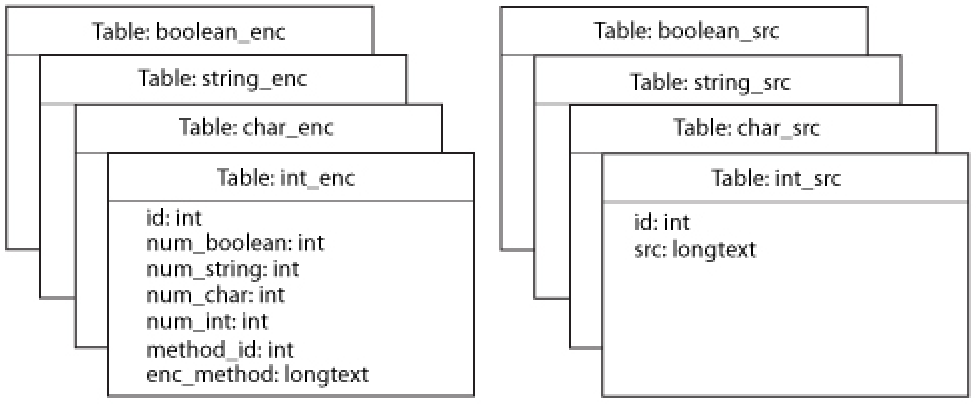
Search Logic Module


search
Search Servlet
System Controller
Database
Z3Worker
Z3
user presses search (assume valid I/O Pairs entered by user)
create I/O Pairs
encode I/O Pairs
get encoded functions with signature matching I/O Pairs
resultSet Stream S
while hasNext encoded function F AND not timeout
for each encoded I/O Pair P
exeutor.submit(new Z3Worker(F,P)
map P inputs to F inputs
create temp smt file
execute Z3 on temp file
return sat or unsat
record result for F,P
executor.awaitTermination(remainingTime)
executor.shutdownNow();
get source code of functions with sat results
source code
source code
create results page
display results

The above sequence diagram outlines the major actions involved in searching. The system controller object encodes the I/O pairs created from text provided in the fields of the index.jsp or results.jsp web page. The database is queried for all functions that match the provided signature. While streaming through these results and monitoring the time remaining to complete the search, worker threads are launched that map the provided inputs to the names of inputs in the encoded function, then create a temp file for z3 to use as input, launch z3 and gather results.

Results are written in parallel by threads as they finish to a ConcurrentHashMap. Once all encoded functions have had threads launched for each of the I/O pairs, or the system has timed out, the system waits for all threads to finish or time to expire. If time expires and tasks continue to run, they are interrupted and shutdown completes. The database is queried for the source code of all satisfactory results. This source code is passed back to the servlet, which creates a web page of results to display.

Data Store Module



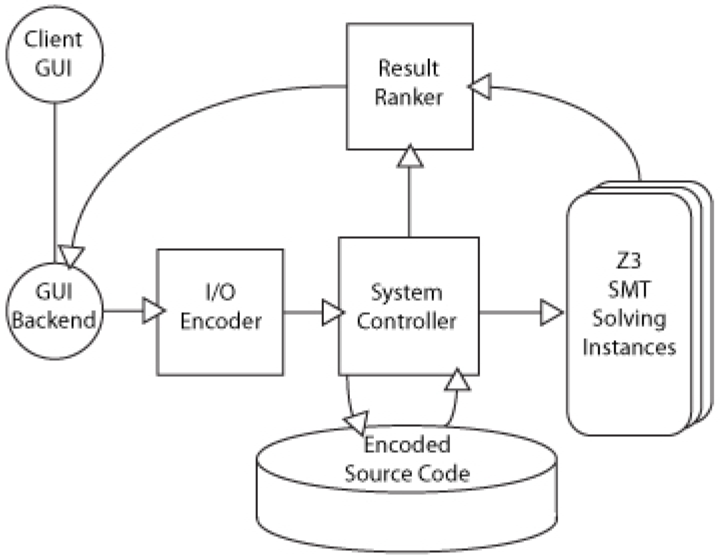
The database has an encoded function and source code table for each of the four possible function outputs: booleans, strings, chars and ints. A DatabaseAdapter class creates a facade between the SystemController and the database that abstracts the fetching of a streaming ResultSet of encoded functions based on the desired I/O signature, and also abstracts the fetching of source code based on the relevant function key.

System Analysis

This system is operating on Amazon Web Services (AWS). At this point in time only the free tier of AWS is being used. The free tier provides a limited amount of resources without cost. More resources can be obtained for a cost if needed. By using AWS, Satsy will have a flexible amount of computational power which will automatically expand to be able to accommodate as many instances of Z3 as needed.

The web server's operating system is Ubuntu Server 12.04 LTS [4]. It is running Tomcat 7 [5] to instantiate servlets in a Java 1.6 Java Virtual Machine (JVM). The web server is also running MySQL Server 5.5 [6] which contains a database that stores all of the encoded paths and corresponding source code for the functions to search through. The various Java modules will interact with the database through a class which uses Java DataBase Connection (JDBC) [7] to connect to the database and to run sql statements on the database. The user interface has been developed using Twitter's Bootstrap framework [8] as our front-end framework as well as additional JavaScript on .jsp pages. The front end will pass the search parameters to the search logic and receive the search results to display to the user.

System block diagram

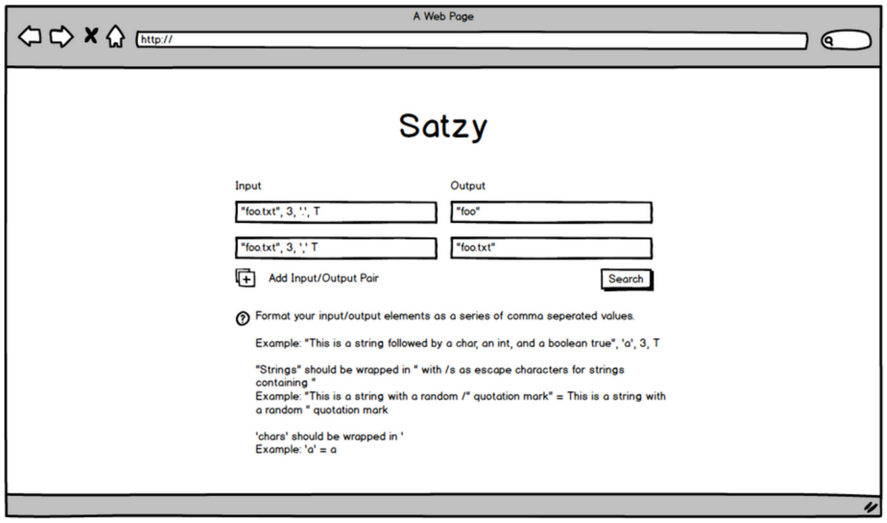


Detailed Description

Interface Specifications

The user interface will be a website with three primary pages:

1. Search Page: users will enter input/output strings representative of the desired behavior. Results will be returned when the user hits search.



"foo.txt", 3, '.', T — "foo"
"foo.txt", 3, ',' T — "foo.txt"

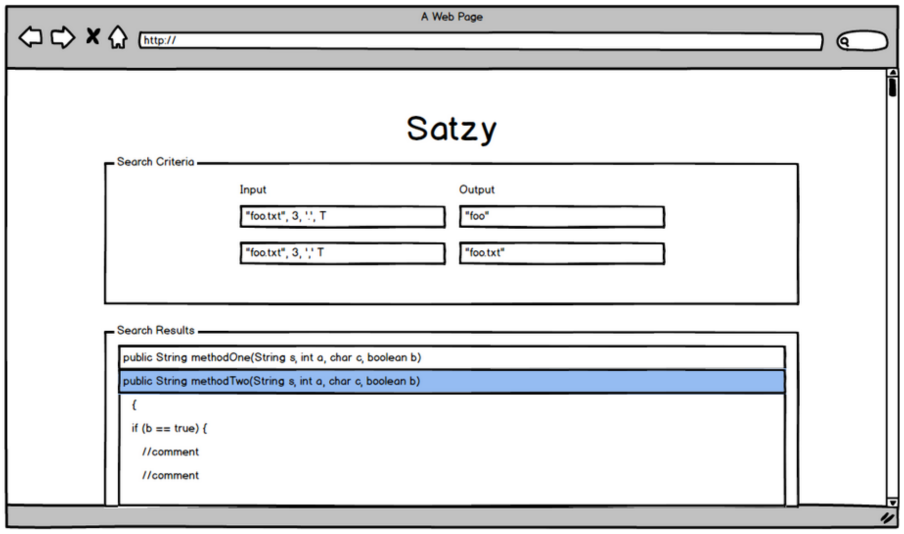
Format your input/output elements as a series of comma seperated values.

Example: "This is a string followed by a char, an int, and a boolean true", 'a', 3, T

"Strings" should be wrapped in " with /s as escape characters for strings containing "
Example: "This is a string with a random /" quotation mark" = This is a string with a random " quotation mark

'chars' should be wrapped in '
Example: 'a' = a

2. Results Page: the users query will remain visible and, if modified, will allow the user to requery. Results will be listed as method signatures, with a preview of the code available. Clicking will take a user to the full details for a particular method.

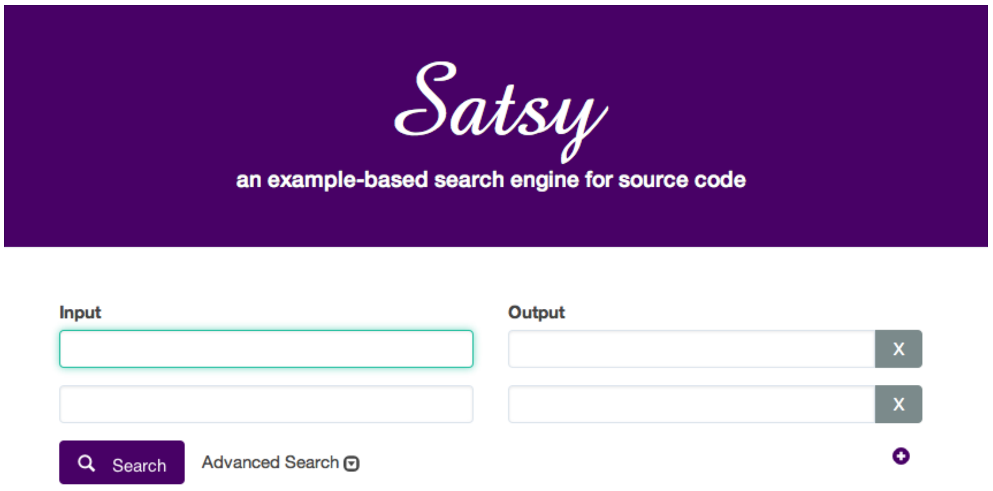


"foo.txt", 3, '.', T — "foo"
"foo.txt", 3, ',' T — "foo.txt"

```
public String methodOne(String s, int a, char c, boolean b)
public String methodTwo(String s, int a, char c, boolean b)
 {
 if (b == true) {
   //comment
   //comment
```

3. Code Details Page: This page will show the details about a matching method including the method signature, the full method body, and any other information relevant to the method and search.

Below is an image of the current Satsy search page:



Hardware/Software Specifications

- **Hardware Specifications**
    - Amazon Web Services EC2 Free Tier Micro Instance
        i. 613 MiB Memory
        ii. 2 EC2 Compute Units
        iii. 30 GB Storage
        iv. Ubuntu Server 12.04 LTS
- **Software Specifications**
    - Apache Tomcat 7.0.26
    - Z3 High-Performance Theorem Prover 4.3
    - MySQL Server 5.5.32
    - Java 1.6

Validation and Verification

I. Validation:

Validation answers the question of “did we build the right thing?” as well as making sure that the product satisfies the client. To ensure that we are building the correct project, we will have both requirement and design reviews.

A. Requirement review:

1. The client will look over the product at its current stage to confirm that it meets the requirements as expected. If not, adjustments can be made in the next iteration.
2. To ensure the everyone is on the same page for all of the requirements, we will talk over what is planned for the next iteration. If there is any

confusion for a given requirement, we will discuss with the client until the issue has been clarified.

B. Design review
    1. For every iteration, we will meet with our client to have her look over the product at its current stage in production. This will be to get feedback on if the changes we have made during the last iteration are going in the correct direction.
    2. At the end of every iteration, each member of the team will present a brief summary of their work to update the entire team.

II. Verification:

Verification answers the question “ did we build the product right?” as well as testing the product for bugs in the system.

A. Code Review
    1. For every commit of the code, changes should be reviewed and approved by at least one other team member before it is pushed to the server.
    2. To help with the review and understanding of the code, the code should be well commented, including having Javadocs for each function.

B. Testing
    1. JUnit white box testing
        a) Unit tests will test at least 50% of the critical code branches
        b) Unit tests will test at least 25% of the non-critical branches
    2. JUnit black box testing
        a) Unit tests will test that the functions return the expected output for the given inputs.
    3. Regression testing after each iteration.

We will be using several types of testing for the validation and verification of Satsy including:

- Code reviews: Team members look over others’ code. If they see something that they feel should be changed, notify the writer of that section of the code so they can take appropriate action if they deem it necessary.
- JUnit testing: By using JUnit testing classes to test all Java code, we do both white and black box testing. For white box testing, we test based on the code that we know is in the modules. These JUnit tests would make sure to test each path through each function. For black box testing, we test based on the API of the modules. These JUnit tests would make sure the function returns the expected output for the given inputs.

Implementation Issues and Challenges

- **Learning how to use Amazon Web Services (AWS)**

  Amazon Web Services was used to setup a web server which allows clients to interface with the Satsy system. Amazon Web Services provides free-tiered versions of their Elastic Compute Cloud (EC2), which can be used to run micro instances with server-oriented operating systems. After figuring out how to signup for and set up a micro instance, it is easy to install a web server which provides the interface necessary for clients to interface with the Satsy system. The biggest challenge with setting up an EC2 free-tiered micro instance is digging through Amazon's convoluted trove of AWS documentation to find out how to setup and use an EC2 micro instance. Once this hurdle is overcome and an EC2 micro instance is set up and working, everything becomes smooth sailing.

- **Configuring Apache Tomcat to run on port 80**
  Apache's Tomcat web server software runs on port 8080 by default. The standard port for serving content via HTTP is 80. Changing Tomcat's port to 80 is as simple as changing an XML configuration file. What can be challenging, though, is when another web server is already using port 80. An example is when using Apache's standard HTTP web server along with Tomcat. This can cause configuration headaches trying to get the two to work together. Fortunately, Satsy's web server is only running Tomcat and there is no need to get two different web servers working nicely together.

- **Configuring development environment using Eclipse and Tomcat**
  The easiest way to set up a development environment which allows Eclipse and Tomcat to work well together is to use Eclipse's latest version of their IDE for Java EE Developers (Kepler Service Release 1). This version of Eclipse comes with the Apache Tomcat 7.0 adapter installed. Installing this adapter with older versions of Eclipse can be cumbersome. With the Apache Tomcat 7.0 adapter already installed the only other setup needed is to let Eclipse know where Tomcat's binary directory is located. Once Eclipse knows the location of Tomcat, server startup and web application execution are as simple as pressing a "go" button.

- **Setting up a Bitbucket repository to allow collaborative development**
  Working in a collaborative development environment sometimes requires a central repository for storing and maintaining project code and files. Bitbucket is a service that allows users to store and access Git repositories. One challenge is figuring out how to get past Bitbucket's maximum allowed repository collaborators, which is set at 5. Fortunately, Bitbucket allows users with educational email addresses (ending with .edu) to have an unlimited number of collaborators.

- **Getting JDBC to work with MySQL**
  JDBC (Java Database Connectivity) can be used to allow Java code to communicate with and use MySQL. It can be challenging figuring out what version of the JDBC driver is necessary for certain projects and how to integrate them with Java code.

- **Converting Redis database to MySQL database**
  Redis is an in-memory key-value data store, which was being used in Dr. Stolee's prototype to store encoded methods and source code. Because the product needs to serve multiple clients, handling concurrent requests effectively, MySQL was chosen as

the data store technology for the project. Some effort was required to transition from using Redis to using MySQL to store encoded methods and source code. These challenges included learning about Redis, redesigning the schema, and tracing through prototype code so that the existing encoded methods and source code could be exported into the new MySQL data store.

References

[1] http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r12.09.09.pdf, http://www.smtlib.org/
[2] http://z3.codeplex.com/
[3] http://aws.amazon.com/ec2/
[4] http://releases.ubuntu.com/precise/
[5] http://tomcat.apache.org/tomcat-7.0-doc/
[6] http://www.mysql.com/
[7] http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[8] http://getbootstrap.com/2.3.2/