SALAMANDAR- A NLP AUTOMATED UNIT TESTING SYSTEM  
Built over Cucumber and Selenium

Requirements and Design Documentation

Walter B Gress V  
Wesley Feltz

ABSTRACT

SALAMANDAR is a Natural Language Processing interface to Cucumber and Selenium. It is designed to parse Gherkin plain text from Cucumber and generate unit tests in an automated fashion.It does this by identifying matching keywords in Gherkin text to code segments. It also matches variable names from Gherkin text to code and operation names to code. The core of SALAMANDAR is a NLP machine learning algorithm,.Conditional Random Fields (CRF) that maps a corpus of phrases to operations over a standard java class. In order to develop the SALAMANDAR library we shallll take sample pieces of code with Gherkin phrases and train a machine learning database which will be packaged along with the SALAMANDAR libraries.

REQUIREMENTS

Note: In Parsing Operations (I), parsing Gherkin text into…. Is meant to parse CRF text into the respective parameter of the sentence.

I.Parsing Operations

1. The system shall parse Gherkin text into keywords
2. The system shall parse Gherkin text into operations
3. The system shall parse Gherkin text into variable names
4. The system shall parse Gherkin text into function names
5. The system shall parse Gherkin text into control structures
6. The system shall parse Gherkin text into class names
7. The system shall parse Gherkin text into constant numbers
8. The system shall parse Gherkin text into constant characters
9. The system shall parse Gherkin text into objects
10. The system shall parse Gherkin text into arrays
11. The system shall parse Gherkin text into generics
12. The system shall parse Gherkin text into try….catch statements
13. The system shall parse Gherkin text into interfaces
14. The system shall parse the miscellaneous text using CRF

II. Source Code Considerations

1. The system shall use the Selenium framework as an automated unit testing system
2. The system shall use Cucumber in cooperation with Selenium to automate testing of web applications
3. The system shall take use Cucumber to receive plain text instructions
4. The system shall incorporate Selenium WebDriver into its testing code
5. The system shall use the Conditional Random FIelds library CRF++ for Java on Github
6. The system shall use CRF to convert Gherkin plaintext into source code
7. The system shall write Gherkin code in the Gherkin source code conventions

III. CRF Considerations

1. The CRF maps sentences to common actions
2. Common actions include such things as “Credit card number too short”
3. Common actions include such things as “I enter a card number that is less than 16 digits long”

IV. Use Case

1. The system reads in a sentence of Gherkin text
2. The system parses the Gherkin text using CRF into intermediate code
3. The system identifies keywords, etc. as defined by (I) in the CRF
4. The system generates (I) with return of CRF function over Gherkin text
5. The system completes operable code from (IV.4)

V. Example

1. The system parses the following phrases:
   1. **Background True for all Scenarios Below**
   2. **Given** I have chosen an item to buy,
   3. **And** I am about to enter my credit card number
   4. **Scenario** − Credit card number too short
   5. **Scenario Definition**
   6. **When** I enter a card number that is less than 16 digits long
   7. **And** all the other details are correct
   8. **And** I submit the form
   9. **Steps**
   10. **Then** the form should be redisplayed
   11. **And** I should see a message advising me of the correct number of digits