Inform 7 Usability Precompiler

A Senior Project

presented to

the Faculty of the Computer Engineering & Software Engineering

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Chad Robert Brantley, Timothy Phan

June 2013

Abstract
Inform 7 Usability Precompiler or I7UP is a project which helps Inform 7 authors with their stories by diversifying the possibilities in the author's interactive fiction. I7UP reads and analyzes the Inform story that is passed in through python and natural language processing then provides the user with a selection of choices to add to his story and lastly automatically generates the corresponding Inform 7 code. By utilizing the combination of Python, Django, and HTML/CSS, I7UP is able to provide an easy-to-use interface which helps generate additional "Understand" statements and "Conjugated Verb" statements. I7UP is an authoring aid for Inform 7 operators. I7UP will simply add on to the story and allow the author the choice of statements to add on. This project is extensible since it allows support for more features and can further aid Inform authors with their interactive fiction.

1. Introduction

The main goal of this project is improve the experience of Inform 7 authors. Inform 7 is an interactive fiction authoring system based on natural language to help users write interactive fiction therefore the code essentially reads like English rather than other coding languages. Inform 7 uses a declarative rule-based programming style and is capable of inferring types and properties of objects based on how they are used. Inform 7 is also able to track and support relations between objects while allowing the author to create their own relations.

To enhance the authorship experience of Inform 7 designers we wanted to provide a selection choice of additional changes the author can make to his/her story while not limiting their freedom and altering the story. Also, we wanted to make the project simple to use because a lot of the Inform authors may not have any real coding background and we did not want to overwhelm or confuse them. By using this project, the author can focus more on their interactive fiction and then use our project to help generate additional Understand or Conjugated Verbs statements that they have missed or have not considered. By generating additional Understand statements, it will help create much more possibilities within the author's story by extending the range of commands a user is capable of using to do a certain task in the story. Since the range of commands is extended, the user would not have to constantly try to find the correct commands or verbs to use during their playthrough and allow them to enjoy the story more. The generated Conjugated Verb statements help catch all possible usages of the verb which allows the author easily relate objects.

2. Requirements

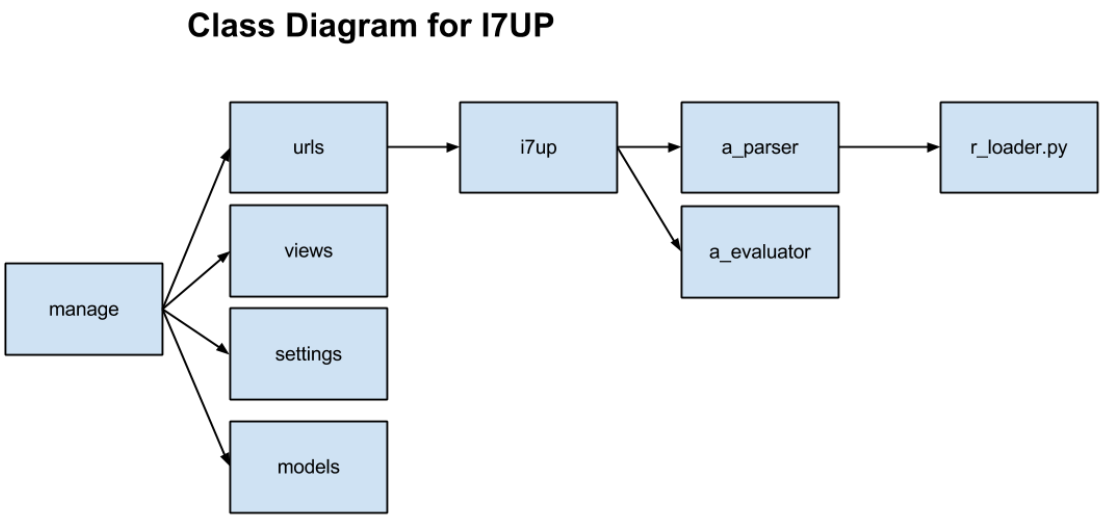
We gathered our requirements by a week-to-week correspondence spanning across 2 quarters with Dr. Foaad Khosmood. During those meetings, we discussed the current direction of the project and current roadblocks. Dr. Khosmood would suggest certain features he would like to see in our project and then we would complete one feature to a satisfactory standard and then we would then go on to complete the next feature. After our meetings with our advisor, we would then meet and talk about how we would implement that feature and come up with possible solutions to show our advisor for the next meeting. Then, after getting suggestions and verification from our advisor, we would continue our development and further polish our implementation.

Our implementation had some basic requirements which had to be maintained and were constantly referenced while we were developing. Our main requirement was to have the project lift some authorial burden to allow authors to focus more on their interactive fiction. We also had to provide an easy-to-use interface because not all Inform 7 authors are fluent in coding. We then came up with the idea of making our project web based that way the authors simply need to paste in their Inform 7 code and copy the resulting generated Inform 7 code. Another requirement was to make the project non-intrusive since we wanted to allow authors to choose what additions they can add to their story.

3. Design

The I7UP system was originally designed for a console interface, in which the user would specify a local file as input, which produced a human-readable interim file. From there, the interim, or “annotated” file would be edited, and the script would be run again. This structure changed, however, since we decided that the installation of Python and all supported libraries would be a bar for anyone actually wanting to use the application; the main users would be authors, who are not necessarily technologically inclined enough to navigate a command line interface. As such, we designed a new system: this new system has a class structure that looks like Figure 1.

Figure 1



As the diagram illustrates, class dependencies were few: only two classes were used, and these were interchangeable with tuples and lists, since they only held data. The web server takes up the first two columns. "manage" is the main driver for the webserver; from there, "urls" is used to parse the received requests and direct it the the appropriate view name. "Views" simply acts as a generator for the html page: it fills in template arguments so the html-templates know what is what. "settings" holds basic properties, such as what database to use as a store for the session. "Models" are the database models that represent any logged data; our current system makes no use of models yet, but a module could be added to log and calculate statistics about generated results: what words are commonly conjugated, and average number of changes made.

Originally, we were planning on using classes to assist in the parsing and evaluation stages; however, we decided against it, since these classes were basically data-holders, and had no associated functions. Instead, we represented most data segments as tuples and lists, to be more Python-ic in our coding.

Figure 2

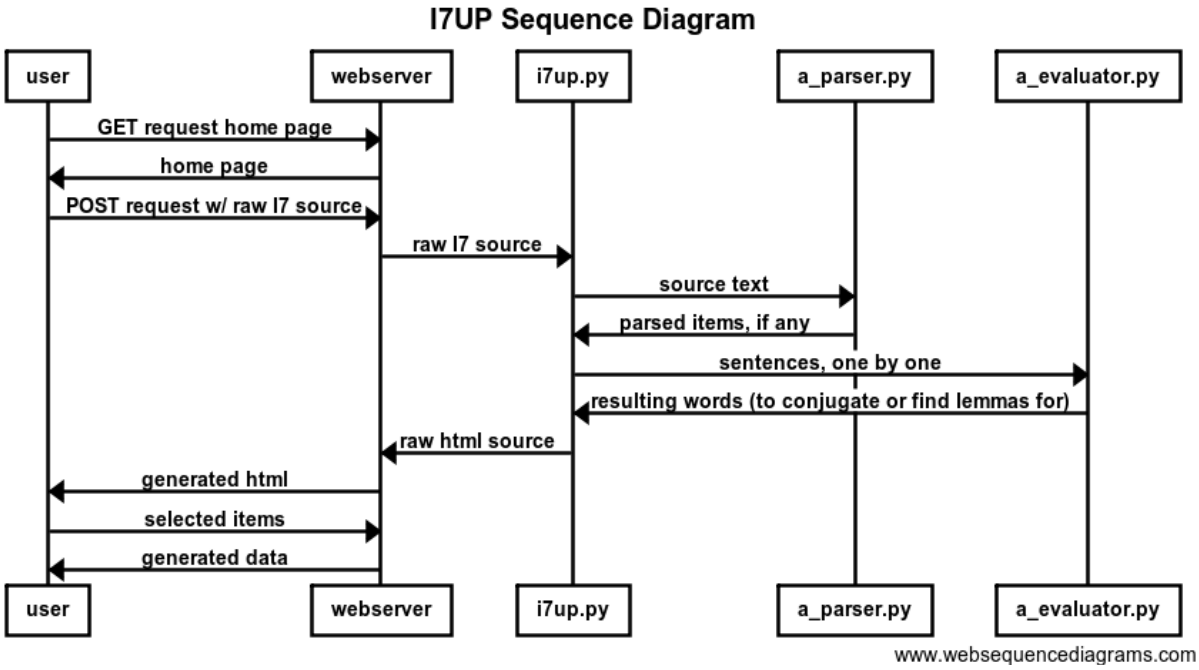


Figure 2 has been simplified on the web server and user ends to make the diagram readable; the management of requests, for instance, is spread across 4 different Python files.

In standard usage, the client requests the home page, which the web server delivers: from there, the client inputs their source Inform 7 code. That code is then parsed, by the parser, and the results are evaluated for synonyms and resulting conjugations, by the evaluator. The parser uses the Natural Language Toolkit (NLTK) which is an open source python library for natural language processing. We use the NLTK to tokenize the Inform 7 code and tag each word, then we grab each sentence based on predetermined formats to pass into our evaluator. The evaluator then uses WordNet found in the NLTK to evaluate the synonyms and lemmas of the target word in the sentence. Our main controller, i7up, acted as the link between the web-interface world and the parser and evaluator modules: it used the data it obtained to stock the html full of values, which the user could see and select from.

For example, the conjugated forms of a verb are hidden as an HTML attribute of the button object; when pressed, Javascript/jQuery go to work enabling the button and replacing the original text with the conjugated forms. This turns out to be intuitive to the user from a UI perspective, and simplifies the server, as opposed to a server that performs that responds to a request each time something is clicked.

4. Testing

Testing was done with a variety of system and unit tests, and compiled into a basic Ant script to run them all automatically; results were then "diff"d against expected results to find flaws in our system. Here, the console interface became crucial to maintaining the code, since a GUI system complicates testing overall. Also, we would take in multiple Inform 7 stories of varying length, run them through our project, and then compile them through Inform 7 to figure out any compiler errors or issues found in our resulting stories.

Web server integration testing was all done manually, since most of the bugs on that front were syntactic or graphical, and not computational. Had any significant calculation been done with the Javascript, we would have produced a qUnit script to test the actions; however, all Javascript and jQuery involved was simple replacements of values and such.

Another method of testing was done by playing through the interactive fiction and then comparing between the original and the generated version. By playing through both versions and gathering feedback, we able to gauge how beneficial our project and figure out what areas need work.

5. Problems Encountered

One of our earliest problems was how to distribute the code: we took some extra time deciding between releasing the raw script, packaging it with Python in a distribution, or running on a web server. The web server solution seemed the most intuitive; however, none of us had any strong experience doing this in Python. Nonetheless, we succeeded in finding Django and learning how to use it.

The next issue we had arose from our choice of web server: originally, we planned on using Google App Engine; however, we quickly found that many of the natural language processing libraries our console application was running on was not supported by Google App Engine: as such, we turned to CADRC for a server, so that we might have more freedom in libraries to use.

6. Results

Our project manage to accomplish the goals that we set and met with the requirements that were appointed by our advisor. We were able to take in the author's interactive fiction and produce a generated text which contains additional Understand statements and Conjugated Verb statements based on the author's selection. In order to do so, we provided an easy-to-use web interface which allows the author to paste in their text and allow them to choose which statements they wanted to add and then they can simply copy the generated text.

7. Future Work

This project has plenty of room for additional features and improvement. For example, pluralizing the generated words to match the targeted word or looking at the hyponyms and hypernyms of the targeted word to generate more possibilities. Another possibility would be to actually alter the author's story by randomizing certain words to help create a different experience each time it is played through.

As the quarter comes to an end, we wanted to polish what we have and release a stable build of our project instead of having several half-finished features which are prone to errors. The next person or group who carries on this project should refer to the user manual found in the appendix and also in our github which is open to the public. Our recommended method of carrying on this project would be to create a separate python module for a new feature and then integrate it with the web framework, that way each feature has its own module and can be easily debugged and worked on.

8. Appendices

8.1 Tools

- Github
  - https://github.com/cbrantley91/i7up
- Python 3
  - http://www.python.org/
- Natural Language Toolkit/Wordnet
  - http://nltk.org/
- Django 1.3
  - https://www.djangoproject.com/
- Nodebox
  - http://nodebox.net/code/index.php/Linguistics
- Inform 7
  - http://inform7.com/
- HTML/CSS
- Javascript/jQuery

8.2 User Manual for Software Developers

This is a simple Python webserver to dynamically generate usability enhancements to existing Inform 7 code. If you are unfamiliar with Inform 7, it is a design-system for text-adventures that uses (somewhat natural) English syntax to generate a world.

Our program parses that input code and generates an html page with possible synonyms (to select from) for nouns/verbs so that the user doesn't have to go "verb-hunting". For example, if the user were to create the object using "The bathtub is in the Bathroom.", I7UP will allow the user to selectively append:

- Understand "bath" as bathtub.
- Understand "tub" as bathtub.
- Understand "bathing tub" as bathtub.

using standard HTML tools (buttons/checkboxes)

Also, if the user has defined any actions, it will conjugate those actions; the phrase:

- The verb to draw is an action applying to a picture.

gets expanded to cover all bases:

- The verb to draw (you draw, he draws, she draws, they draw) is an action applying to a picture.

The code is a bit messy right now, but I will do my best to clean it up before I set this project to rest. Currently, the code is only deployed on a private server with no outside accessibility, so you'll have to run it using:

- python manage.py runserver

while in the mysite directory

Required installs to run:

- Python 2.x or 3.x
- Django 1.3
- NLTK (Natural Language Toolkit)
- (additionally, Nodebox is used, however, we included the source of that with the distribution)