The FreeCoffee Editor: Using Natural Language Sentence Structure to Make Blocks More Readable

Robert Holwerda
Academy for Informatics and Communication
HAN University of Applied Science
Arnhem, The Netherlands
Robert.Holwerda@han.nl

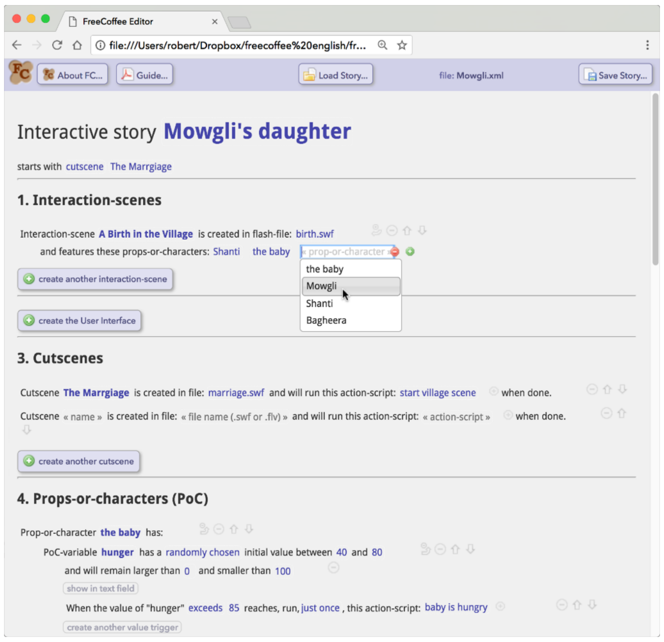
***Abstract*— One aspect that increases the accessibility of many blocks-based languages is their use of labels to tell users the meaning of the input-slots in the blocks. In this regard, every block is a little form. FreeCoffee is a blocks-based language whose editor goes well beyond terse slot labels and communicates the meaning of a block using complete grammatical sentences. These sentence-blocks still contain input-fields and drop-down menus for user-input, and will, in response to user input, adapt both their structure and their wording to keep the sentence grammatically correct. Sentences structure can also be changed in order to add optional features to the block. This sentence-oriented blocks-editor was designed for a domain specific language used in a multimedia design course. The abstractions and semantics of this language are far removed from mainstream scripting languages, and some of its structures were complex. Still, the language interface helped to enable students with almost no training to create interactive multimedia productions.**

I. Introduction

FreeCoffee started out as a domain specific language for interactive multimedia documentaries, to support a bachelor course in interactive multimedia design. It came with an editor consisting of a set of forms, built using Microsoft InfoPath, and an interpreter written for Adobe Flash. A user would create the graphics, sounds, and animations in Flash, and add the interpreter to her Flash document. She used the FreeCoffee forms to declare scenes, characters, transitions, interactions, and events, referring to the multimedia assets in the Flash document. When executing the Flash-application, the interpreter would read these forms to make the interaction come alive. The form-based editor, however, was difficult to use for students of multimedia design, and a new editor was developed, see Fig. 1 and Fig. 2. This second editor is the editor described in this demo summary. Readability was a major design goal, in order to (1) allow new design-students to understand the semantics of the DSL very quickly, and (2) enable their design teachers to read their code without much study.

II. Blocks with full sentence structure

Compared to text-based languages, in blocks-based based languages the verbosity of extensive labeling is much less of an annoyance since the user does not have to type everything that the reader gets to read. But short labels can still fail to communicate much about the concepts, the mechanics, and the relationships in a rich, high-level DSL.



Interactive story **Mowgli's daughter**

starts with cutscene The Marrgiage

1. Interaction-scenes

Interaction-scene **A Birth in the Village** is created in flash-file: birth.swf
and features these props-or-characters: Shanti the baby « prop-or-character »

3. Cutscenes

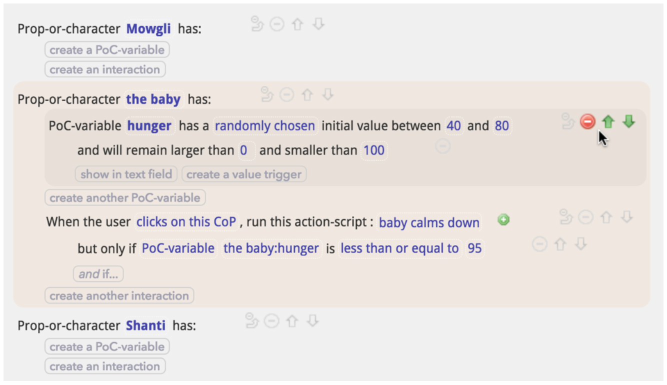
Cutscene **The Marrgiage** is created in file: marriage.swf and will run this action-script: start village scene when done.

Cutscene « name » is created in file: « file name (.swf or .flv) » and will run this action-script: « action-script » when done.

4. Props-or-characters (PoC)

Prop-or-character **the baby** has:
PoC-variable **hunger** has a randomly chosen initial value between 40 and 80
and will remain larger than 0 and smaller than 100
When the value of "hunger" exceeds 85 reaches, run, just once , this action-script: baby is hungry

Fig. 1. A screenshot of (part of) the FreeCoffee editor. Text in blue is input by the user, text in black are labels provided by the editor. Some blue input fields are free input fields (like the story title), while others are drop-down menus. Empty fields are indicated by a gray placeholder between « » marks (e.g. in the second cutscene declaration).



Prop-or-character **Mowgli** has:

Prop-or-character **the baby** has:
PoC-variable **hunger** has a randomly chosen initial value between 40 and 80
and will remain larger than 0 and smaller than 100
When the user clicks on this CoP , run this action-script : baby calms down
but only if PoC-variable the baby:hunger is less than or equal to 95

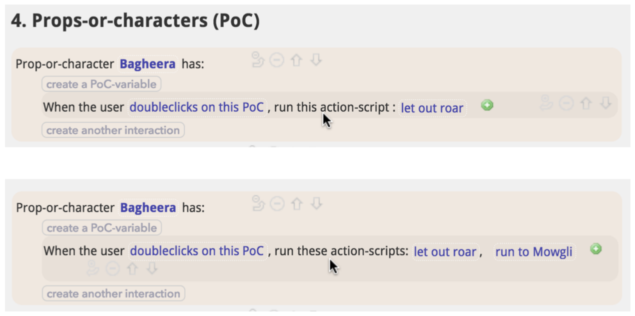
Prop-or-character **Shanti** has:

Fig. 2. The block-structure becomes visible when the mouse hovers over blocks (in this case: a variable-block nested inside a Prop-or-character block.) Icons to manipulate the block become available when the mouse is near.

For the second editor for FreeCoffee, we created a blocks-based language in which each block presents a full natural language sentence. For example: Story-variable *«name»* has a *fixed* initial value of *«value»*, with *«name»* and *«value»* being text input fields, and *fixed* being an option in a drop-down menu. (FreeCoffee was in Dutch; these examples are translations). In contrast to some block-based languages, these sentence-blocks can change their structure, depending on user-input. If in the example above, the user changes *fixed* into *randomly chosen*, the tail of the sentence changes to allow the user to input the range from which the value is chosen: Story-variable *«name»* has a *randomly chosen* initial value between *«min»* and *«max»*. For some sentence-blocks, extra features/options can be enabled that add subordinate clauses to the sentence. So, the story-variable sentence-block can be extended to read, for example: Story-variable **cities to visit** has a *fixed* initial value of *6*, and will be displayed in the text field *cities_left*. Any diffuseness, caused by the resulting complexity of the sentence, is mitigated by using layout and typography, both to indicate the structure, and to make the sentence scannable for readers.

The editor was designed with keyboard manipulation in mind. Small, unobtrusive buttons for manipulating blocks (deleting, moving, extending) are inlined after the sentence, and a user who uses the tab-key to move from input-slot to input-slot is also able to 'tab' her way to these buttons and activate them from the keyboard. To prevent confusion, any input field or menu that might cause a change in sentence structure will only change the part of the sentence that comes *after* the input field or menu. The sentences are designed in such a way that a user who fills out the sentence from left to right will not have to retreat to the left to deal with changes to the sentence structure there. She can proceed to fill out or extend the sentence in the rightward direction.

Another linguistic aspect is that words will be inflected, depending on user input, to keep the sentence grammatically correct. When, for example, a second element is added to a list-input, the right verbs and nouns are pluralized (Fig. 3).



4. Props-or-characters (PoC)

Prop-or-character **Bagheera** has:
create a PoC-variable
When the user doubleclicks on this PoC , run this action-script : let out roar
create another interaction

Prop-or-character **Bagheera** has:
create a PoC-variable
When the user doubleclicks on this PoC , run these action-scripts: let out roar , run to Mowgli
create another interaction

Fig. 3. The label for the list of action-scripts is inflected, depending on the number of items in the list: “this action-script” becomes “these acrionscripts”.

III. User Experience

This resulted in a language-UI that was effective at onboarding new users who had no experience with programming and received very little training in the use of FreeCoffee. At least 20 interactive stories, documentaries, and training applications were created by student teams over the course of three years. No systematic user evaluation was performed. User feedback consisted almost entirely of functionality requests for the DSL. Learnability and readability were deemed effective by the multimedia design teachers who supported the students. Two serious complaints were raised against the FreeCoffee editor itself: (1) its performance would decrease when the editor contained many hundreds or thousands of blocks, and (2) it did not support simultaneous editing by multiple users, which hindered teamwork.

There are some indications that students did not regard working with FreeCoffee as a form of programming. Some used the term 'data entry' instead, which seems a bit pedestrian on first reflection. But one might, more optimistically, interpret this as a sign that these users felt little or no cognitive challenge in conveying their design ideas to FreeCoffee.

FreeCoffee is named after the reimbursement negotiated by the student-team that created the original DSL.