A DSL for executable 'how to' manuals

Marcel Heinz Philipp Helsper Ralf Lämmel Tobias M. Schmidt
Software Languages Team
University of Koblenz-Landau, Germany
http://softlang.wikidot.com/

ABSTRACT

'How to' manuals help potential users with deploying and understanding software technologies such as web applications or servers. In a domain analysis, we survey existing 'how to' manuals to assess the feasibility of making the manuals executable and to derive a suggestion for domain-specific language support. We realize a DSL for executable 'how to' manuals and refer to the approach as 'literate deployment scripting', as it is inspired by literate programming in that all the code for deployment and configuration is embedded into its documentation. This includes code (or commands) for an initial deployment or changes to a deployed software system.

CCS Concepts

•Software and its engineering → Domain specific languages; *Command and control languages;* **•Social and professional topics** → Implementation management; Software management; File systems management;

Keywords

Literate programming; Software deployment; Install scripts; Executable How To; Executable manual; Literate deployment

1. INTRODUCTION

Software manuals explain how to set up, configure, and use software technologies. These manuals differ in authorship, quality, level of detail, and others [6]. Completeness is critical, as a reader relies on the manual to set up a working system by following the steps in the manual. In this paper, we focus on 'how to' manuals as step-by-step guides for software deployment of web applications or servers etc.

We perform a systematic survey of 'how to' manuals, which delivers the set of concepts that are to be supported by our domain-specific language (DSL) for executable 'how to' manuals, called *LDS* (for *Literate Deployment Scripting*). Most notably, LDS features executable activities such as system configuration scripts and file manipulation. In analogy to literate programming [5] these activities are integrated with their explanations. For a deeper understanding, a stepwise half-automated execution is leveraged as opposed to full-fledged deployment tools, e.g., SoftwareDock [1]. LDS manuals are made for those who want to leverage controlled stepwise execution as means of comprehending a system [4]. At this point, LDS manuals are only made for linear flows.

Based on our realization of executable LDS manuals as an HTML dialect, we measure the effort it takes to turn a conventional manual into a LDS manual. The implementation, data and artifacts related to the domain analysis and effort measurement, and sample LDS manuals are available online from the paper's website.[1]

SAC 2016,April 04-08, 2016, Pisa, Italy

http://dx.doi.org/10.1145/2851613.2851957

2. ILLUSTRATION

'How to' manuals mix natural language content with software language content. Fig. 1 shows an interactive LDS GUI. Panels for selecting an LDS manual and a workspace can be found at the upper left corner. Below them, one can find an integrated webview. To the right of the webview a list of selectable ids for each of the LDS manual's execution steps is presented. These can be started with the buttons below the list. Additionally, the preprocessor adds clickable buttons to each executable step in the original HTML file.

The webview presents an LDS manual for setting up a web application with the Django framework.[2] There, the first block concerns a *command* for execution of a Python script. Next, the second block presents Python *code* that should be inserted into a file of the emerging web application; the relative path '`polls/models.py`' to the project's file is also specified. Hidden DSL annotations express these execution semantics as shown below:

```
<p>At the command line, you run this command</p>
<div style="..."><pre>
<!--LDS BeginRunScript id="Create an app" executor="sh"-->
cd src/mysite
python manage.py startapp polls
<!--LDS EndRunScript--></pre>
</div>
```

[1] http://softlang.uni-koblenz.de/lds/

[2] We adapted the actual manual https://docs.djangoproject.com/en/1.8/intro/tutorial01/ for clarity.

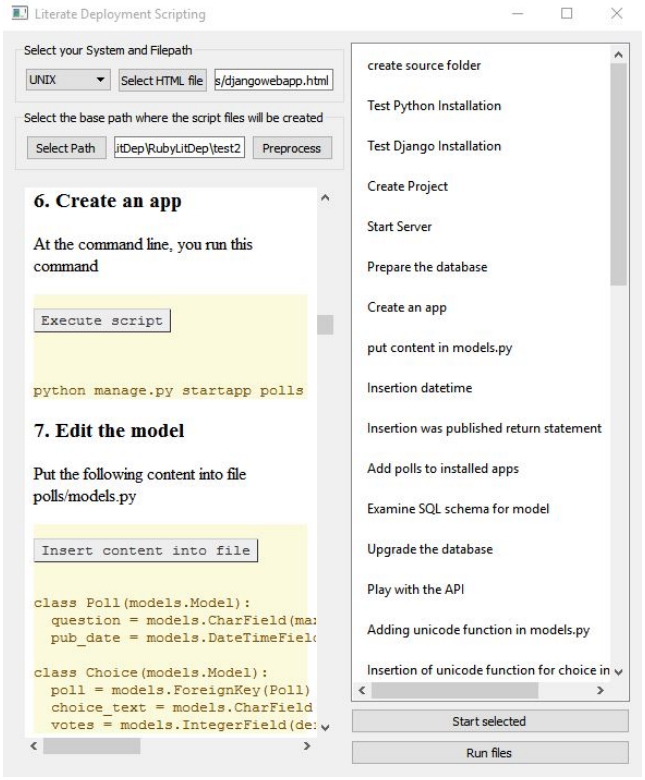

Figure 1: An implemented GUI supporting the execution of LDS manuals.

That is, LDS commands are embedded as HTML comments. There is a 'run script' command which is delimited by `BeginRunScript` and `EndRunScript`, whose actual code body is enclosed into a `pre`formatted block. The execution of the script is delegated to the command-line shell `sh`.

Because conventional manuals are not executable, they may lack information. Thus, users scan manuals for helpful information and they determine a stepwise process by trial and error. These are the limitations that LDS addresses; we adopt the common idea to make yet another natural language-based notation executable [2]. We emphasize understandability and transparency supported by the possibility for a step-wise execution instead of a full automation, since LDS manuals are supposed to help the user gain a deep understanding of what is happening in the deployment process. There, the output of each step can be observed by the user. As a result, troubleshooting can be simplified. Multiple steps can be executed at once.

3. DOMAIN ANALYSIS

We follow the method of Harsu [3] in that we specify the *scope* with inclusion and exclusion criteria and motivate design decisions based on a *commonality analysis* on existing 'how to' manuals.

We limit our scope deliberately for this work and aim at complete, simple, and transparent 'how to' manuals to be executed on a given machine without management of dependencies, building, and configuration. That is, we aim at the execution of a script which sets up a system in a linear flow. Interactions between script and user would be limited

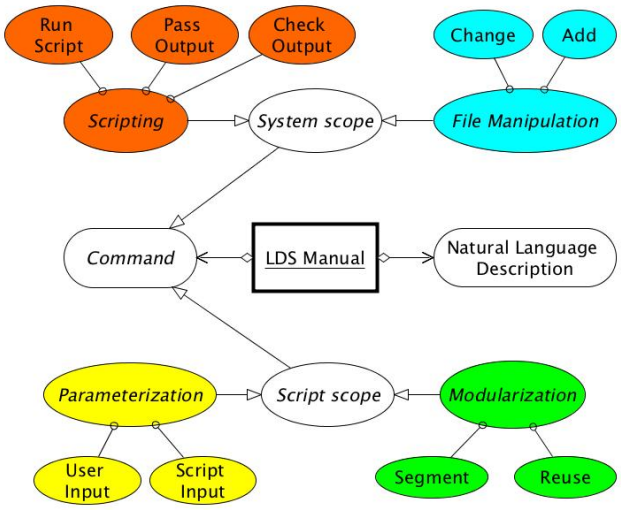

Figure 2: Domain concepts with grouping

to entering parameters for the sake of transparency.

In our systematic survey, we considered GitHub's *showcases* 'Projects with great wikis'[3] and 'NoSQL databases'[4]. Additionally we analyzed projects corresponding to popular Ruby on Rails applications[5]. We examined readmes and wikis of the selected projects to see whether they provide or point to deployment-related information. The resulting corpus and inclusion/exclusion choices are documented at the paper's web page.

3.1 Commonality analysis

We examined natural and software language content of the manuals guided by the following questions. What is the nature of a given content block: a) code to be executed; b) input for an activity; c) output of an activity; d) code for file manipulation? By examining all such blocks, we also encountered the use of variables, such as a placeholder for a user name. Thus, we added the following question: Does a given block of content specify a variable?

The concepts in the leaf nodes in Fig. 2 were used for tagging all manuals of the survey. All concepts can be clustered in groups. One group concerns *scripting*, which includes running code at a command-line (*Run script*), additional dataflow (*Pass output*) and straightforward automatizable constraint checking (*Check output*). Another group deals with *file manipulation*, including file changes *Change* and the addition of file content *Add*. Other file operations are conceivable, but we did not encounter them in the survey. *Parameterization* forms an auxiliary group of concepts including requested *user input* or *script input*, where script input is already covered by the tag *pass output*. The second auxiliary group encourages *modularization* of software language content, including separation of code blocks in *segment*s and enabling *reuse*.

Fig. 3 summarizes the tag frequency for six selected manuals;

[3]https://github.com/showcases/projects-with-great-wikis
[4]https://github.com/showcases/nosql-databases
[5]https://github.com/search?q=Ruby+on+Rails+language:Ruby+stars:>200+size:>1000

	Change File	Add File	Reuse	Segment	User Input	Run Script	Pass Output	Check Output	Context-less	User Interaction	Total	#Tags
openprojectm	6	2	2		6	32	3	9		1	61	8
tracks installation	7		7		1	13	1	3		4	36	7
chef repo	3			5	7	10		3	1	3	32	7
rails1	41	5				15		5		1	67	5
ArangoDB	10				2	18	1	14			45	5
flockdb_demo				7	1	21		10		4	43	5
...	...	...	...	...	...	...	...	...	...	...	...	...
Total	150	20	17	26	32	402	9	84	23	41	804	
#Manuals	22	5	6	7	13	35	5	21	7	18	35	

Figure 3: Concept frequency in the survey

Index	Concept	Gen	BD	PD	TA	EK	HK
1	run script	x	x				x
5	user input			x			
7	run script	x			x		x
9	add to file	x	x			x	
15	change file	x	x		x	x	
...	...	...	...	...	...	...	...

# Commands	# Rule applications					
27	20	18	7	5	14	16

Figure 4: Effort types for exemplary commands are Auto-generation (Gen), Pattern detection (PD), Code body detection (BD), Text analysis (TA), Execution knowledge (EK) and Human knowledge (HK).

see the paper's web page for complete information. The figure displays the number of manuals ('#Manuals') and the number of distinct tags used in a manual ('#Tags'). We used extra tags, such as *context-less* to imply irrelevance for the executable manual and required *user interaction* where a task cannot simply be scripted. Command-line scripts are most popular, followed by file changes. All other concepts are significantly less common. Half of the manuals involve 'user interaction'—this provides an indication of the feasibility of complete automation for the manuals at hand. Six manuals contain context-less code blocks that corrrespond to examples on how to use the deployed system.

4. EFFORT ANALYSIS

We are interested in what kind of effort has to be conducted for concrete commands. Therefore, we documented the transformation for the manual on 'chef repo'; see the paper's web page for details. In the corresponding documentation we tried to identify what distinct kinds of effort have to be conducted in a transformation from the manual into an LDS conform manual, which also hints partially to what kinds of effort a reader normally has to conduct to follow the manual's guidance.

Since referenced variables have unique names, a simple *Auto-generation* technique is necessary to assign values to such attributes. *Code body detection* cares about screening the manual and identifying highlighted code bodies, which conform to fragments for block commands. Next, placeholders such as '<>' can be identified via *pattern detection*. Some information may be explicitly apparent in the text on what has to be done. Thus, *text analysis* is required to identify values for several attributes that may also be implicitly stated in the text. Necessary information may not be available in the manual itself. Some values have to either be looked up through trial and error runs (*Execution Knowledge*) or a reader may have to resort to his own or others' expertise (*Human Knowledge*) on specific challenges in order to realize an executable script.

Figure 4 presents an overview with numbers for five exemplary LDS commands and total numbers for all commands. Each concrete command is built up from multiple attributes. For instance, a 'run script' command has contained source code identified in the manual through BD and a specified executor implied by the text (HK).

5. CONCLUSION

Our domain analysis showed that there exist several common concepts in a manual. We have implemented these in an HTML dialect and applied them to existing manuals. Alternative implementations, possibly based on other documentation languages, may be interesting, if LDS should see widespread adoption.

The most interesting area for future work is the further generalization or integration of our literate deployment approach to be more generally applicable to deployment, e.g., by including aspects of build, dependency, and configuration management [7].

6. REFERENCES

[1] R. S. Hall, D. Heimberger, A. Van Der Hoek, and A. L. Wolf. The Software Dock: A Distributed, Agent-based Software Deployment System. Technical report, DTIC Document, 1997.

[2] M. Harman. Why Source Code Analysis and Manipulation Will Always be Important. In *Proc. of SCAM 2010*, pages 7–19. IEEE, 2010.

[3] M. Harsu. A survey on domain engineering. Technical Report 31, Institute of Software Systems, Tampere University of Technology, 2002.

[4] U. Kargen and N. Shahmehri. InputTracer: A Data-Flow Analysis Tool for Manual Program Comprehension of x86 Binaries. In *Proc. of SCAM 2012*, pages 138–143. IEEE, 2012.

[5] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[6] M. Lount and A. Bunt. Characterizing Web-Based Tutorials: Exploring Quality, Community, and Showcasing Strategies. In *Proc. of SIGDOC 2014*, pages 6:1–6:10, 2014.

[7] N. Zhang, G. Huang, Y. Zhang, N. Jiang, and H. Mei. Towards Automated Synthesis of Executable Eclipse Tutorials. In *Proc. of SEKE 2010*, pages 591–598, 2010.