A Scientific Workflow Construction Command Line

Paul T. Groth and Yolanda Gil
USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292
{pgroth, gil}@isi.edu

ABSTRACT
Workflows have emerged as a common tool for scientists to express their computational analyses. While there are a multitude of visual data flow editors for workflow construction, to date there are none that support the input of workflows using natural language. This work presents the design of a hybrid system that combines natural language input through a command line with a visual editor.

Author Keywords
Scientific workflows, command line, natural language.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI)

INTRODUCTION

As scientists increasingly perform their data analyses and experiments computationally, scientific workflows have arisen as a useful mechanism to represent, execute and share these analyses and experiments. Scientific workflows declaratively capture the steps of an analysis and the dependencies between them [4]. Steps are represented as components (e.g. software programs or web service invocations) that define the computations that should take place. Typically, dependencies are specified through the data flow between components. Once an analysis has been defined as a workflow, workflow systems can then be used to execute the analysis in changing environments [5], find appropriate data for the analysis [5], as well share it with other users [1]. Workflows are also widely used for business process management and many other applications.

In addition to the aforementioned functionality, workflow systems often provide graphical user interfaces for the construction and editing of workflows. When we use the term workflow system in the rest of this paper, we are referring to those that provide end user interfaces. Examples include Taverna [9] and Kepler [15] as research environments for scientific workflows, and there are also commercial workflow editors such as Tivoli and YAWL. While the editing interfaces of these systems differ in some respects, they all follow the general approach of representing workflows as a series of nodes (components) and arcs depicting dataflow between components.

Textual instruction using natural language is a very common way to describe procedural artifacts [16]. In spite of its ambiguity and other characteristics that make interpretation hard, natural language is a ubiquitous means for specifying concisely what needs to be done. In addition, research into visual programming calls into question the superiority of visual notations over textual languages [7, 17]. In this paper, we explore the textual input of workflows in natural language. A textual interface does not necessarily need to be in lieu of a visual dataflow editor, but rather be an alternative complementary form for a user for specifying workflows. Hybrid approaches that combine text and diagrams have been found to be very effective [11, 17].

Interpreting natural language comes with many challenges. Natural language has great variability and is inherently ambiguous. This makes the development of a textual editor challenging, since the system needs to interpret open-ended ambiguous text. To address this challenge, our system generates alternative interpretations of textual input, identifies those that are inconsistent with what it knows, and then displays to the user the interpretations of the instruction in the visual dataflow editor. The user can then select the appropriate interpretation or edit one of those generated by the system to create the workflow they had intended.

Despite the challenges, using natural language input may have other advantages. Our work explores the use of a command line interface to integrate the textual instruction with search capabilities. One important motivation is vocabulary variability, where users use different terms to refer to the same workflow components. [3] found that less than a dozen people out of a thousand would use a term that had been pre-selected to refer to a specific computer command. In recent work, a tool for finding API components (a similar environment to workflow systems), text based search over the Internet was shown to be an effective mechanism to allow programmers to find the correct component using their own vocabulary [20]. This issue is aggravated when repositories contain a large number of available workflow components [14]. Using the command line, we can integrate search directly with the input mechanism for workflow construction. Through the use of textual input, we aim to allow users to express workflows with a personalized vocabulary.

The rest of this paper is organized as follows. We begin by motivating our adoption of a hybrid-approach, in particular, the use of a command line for text entry. We then describe our user interface and how user input on the command line is processed and the results displayed to the user. After which, we describe an example usage scenario. Finally, we discuss related work and conclude.

COMMAND LINE INTERFACES

Search is used by millions of people everyday, increasingly; search not only retrieves results but also performs actions. Consider the following example: Typing the phrase "convert 12 kg to pounds" into Google performs the conversion and the answer (26 pounds) is returned.

Thus, search engines are becoming command lines, and as Norman states [18]:

> "*These modern command languages have some major virtues over the ones in the past. They are tolerant of variations, robust and exhibit slight touches of natural language flexibility*"

In this light, a properly implemented command line interface can provide the sort of robust user interaction to deal with vocabulary variability.

Another strength of command line interfaces is that, unlike GUIs where only a limited number of actions can be accessed through menus, buttons, and tree hierarchies, they can scale to a large number of actions. This is essential in the scientific workflow context, where there can be thousands of available components. A major disadvantage of command line interfaces was the necessity to memorize commands. With modern command lines this is no longer the case, as appropriate suggestions based on the system's current knowledge are made while the user types.

Command lines provide a familiar user interface element for textual input. Modern incarnations provide support for variable vocabulary and suggestions. Given these benefits, we adopt a command line as the interface element for the textual input of workflows. While modern command lines are flexible, they cannot replace the common visual dataflow representation of a workflow. Thus, we view the command line as an addition to current workflow editing practices, and, as we later show, this hybrid approach allows for important interactivity with the user.

W-CMD: THE WORKFLOW COMMAND LINE

The workflow command line (W-CMD) is designed to be an addition to current data flow based visual workflow editors. Currently, W-CMD is implemented as part of the Wings Workflow System [5]; however, the design of W-CMD is not tied to the Wings feature set. Through the rest of this discussion, we note when W-CMD can take advantage of particular Wings' features. W-CMD supports the most common type of information found in workflow descriptions, namely, step information (component types, inputs, outputs, preconditions, component ordering). As it interacts with the user, W-CMD cycles through the following phases: Suggestion Generation, Command Line Parsing, Hypothesis Generation, and Hypothesis Pruning.

The cycle can be summarized as follows. As the users types, suggestions are generated based on the current knowledge of the workflow system. When the user completes a command and hits enter, the command is parsed extracting as much information as possible about the intended workflow additions and modifications. Using this information along with the current workflow, a set of potential workflows is then generated. This set is then pruned to a single workflow through a combination of consistency checks (done by the system) and user selection. The user is then free to input another command.

Before describing each phase, we first describe the knowledge sources that W-CMD relies upon. This specifies the set of requirements for a workflow system that wishes to support command line entry of workflows.

- A mechanism for discovering available components, in particular, their names, inputs and outputs and their types (i.e. component/service descriptions). This is commonly referred to in workflow systems as a component or service registry [14].
- A list of *synonyms for components*.
- A list of *paraphrase patterns* that map sets of words to *operations* that can be performed on the workflows.

Each paraphrase pattern reflects how a user might express an operation. An example operation, AddComponent, is shown in Figure 1 along with some of its associated paraphrase patterns. The operation has five arguments denoted by a "+". Arguments with a trailing "?" are optional. Each paraphrase pattern consists of arguments with interspersed keywords. So for the first paraphrase pattern in Figure 1, the keywords are "performs", "on" and "and return" and the arguments are +component, "+input" and"+output".

Suggestion Generation

Suggestions are generated from a combination of component synonyms, component names and keywords extracted from paraphrase patterns. As the user types, suggestions are displayed according to alphabetical match. Figure 2(A) shows the interface for displaying suggestions. Once a user has selected a suggestion, it is inserted into the command.

Command Line Parsing

After the user finishes inputting a command (denoted by hitting return), it is then processed into an operation to be performed on the workflow being created and displayed as a workflow diagram. Inspired by work on to-do lists [6], our approach uses paraphrase patterns to perform the mapping of text to operations. Like to-do lists, workflows often have irregular and partial sentence constructs and the language used is often specific to the user.

When a command is parsed, W-CMD attempts to align the command with each paraphrase pattern in the system. For example, the command "This workflow performs a blastp search on protein sequence" would align with the first paraphrase in Figure 1 based on the keywords "performs" and "on". Note that alignment begins when the first keyword in a paraphrase pattern is matched. Hence, in the previous example, "This workflow" would be ignored.

Once alignment has taken place, the values for each argument can be extracted from the text resulting in *argument mappings*. Thus, "blastp search" would map to +component and "protein sequence" would map to +input in Figure 1's first paraphrase pattern. The alignment is best effort, if there are any keywords at the end or beginning of the command that do match the pattern, the system will still accept the paraphrase pattern.

It is often that case that more than one paraphrase pattern matches a command. To select the most appropriate interpretation, we rank the paraphrase patterns based on specificity defined as the greatest number of matching keywords. This ranking scheme is taken from [6].

There are some restrictions on the paraphrase patterns the system supports. Except for the +component argument, all arguments must be separated by keywords. We are able to relax this restriction for the component argument because the system can recognize component names using its internal knowledge of component types and synonyms. For the other arguments, this recognition knowledge is not currently supported. Thus, our system can support paraphrase patterns such as "1: +component +output". Once a paraphrase pattern has been selected by ranking, its corresponding operation can be looked up. Before proceeding to hypothesis generation, one more step is necessary.

Because of the vocabulary variability problem, even after mapping a component name (e.g. blastp search) to the component argument (+component), the underlying workflow system still may not be able recognize the given name. Hence, in this step, W-CMD uses its synonym knowledge to find a component name that is recognizable (e.g. blast_ddbj) and inserts the result in the argument mappings. If a recognizable name is not found, the component name extracted from the command is used. Once this step is finished, hypothesis generation is then provided the operation to perform on the workflow along with the argument mappings.

***Operation**:*
AddComponent +component +input? +output? +input_type? +output_type?

***Paraphrase patterns**:*
performs +component on +input and return +output
1: +component +output
given an +input of +input_type , +component
add +component that uses +input

Figure 1: An operation and its paraphrase patterns

In our implementation all paraphrase patterns are provided by hand and were derived from sentences found in a workflow description corpus [1]. However, we conjecture that paraphrases could be learned from the user by observing how the user modifies a workflow after the system fails to understand a command.

Hypothesis Generation

Given our prior analysis of textual descriptions of workflows, the argument mappings retrieved from parsing will be incomplete for the specified operation. For example, the command "add blastp resulting in a sequence" is missing the definition of the input to blastp. Hence, during hypothesis generation, W-CMD tries to fill in these missing arguments. This process often results in many possible argument mappings. For each possible mapping, the operation is then applied to the current workflow. As a consequence several *workflow hypotheses* are produced. We now discuss this hypothesis generation process in more detail using the example of adding one or more components to a workflow.

The first step, for adding components, in hypothesis generation is to determine any missing inputs and outputs. To do this, for each component obtained in the previous phase, W-CMD retrieves its component description from the component registry. From this description, the inputs, outputs and their types are extracted. If the user did not define any inputs or outputs for a component, those defined in the component registry are used.

After this step, if there is more than one component specified, W-CMD then reasons about the data flow between components using projection. It begins by connecting all components provided as input from parsing. If more than one component is provided, W-CMD starts with the first component and connects its outputs to compatible inputs from the second component in the list. The process then repeats starting with the second component and so on. Compatibility is determined by type checking. The result of this step is a set of workflow subsections that can be inserted into the current workflow. Currently, only one subsection is selected for insertion. Insertion into the workflow is determined by what outputs of the workflow correspond to the inputs of the first component within the subsection. For each compatible output and input, a *separate workflow* is hypothesized.

Hypothesis Pruning

Once a set of workflow hypotheses has been generated, the user, with help from W-CMD, needs to decide, which hypothesis corresponds to their intent, we term this phase hypothesis pruning. This phase is where the hybrid approach is critical. Each workflow hypotheses is presented to the user as a data flow diagram. The user moves between the hypotheses by clicking the next button as shown in Figure 2 (C) and (D), which also show two different hypotheses for the same command.

To aid the user, we use the workflow consistency check built into Wings to mark inconsistent workflows. This consistency check performs a form of goal regression on the workflow determining if input/output pairs are consistent according to the constraints on those variables retrieved from the component registry. Beyond these approaches, there are other possible pruning mechanisms including executing the hypotheses and marking those which do not complete.

While W-CMD could remove inconsistent hypotheses, we chose instead to allow the user to see all the available hypotheses because they could possibly be closer to the user's original intent. Essentially, the consistency provides extra information to the user for their pruning/selection task. Once the user selects a hypothesis, they can either operate on it with the standard visual interface or enter more commands. We now show how our current implementation of W-CMD performs in a particular scenario and present the features of the hybrid approach that aid usability.

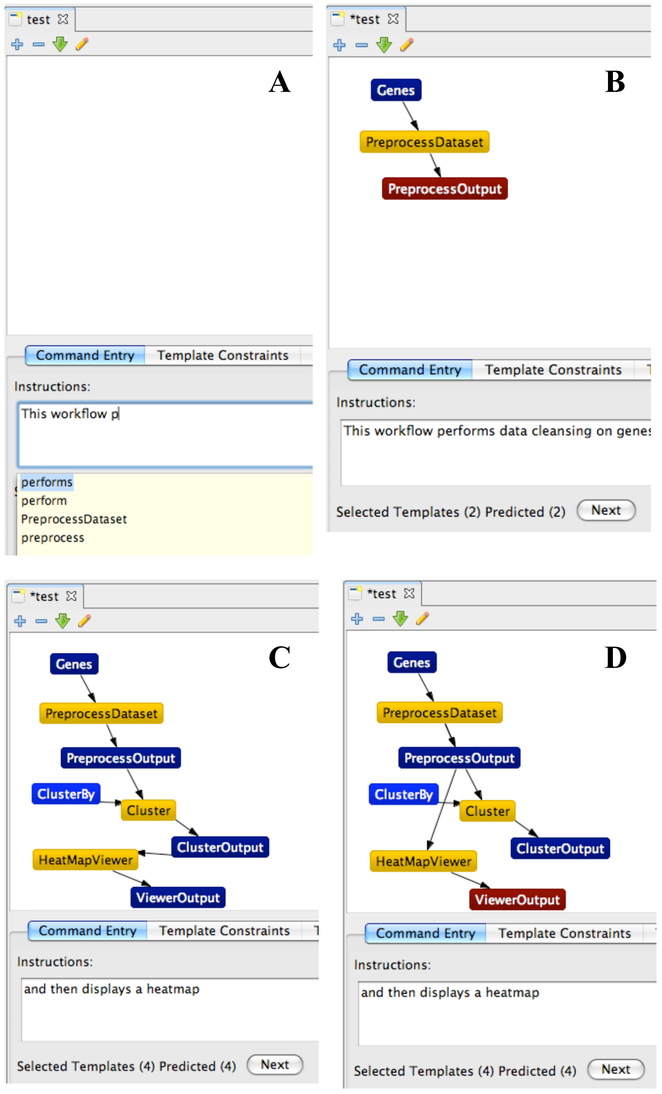
A USAGE SCENARIO

To demonstrate the operation of W-CMD, we used a set of bioinformatics components from the GenePattern project (www.genepattern.org). These components were chosen because they have already been modeled in the Wings workflow framework. In the context of this domain, we mirrored a textual description found in a workflow corpus [1]: "This workflow performs data cleansing on genes, clusters the results and then displays a heatmap."

The text was entered as three separate commands:

1. This workflow performs data cleansing on genes,
2. clusters the results
3. and then displays a heatmap

The workflow shown in Figure 2 (B) is the result of entering command 1. Note that "data cleansing" is a synonym for the PreprocessDataset component. The workflows shown in Figure 2 (C) and (D) are the two hypotheses generated after entering the last command. Both hypotheses are consistent with the user's input. The user then selects Figure 2 (C), as it is the workflow they had intended to specify.



This workflow p

performs
perform
PreprocessDataset
preprocess

This workflow performs data cleansing on genes

and then displays a heatmap

Figure 2: A hybrid environment for workflow creation

RELATED WORK

Other approaches to learning procedural knowledge from instruction use natural language [8]. These systems take on instruction in natural language, but then well-known phenomena in natural language processing start to crop up including ambiguities, lack of referents, and unresolved attachments. Our approach constrains the utterance that the system is trying to interpret to deal with these difficulties.

An interface design based on controlled natural language might be an alternative option [2]. Controlled languages are designed by defining a well-specified grammar for each utterance. The user is then confined to specify their statements by following that grammar. The advantage is that it is easier for the system to interpret the utterances. The disadvantage is that users must get used to the grammar and how it allows them to express their statements.

Specifying procedures through example demonstrations is another approach investigated in the literature [12, 13, 19]. However, the expressivity of the languages and the constructs allowed pose limitations on the complexity of the procedures that users can specify.

Visual interfaces and other natural interface designs have been developed that are effective means for end users to specify procedural information to a system. [10] provides a thorough overview of such systems, which they call empowering systems since they aim to help non-programmers to specify behaviors for a computer system. Some of the best-known systems are AgentSheets, Stagecast, Logo, Alice, Forms/3, and Hypercard. Most of these tools are designed to target specific tasks, objects, or behaviors. They have been shown effective in experiments with non-programmers. Perhaps integrating some of these approaches into workflow interfaces would be beneficial.

CONCLUSION

This work's contribution is a hybrid interface for the textual input of workflows. This work has only begun to address the difficult problem of handling workflows input as text. Areas for future work include learning and capturing paraphrases, handling advice, goals, and sub-workflows, allowing for personalization, dealing with more workflow operations, and providing more relevant suggestions.

ACKNOWLEDGEMENTS

We gratefully acknowledge support from the US Defense Advanced Research Projects Agency under grant number HR0011-07-C-0060.

REFERENCES

1. De Roure, D., Goble C., Stevens, R. "The design and realization of the myExperiment Virtual Research Environment for social sharing of workflows". Future Generation Computer Systems. In Press.
2. Fuchs, N. E. and R. Schwitter. "Attempto Controlled English (ACE)." Proceddings of the First International Workshop on Controlled Language Applications (CLAW), 1996.
3. Furnas, G., Landauer, T., Gomez, L., and S. Dumais. "The Vocabulary Problem in Human-System Communication." *Communications of the ACM*, 30, 1987.
4. Gil Y, Deelman E., Ellisman M., Fahringer T., Fox G., Gannon D., Goble C., Livny M., Moreau L., Myers J., "Examining the Challenges of Scientific Workflows," Computer , vol. 40, no. 12, pp. 24-32, December, 2007.
5. Gil Y., Ratnakar V., Deelman E, Mehta G, and Kim J. "Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows," *Proc. of the 19th Annual Conf. on Innovative Applications of Artificial Intelligence (IAAI),* Vancouver, British Columbia, Canada, July 22-26, 2007.
6. Gil Y. and Ratnakar, V. "Automating To-Do Lists for Users: Interpretation of To-Dos for Selecting and Tasking Agents." Proc. of the Twenty-Third Conference of the Association for the Advancement of Artificial Intelligence (AAAI-08), Chicago, IL, July 13-17, 2008.
7. Green, T.R.G. and M. Petre (1996). "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework." Journal of Visual Languages and Computing 7(2): 131-174.
8. Huffman, S. and Laird, J. 1995. "Flexibly Instructable Agents." *Journal of Artificial Intelligence Research*, 3.
9. Hull D, Wolstencroft K, Stevens R, Goble C, Pocock MR, Li P, Oinn T. "Taverna: a tool for building and running workflows of services." Nucleic Acids Res., 34, 2006.
10. Kelleher, C. and R. Pausch. "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers." *ACM Computing Surveys*, 37(2), 2005.
11. Koedinger, K. and J. Anderson. "Abstract Planning and Perceptual Chunks: Elements of Expertise in Geometry." *Cognitive Science*, 14(4), 1992.
12. Lau, T., Wolfman, S., Domingos, P. and D. S. Weld. 2003. "Programming by Demonstration using Version Space Algebra." *Machine Learning*.
13. Lieberman, H. (Ed). 2001. "Your Wish Is My Command: Programming By Example", Morgan Kauffman.
14. Lord P., Alper P., Wroe C., and Goble C. "Feta: A lightweight architecture for user oriented semantic service discovery." 2nd European Semantic Web Conference, (2005)
15. Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger-Frank E., Jones M., Lee E., Tao J., Zhao Y. "Scientific Workflow Management and the Kepler System." *Concurrency and Computation: Practice & Experience*, 18(10), pp. 1039-1065, 2006.
16. Miller, L. "Natural language programming: Styles, strategies and contrasts", *IBM Systems Journal*, 20, 1981.
17. Nardi, B. A. (1995). "A Small Matter of Programming: Perspectives on End User Computing." MIT Press, Cambridge, MA, 1995.
18. Norman, D. 2007. "The next UI breakthrough: command lines." *interactions* 14, 3 (May. 2007)
19. Smith, D. C., Cypher, A., Tesler, L. G. 2000. "Novice Programming Comes of Age". *Communications of the ACM* 43(3).
20. Stylos J. and Myers B. "Mica: A Programming Web Search Aid". IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006). 2006.