

Workflow Instance Detection: Toward A Knowledge Capture Methodology for Smart Oilfields

Fan Sun
Dept. of Computer Science
Univ. of Southern California
Los Angeles, CA 90089
fansun@usc.edu

Viktor Prasanna, Amol Bakshi
Ming Hsieh Dept. of EE
Univ. of Southern California
Los Angeles, CA 90089
{prasanna, amol}@usc.edu

Laurent Pianelo
Energy Technology Company
Chevron Corporation
San Ramon, CA 94583
lpnl@chevron.com

Abstract

A system that captures knowledge from experienced users is of great interest in the oil industry. An important source of knowledge is application logs that record user activities. However, most of the log files are sequential records of pre-defined low level actions. It is often inconvenient or even impossible for humans to view and obtain useful information from these log entries. Also, the heterogeneity of log data in terms of syntax and granularity makes it challenging to extract the underlying knowledge from log files. In this paper, we propose a semantically rich workflow model to capture the semantics of user activities in a hierarchical structure. The mapping from low level log entries to semantic level workflow components enables automatic aggregation of log entries and their high level representation. We model and analyze two cases from the petroleum engineering domain in detail. We also present an algorithm that detects workflow instances from log files. Experimental results show that the detection algorithm is efficient and scalable.

1 Introduction

The oil industry expects to face an imminent shortage of professionals due to a combination of factors such as a large numbers of oil workers nearing retirement age and a shortage of replacements [1]. As a result, there is great interest in systems that can capture the knowledge of experienced workers, help maintain reliable operation in the face of personnel turnover, and act as a knowledge base to assist with troubleshooting and training new employees.

A smart oilfield is one that uses technical innovations to measure, analyze and optimize cost-effective oil and gas production [2]. In the past few decades, many software applications have been developed to model, simulate, and predict the behavior of reservoirs. Log files are widely used in such applications to record actions performed by users

[3, 5]. An engineer making decisions typically performs analysis involving, for example, different uncertainty realizations of the asset and different operational strategies. Audit trails of the decision making processes can be maintained by logging, including information about the sequence of actions, the consumed and generated data objects, and metadata that provides additional information for the data. All this information makes log files a valuable source of knowledge to be captured from experienced users.

Such log files are often not suitable directly as knowledge base for two reasons. First, log files record only low level information about user activities. They do not capture the semantics, such as how the domain expert accomplishes a certain task, what a particular data set means in the domain context, the potential causal relationships between a series of activities, etc. Second, different software applications are not designed to work together. The heterogeneity of log data in terms of syntax and granularity makes it challenging to extract underlying knowledge from log files.

In this paper, we propose a methodology to extract workflow instances from software application logs. A workflow, which consists of a sequence of operations, is an abstraction of real work for a specific goal. Workflow instances are runtime instances of workflows, which contain runtime information about user activities. This paper is the first step toward the knowledge capture system we are building in the petroleum engineering domain. We define a semantically rich workflow model to provide mapping from application-specific log entries to application-independent logical steps. The semantic information in the proposed workflow model describes the “meaning” of user activities. We present WIDA, a workflow instance detection algorithm, which efficiently detects workflow instances from log files. Semantic information can be used to query and retrieve the detected workflow instances.

The rest of this paper is organized as follows. Section 2 surveys existing methodologies and systems relevant to our

work. In Section 3, we define a semantically rich workflow model in the petroleum engineering domain, and use it to model and analyze two real-world cases in detail. Section 4 presents the workflow instance detection algorithm. We perform experiments on Gocad log files and analyze experimental results in Section 5. Section 6 concludes the paper.

2 Related Work

There has been increasing interest in research on smart oilfields by both academia and industry. Many key challenges in developing a smart oilfield have been identified, such as real-time asset management [6], data exchange format [7], and implementation on a real oilfield asset [8].

Knowledge capture and management are challenging issues in smart oilfields. According to Irrgang et al. [9], there are two main categories of knowledge management: people-to-people networks and IT assisted people knowledge base exchange. Drnec et al. [10] developed an integrated project management system for knowledge capturing and sharing. Gibby et al. [11] describe a knowledge framework to address the governance of knowledge as well as the support elements including people, process and technology. Both works mainly focused on the first category of knowledge management, while our work focuses on the second one. Our objective is to capture underlying knowledge from log files to help with reservoir management.

Zhao et al. [12] proposed a workflow model with two layers, which provides mapping between logical steps and concrete tasks. The hierarchical structure of their workflow model definition looks very similar to ours (see Section 4.1), but the intentions are different. First, their workflow model was designed for business transactions and processes, which are modeled using graph structure. In our system, typical workflows are sequential patterns. Second, the data in their workflow were text files, which can be unified as documents in XML format. In our system, heterogeneous data from various applications cannot be unified with a single format. Thus, generalization of heterogeneous data must be addressed. Third, many workflows in reservoir management are data centric. Dataflow constraints must be specified to maintain consistency.

3 Semantically Rich Workflow Model

3.1 Workflow Model Definition

The semantically rich workflow model is defined in a hierarchical structure. It consists of three levels: workflow, intentions and realizations. Essentially, a workflow is composed of a list of intentions in a sequential pattern. Each intention semantically describes “what to do” in one step in the workflow. An intention may have one or more realizations. Each realization describes “how to execute” an intention in concrete steps within a specific application.

Dataflow constraints are specified over intentions to maintain consistency across data. Figure 1 shows the proposed hierarchical structure of workflow model.

3.1.1 Workflow Level

The workflow level provides an overview of the workflow model. A workflow is defined as a tuple

$$wf = \{inputs, outputs, ints, dfcs\},$$

where *inputs* and *outputs* are the input and output data of this workflow. *ints* denotes a list of *intentions* that comprise the workflow in a sequential pattern. *dfcs* is a set of *dataflow constraints* on these intentions. A dataflow constraint (shown as the dashed arrows in Figure 1), is a one-to-one mapping of data to maintain consistency. For example, a constraint ($ints[i].inputs[j], data_k$) specifies that $data_k$ is required to be the j -th input data of the i -th intention.

Beside these fields, metadata (including name, author, description, version, etc.), of a workflow are also recorded in this level. Metadata provides semantic information about workflows, which makes them both processable by machines and understandable by humans.

3.1.2 Intention Level

The intention level defines intentions as logical components of a workflow. An intention describes one step of a workflow conceptually, while keeping the implementation details transparent. The intention is essentially an abstraction of application-specific actions on an application-independent logical step. The generalization of intentions provides flexibility and reusability in composing new workflows.

An intention is defined as a tuple

$$int = \{inputs, outputs, desc, realizes\},$$

where *inputs* and *outputs* are the input and output data of this intention. *desc* describes what the intention aims to do. *realizes* denotes a set of *realizations* which implement this intention in various software applications. Each intention may have one or more possible realizations, while a realization maps to exactly one intention. For instance, suppose we have an intention to “run a reservoir simulation.” The realization could run either a numerical simulation such as ECLIPSE [13], or a material balance simulation such as MBAL in IPM [4]. Both types of realizations implement this intention.

3.1.3 Realization Level

The realization level provides implementation details for intentions. A realization describes how an intention can be achieved within a specific application.

A realization is defined as a tuple

$$realiz = \{int, app, actions, dms\},$$

where *int* denotes the intention it implements, and *app* is the application in which it implements. *actions* is a sequence of actions in *app*, which describes how the corresponding intention is achieved step-by-step. Each action

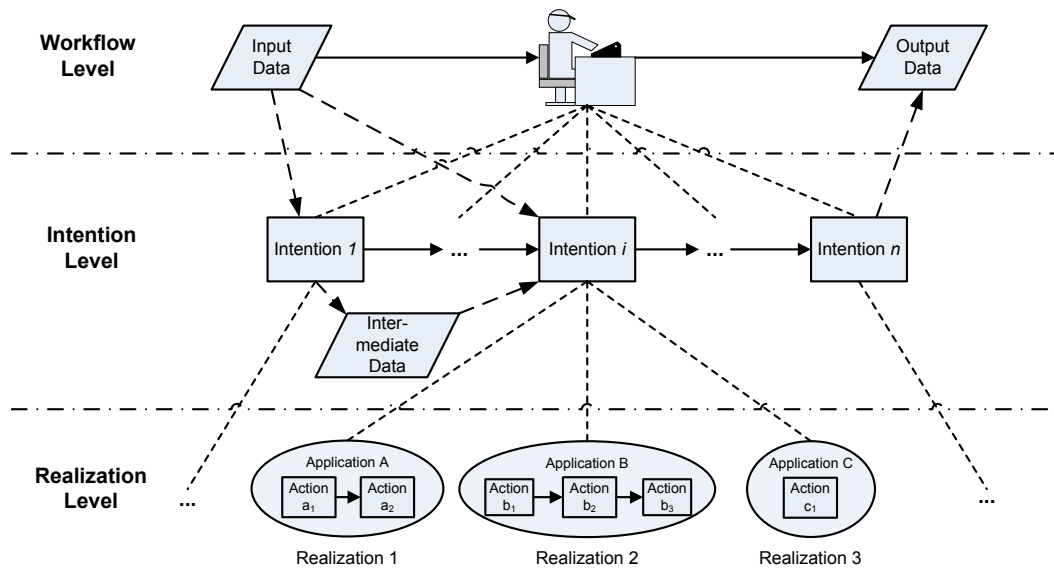


Figure 1. Hierarchical Structure of Workflow Model

contains a pre-defined operation and its relevant data objects as parameters. dms is a set of *data mappings* which maps those data objects to the input and output data defined in the intention level. For instance, a mapping entry ($int.inputs[i], actions[j].data_k$) specifies that the data object $data_k$ in the j -th action corresponds to the i -th input data of the corresponding intention int .

3.2 Case Studies

We performed case studies on two real-world examples to illustrate the workflow model. The first workflow showed the mapping between abstract intentions and concrete implementations, and the second workflow focused on intermediate data objects modeling and application interactions.

3.2.1 Upscaling Workflow in Gocad

Upscaling refers to the process that generates a simulation grid from a geological grid. To reduce the large amount of computation in reservoir simulation, the simulation grid is usually made to be more coarse-grained than the geological grid. To build such a coarse grid, design of the upscaling, including proportion of the scaling and properties we want to scale, must be specified. Essentially, there are three logical steps in a upscaling process: upscaling design, coarse grid creation, and properties calculation.

Gocad (Geological Object Computer Aided Design) [3] is a widely used software application for geological modeling in petroleum engineering. As shown in Figure 2, the upscaling workflow is implemented using 7 built-in commands in Gocad. Three logical steps have been implemented as intentions, each consisting of a sequence of commands. In Figure 2, two types of dataflow constraints have

been specified. The dashed arrows from the input data imply that all three intention modules take the “fine grid” as input. The dashed arrows pointing to the output data show the second type of constraint. It implies that the coarse grid, which is generated by the second intention, is also the output of the third intention. The intermediate data and dataflow constraints have been omitted for simplicity.

```

1 gocad set_working_directory "dir_a"
2 gocad on SGrid SGrid_1 scp_create name SCP_new
3 gocad on SGrid SGrid_1 scp_initialize_properties name SCP_new por..
4 gocad on SGrid SGrid_1 scp_set_template_file name SCP_new templat..
5 gocad on SGrid SGrid_1 scp_build_coarse_grid name SCP_new use_cus..
6 gocad on SGrid SGrid_1 scp_effective_properties name SCP_new defa..
7 gocad on SGrid SGrid_1 scp_derived_properties name SCP_new comput..

```

Figure 3. A Log Segment in Gocad

Figure 3 shows a log segment in Gocad, which records an execution of the 7 commands described above. In this example, “SGrid_1” is the fine grid as input, and “SCP_new” is the upscaling problem that contains the generated coarse grid as the output. The dataflow constraints indicate that “SCP_new”, created in the first intention, is taken as an input by the second and third intentions.

3.2.2 OOIP Tracking Workflow

OOIP (Original Oil In Place) tracking is one of the key workflows in oilfield reservoir management, and involves a periodic estimation of the OOIP content based on the observed production data from the field.

Logically there are four steps to achieve an OOIP tracking workflow. First, we obtain and validate the well test data. Second, we run a diagnostic process to analyze the validity of the well test data in detail, and update the well

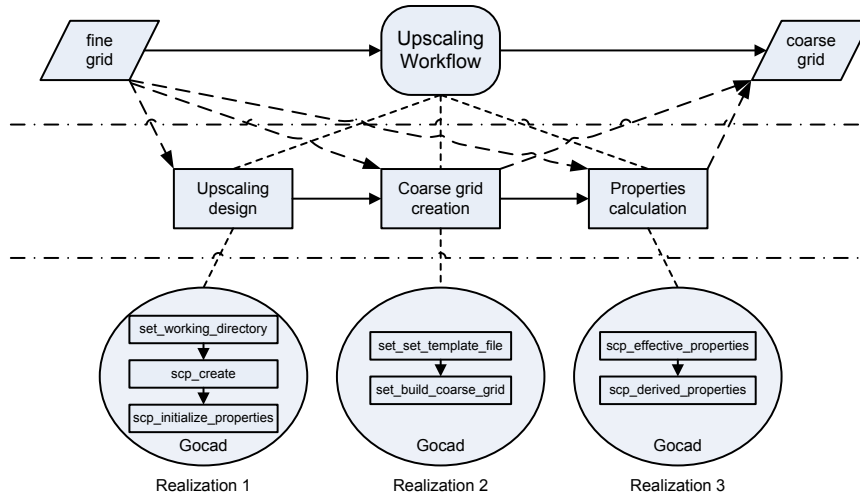


Figure 2. The Upscaling Workflow Model in Gocad

test data or well model accordingly. Third, we calculate the reservoir pressure based on the well test data. Finally, we compute the regressed OOIP estimate based on the validated well test data and cumulative production history data. We implement this OOIP tracking workflow in IPM (Integrated Production Modelling) [4] and IFM (Integrated Field Management) [5]. The first three steps are implemented by invoking the built-in functionalities of IFM. For the last intention, we run an inverse material balance computation in MBAL to get the OOIP estimate.

A set of dataflow constraints must be specified in the OOIP tracking workflow. For instance, the validated well test data are taken as input by the well test diagnostic process, the updated well test data or well model are used in the reservoir pressure calculation, and the calculated reservoir pressure is an input of the OOIP regression step.

Figure 4 shows the OOIP tracking workflow model designed above. The input includes three data: cumulative production history, simulation model, and well test data. The output is the regressed OOIP. Intermediate data, including the validated well test data, updated well test or well model, and reservoir pressure, are also shown in detail. Dashed arrows show the dataflow constraints described above. The implementation details of IFM and MBAL have been omitted for simplicity.

4 Workflow Instance Detection

Workflow instances are execution records of workflows. These instances contain runtime information about the corresponding workflow, e.g. which application has been chosen to realize each intention, how the data are consumed and generated, how the parameters are set, etc. The goal of *workflow instance detection* is to identify all the workflow instances in an input log file according to a given set

of workflows.

4.1 Problem Definition

We first define the relationship between log entries and the elements in our hierarchical workflow model in a bottom-up fashion.

- A log entry l is an instance of (or matches) an action $actions[i]$ if it records an execution of $actions[i]$.
- A sequence of log entries $L_k = l_1 l_2 \dots l_n$ is an instance of (or matches) a realization $realiz$ if $realiz$ contains n actions, and the i -th log entry l_i matches the i -th action of $realiz$, i.e. $realiz.actions[i]$ for all $i = 1 \dots n$.
- If $L_k = l_1 l_2 \dots l_n$ is an instance of a realization $realiz$, it is also an instance of (or matches) the corresponding intention, i.e. $realiz.int$.
- A log segment L is an instance of (or matches) a workflow wf if the log segment can be partitioned into m subsequences $L = L_1 L_2 \dots L_m$, and wf contains m intentions, such that L_i is an instance of $wf.intentions[i]$ for all $i = 1 \dots m$, and all the dataflow constraints $wf.dfcs$ are satisfied.

The problem of *workflow instance detection* is defined as follows:

- Given a log file $L = l_1 l_2 \dots l_n$ and a set of workflows $wfs = \{wf_1, \dots, wf_t\}$, identify all the instances of wfs in L .

We assume that a workflow instance consists of consecutive log entries in the input log file. In preprocessing, we merge multiple log files by sorting log entries by their time stamps. For each log entry, the corresponding application that generated it could be identified.

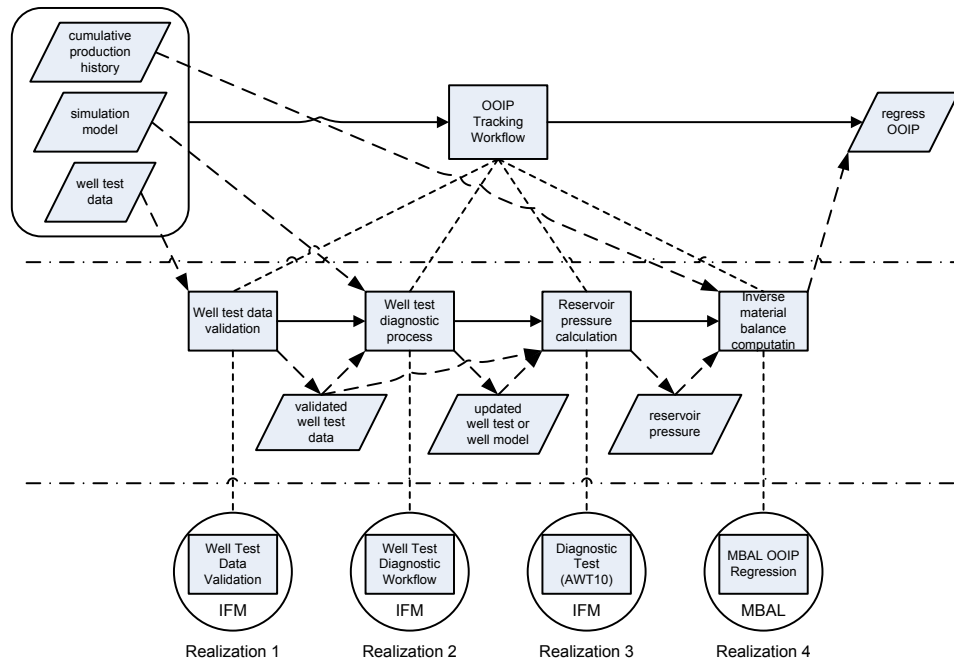


Figure 4. The OOIP Tracking Workflow Model

4.2 Aho-Corasick Algorithm for String Matching

According to the workflow model definition in Section 4, the hierarchical structure contains two layers of sequential patterns. In the upper layer, a workflow consists of a sequence of intentions, called a *workflow pattern*. In the lower layer, each realization consists of a sequence of application-specific actions, called a *realization pattern*. To detect precisely a workflow instance, we perform string matching in both layers.

Aho-Corasick algorithm [14] is a precise multi-pattern string matching algorithm, which was conceived to solve the dictionary problem. It uses a keyword tree to find all occurrences of any pattern from a set in a text string. The keyword tree is a finite automaton that encodes each keyword as a path from the *start state* to an *output state*. A keyword is matched if the corresponding output state is reached.

The keyword tree also stores information about pattern similarity in the form of *failure links*. When the algorithm encounters a partial match in the text, it uses the failure links to continue the search at the mismatch character without having to re-sample characters in the text. Use of this mechanism allows the Aho-Corasick algorithm to achieve linear time complexity [14].

4.3 WIDA: Workflow Instance Detection Algorithm

WIDA is a workflow instance detection algorithm that uses Aho-Corasick keyword trees for string matching in

both workflow and realization layers. Figure 5 shows the flowchart of WIDA.

In the realization layer, the log file is the input string, and each log entry is an input symbol. Each realization pattern is a keyword, and keyword trees are constructed according to the keywords. Since realizations in different applications do not overlap, we group realization patterns by their corresponding applications, and construct one keyword tree for each application. This reduces the size of the keyword tree. Such keyword trees are called *realization pattern trees*. Each time a log entry is read, WIDA checks the application that generates it, and feeds the log entry as an *action* into the corresponding realization pattern tree as an input. The realization pattern trees detect if a subsequence of the input log file matches any realization patterns. If so, an instance of the corresponding intention is generated as an output. The input and output data fields are populated by the data objects from logs according to the data mapping of the detected realization.

In the workflow layer, the input symbols are intentions generated by the realization pattern trees in the realization layer. Each workflow pattern is considered a keyword. A keyword tree is constructed according to these keywords, and is called a *workflow pattern tree*. It takes the intention instances generated from actions trees as inputs, and detects if any workflow patterns can be matched. If so, the algorithm then checks if all the dataflow constraints in the matched workflow are satisfied, in which case a workflow instance is generated and populated in the workflow

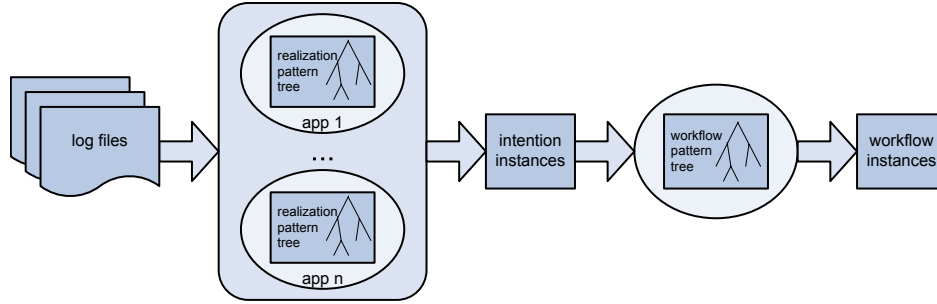


Figure 5. The Flowchart of WIDA

instance repository.

The detecting process of WIDA is shown in Figure 6.

Input. A log file $L = l_1 l_2 \dots l_n$ where each l_i is a log entry, a set of realization pattern trees $\{R_j\}$, and a workflow pattern tree W .

Output. Workflow instances detected in L .

Method.

```

for  $i = 1$  to  $n$  do
  identify  $R_j$  of the application that generates  $l_i$ ,
  input  $l_i$  to  $R_j$ ,
  if a realization pattern  $r_s$  in  $R_j$  is matched then
    generate intention instance  $inst_{int}$  according to  $r_s$ ,
    input  $inst_{int}$  to  $W$ ,
    if a workflow pattern  $w_t$  in  $W$  is matched then
      check dataflow constraints in  $w_t$ ,
      if all the dataflow constraints are satisfied then
        generate workflow instance  $inst_{wf}$  according
        to  $w_t$ ,
        output  $inst_{wf}$ .
      end if
    end if
  end if
end for

```

Figure 6. Key Steps of WIDA

4.4 Complexity Analysis

Tree Construction Time. The construction time of a Aho-Corasick keyword tree is linear in the sum of the keyword lengths [14]. In WIDA, there is one workflow pattern tree and h realization pattern trees, where h is the number of applications. Hence, the construction time of WIDA is linear in the sum of all the workflow pattern lengths plus the sum of all the realization pattern lengths. We define the length of a workflow as its workflow pattern length plus the sum of its realization pattern lengths. Thus, the construction time of WIDA is linear in the sum of the workflow lengths.

Execution Time. In the detection process, each time a realization pattern match occurs, WIDA populates the in-

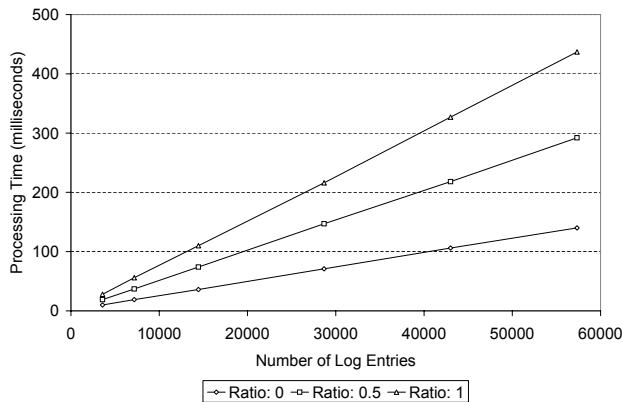
put and output data of the generated intention instance according to the *data mapping* in the realization. Each time a workflow pattern is matched, WIDA checks the dataflow constraints of the corresponding workflow. The execution time of WIDA is $O(n + mp + kq)$, where n is the number of log entries in the input log files, m is the number of intention instances, p is the maximum number of data mapping entries among all intentions, k is the number of workflow instances, and q is the maximum number of dataflow constraint entries among all workflows. Note that m and k are both less than n . For fixed p and q , the execution time of WIDA is proportional to the length of the input, i.e. $O(n)$.

5 Experimental Results

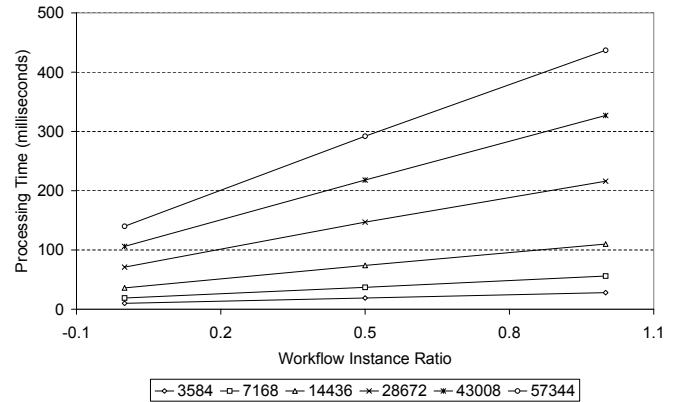
We tested the performance of WIDA using Gocad generated log files as input data. An example of Gocad log files has been shown in Figure 3. To parse the Gocad log files, we developed a set of grammar rules, and then generated a parser using ANTLR [15]. The Gocad upscaling workflow (see Figure 2) was used as the workflow pattern in the experiments. With an input log file containing 50,000 log entries, the processing time was less than 0.4 seconds. This shows that WIDA is efficient.

We also tested the scalability of WIDA. We performed experiments on data sets with various numbers of log entries and various *workflow instance ratios*. Here workflow instance ratio denotes the ratio of number of log entries that are contained in detected workflow instances to the total number of log entries. We created 18 data sets with six different length of logs: 3584, 7618, 14436, 28672, 43008, 57344, and three workflow instance ratios: 0, 0.5, 1.

Figure 7 shows the experimental results on the 18 data sets. Figure 7a shows the trends in the execution time of WIDA on different sizes of data set, while each line refers to a fixed workflow instance ratio. Figure 7b shows the trends of running time of the detection process on different workflow instance ratios, while each line refers to a fixed number of log entries. The experimental results show that the execution time scaled linearly with regard to length of log file, as well as the number of workflow instance occurrences.



(a) Plot I



(b) Plot II

Figure 7. Experimental Results

6 Conclusions

In this paper, we defined a semantically rich workflow model specifically for smart oilfields, and proposed an algorithm to detect workflow instances from log files. Workflow instances capture runtime information of user activities in semantic level, and will be further analyzed for knowledge extraction. An assumption of workflow instance detection is that all the workflow patterns are pre-defined by users. To relax this assumption, our next step is to semi-automatically detect workflow patterns from log files.

Acknowledgment

This research was funded by CiSoft (Center for Interactive Smart Oilfield Technologies), a Center of Research Excellence and Academic Training, and a joint venture between the University of Southern California and Chevron. We are grateful to the management of CiSoft and Chevron for permission to present this work. We thank Will Da Sie (Chevron Corp.) for sharing a wealth of domain knowledge, and providing feedback on our prototypes.

References

- [1] F. Rengers and J. Scholten. Skills Shortage 'The Way Forward'. Middle East Oil Show, Bahrain, June 2003.
- [2] I. Ershaghi and Z. Omoregie. Continuing education needs for the digital oil fields of the future. 2005 SPE Annual Technical Conference and Exhibition, SPE 97288, Dallas, Texas, October 2005.
- [3] Gocad, <http://www.gocad.org/>
- [4] IPM, <http://www.petex.com/products/>
- [5] IFM, http://www.petex.com/ifm_products/
- [6] T. Unneland and M. Hauser. Real time asset management: From vision to engagement an operators experience. SPE Intelligent Energy Conference and Exhibition, April 2006.
- [7] B. Weltevrede et al. A multivendor data-exchange format to support digital oilfields. SPE Intelligent Energy Conference and Exhibition, April 2006.
- [8] J. Ouimette and K. Oran. Implementing Chevrans ifield at the San Ardo, California, asset. SPE Intelligent Energy Conference and Exhibition, April 2006.
- [9] R. Irrgang, S. Kravis, and E. Nakagawa. Drilling Knowledge Management, What is Missing and Can We Fix it? IADC/SPE Asia Pacific Drilling Technology, 8-11 September 2002, Jakarta, Indonesia.
- [10] M.L. Drnec, B. Balci, and J. Etkind. New Shared Organization-Learned Project Management System provides a Knowledge Hub for Integrated Reservoir Optimization. IADC/SPE Asia Pacific Drilling Technology, 8-11 September 2002, Jakarta, Indonesia.
- [11] P.J. Gibby, N. Milton, W.A. Palen, and S.E. Hensley. Implementing a Framework for Knowledge Management. International Petroleum Exhibition and Conference, 5-8 November 2006, Abu Dhabi, UAE.
- [12] L. Zhao, J. Xing, and L. Meng. The Research and Realization of a New Workflow Model with Step-Task Two Layers Based on Document. IEEE Asia-Pacific Conference on Services Computing (APSCC), 2006.
- [13] ECLIPSE, <http://www.slb.com/>
- [14] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. Communications of the ACM 18 (6): 333C340, June 1975.
- [15] ANTLR, <http://www.antlr.org/>