Graph Based Workflow Validation

Anastasios Giouris and Manolis Wallace
Department of Computer Science,
University of Indianapolis Athens,
Ipitou 9, Syntagma, 10557, GREECE
http://cs.uindy.gr
cs@uindy.gr

Abstract. Content Management Systems in the field of content production and publishing need to monitor the progress of large numbers of simultaneously evolving workflows. In order to make such systems adaptive to the varying needs of their organizations, the utilization of a graphical editor for workflows has been proposed, thus generating the hazard of the specification of erroneous and invalid workflows. In this paper we provide a formal modeling for workflows and then, based on it, we explain how the validity of a simple workflow can be evaluated through the examination of the transitive closure of its graph. Continuing to the more general case, we develop a problem transformation methodology that allows for all workflows to be validated using a transitive closure examination approach. The Python implementation of our methodology is provided as is freely to all interested parties.

1 Introduction

Living in the information age, we are constantly faced with the need to be able to handle large amounts of content. Especially with the constant expansion of Web 2.0 type websites which are becoming increasingly popular along with which the content handled by such websites grows also.
In this direction, content management systems (CMSs) have been developed that undertake the burden of online monitoring of the managing process for large numbers of elements. Of course, in order for this managment to take place, detailed production workflows need to be defined in a formal and machine understandable way [1].

Manually editing a workflow can become a cumbersome task and often the source of operational errors; it is not always easy for a human user to detect all possible deficiencies associated with a complex graphical workflow description and, consequently, problems in the specification of the workflow will reflect in the operation of the overall system. Therefore, what is needed is an automated tool that is able to inspect the manually provided workflow specifications and assess their validity [2,3].
In this paper, utilizing a graph-based model for workflows, we propose a transitivity based methodology for the validation of workflows. The structure of the paper is as follows: In section "Workflow graph model" we discuss workflows and present our graph based workflow model. In section "Workflow Validation" we present our methodology for automated validation of workflows and in section "Experimental results" we provide results from the experimental application of our approach. Finally, section "Conclusions" lists our concluding remarks.

2 Workflow graph model

A workflow can best be described as an outline of the path that needs to be followed in order for a task to be successfully completed. The specific path that is followed often determines not just the quality of the results, but also whether the task is ever completed or not [4,5].

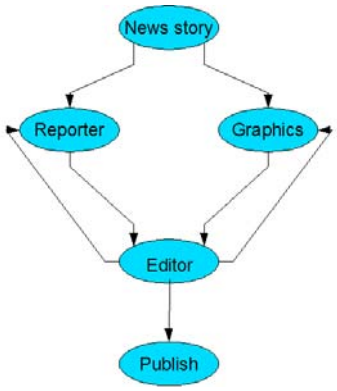

Fig. 1. An example of a workflow

Visually, a workflow can be thought of as a flowchart. From a mathematical point of view, it can be thought of as a Finite State Machine. But neither of these two views can exactly represent the complicated conditions a workflow requires, such as handling different and multiple inputs and outputs. Also, where in programming we consider the termination problem unsolvable, we wish to have an automated methodology that is able to determine whether a workflow specification will lead to

successful completion of the task which it describes. Therefore, an alternative model is required for the representation of the workflow.
The following is a simplified real world graphical example of a workflow: in a moderated news portal environment that publishes edited versions of articles provided by users, each article submitted needs to be tracked though the process displayed in Fig 1. The article is first forwarded for both text proofs and graphics touch up. Either of these two sub-processes could develop into a longer sequence of edit-review iterations, until the desired quality has been reached. Finally, when the editor has approved both the textual and graphical part of the article, it can be published. Clearly, the real life version of this workflow is even more complex, also describing the editor's ability to totally reject the article, re-assign it to different proofreaders or graphics artists, merging it with other articles, associating it with a specific section of the portal, requesting changes to its length and so on.

Fig. 2. The start node

Fig. 3. The end node

Fig. 4. SINGLE_IN/SINGLE_OUT node

Fig. 5. SINGLE_IN/ALL_OUT node

Fig. 6. ALL_IN/SINGLE_OUT node

Fig. 7. ALL_IN/ALL_OUT node

In order to construct a formal model for the workflow presented above we will extend on the notion of a graph by providing definitions for six different types of graph vertices, as explained in the following:

- The start of the process is a special node in the graph, as all paths originate there. We depict the start using the type of node presented in Fig. 2.
- The end of the process is also important as it automatically signifies the end of the task. This is depicted with the type of node presented in Fig. 3.
- Proof readers, right after the article has been assigned to them, as well as when the editor returns the article for further proofing, need to work on the document. Therefore we have a node with multiple inputs either of which may activate the node. This is called a SINGLE_IN node and is depicted with a rectangle in the input area, as shown in Fig. 4 and Fig 5.
- The editor, when receiving proofs from the proof readers, has the option to either accept the proofed article or return it for further proofing. Therefore we have a node with multiple outputs only one of which is activated in each case. This is called a SINGLE_OUT node and is depicted with a rectangle in the output area, as shown in Fig. 4 and Fig. 6.

- In order for the overall article to be accepted, both the text and the graphics need to be approved by the editor. This operation corresponds to a multiple input node for which all of the inputs are required for activation. This is called an ALL_IN node and is depicted with an oval in the input area, as shown in Fig. 6 and Fig. 7.
- When an article has been received it needs to be forwarded to both proof readers and graphics artists. This corresponds to a multiple output node that activates all of its outputs concurrently. This is called an ALL_OUT node and is depicted with an oval in the output area, as shown in Fig. 5 and Fig. 7.

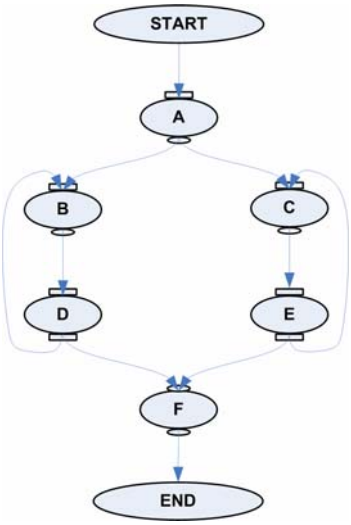

Fig. 8. The graph for the workflow of the example

Using the above, the workflow of the example is modeled as shown in Fig. 8. Of course, in cases where only one input or only one output is connected to a node, there is more than one type of node that can be utilized. For reasons that will become obvious in the following section where the workflow validation methodology will be presented, SINGLE_IN and ALL_OUT are preferred options for the nodes and are always preferred when it does not make any difference.

3 Workflow validation

When a user is allowed to specify workflows in a graphical manner, by providing graphs such as the one depicted in Fig. 8 (or much more complex) using a graphical editing tool, there is always the possibility of a logical error. Therefore, a methodology needs to be developed for the identification of such erroneous cases. In the following we explain how the inspection of the transitive closure of workflow graphs can be utilized to automate the process.

2.1 SINGLE_IN/ALL_OUT case

Having a workflow consisting of nodes of which any input can activate a node and nodes that activate all of their outputs, a simple connectivity test can assure that the workflow is correct. Since the mathematical counterpart of network connectivity is transitivity of relations, we will utilize the operation of transitive closure in order to evaluate the validity of workflows.

In order to best demonstrate this we present the example workflows of Fig 9. In Table 1 we present the matrix representations of the graphs of Fig. 9. The transitive closure of these matrices is presented in Table 2. The connectivity checks that need to be made are:

- do all states lead to successful completion of the task?
- are all states meaningful?

In order to assess whether a state leads to the completion of the task we need to assess whether it leads to the end state. Therefore, we require that all nodes lead to the end node, i.e. that the last column in the transitive closure matrix is filled with ones except for the last row For example, we can see in the transitive closure of the first graph that node C does not lead to completion.

In order to assess whether a state is meaningful we need to assess whether it can be reached. Therefore, we require that all nodes are subsquent to the start node, i.e. that the first row in the transitive closure matrix is filled with ones (first position excluded). For example, we can see in the transitive closure of the second graph that node C is not reachable.

Table 1. Matrix representations of Fig 9. graphs

	START	A	B	C	END
START	0	1	0	0	0
A	0	0	1	1	0
B	0	0	0	0	1
C	0	0	0	0	0
END	0	0	0	0	0

	START	A	B	C	END
START	0	1	0	0	0
A	0	0	1	0	0
B	0	0	0	0	1
C	0	0	0	0	1
END	0	0	0	0	0

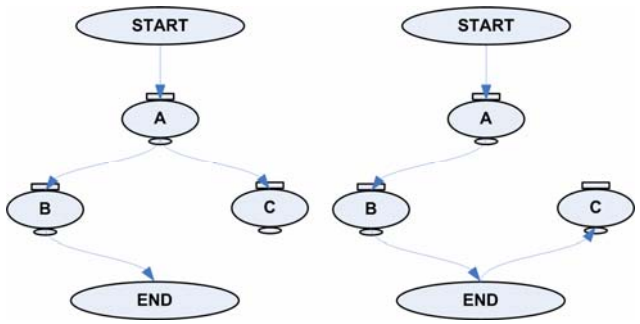

Fig. 9. SINGLE_IN/ALL_OUT checks

Table 2. Transitive closure of the two matrices

	START	A	B	C	END
START	0	1	1	1	1
A	0	0	1	1	1
B	0	0	0	0	1
C	0	0	0	0	0
END	0	0	0	0	0

	START	A	B	C	END
START	0	1	1	0	1
A	0	0	1	0	1
B	0	0	0	0	1
C	0	0	0	0	1
END	0	0	0	0	0

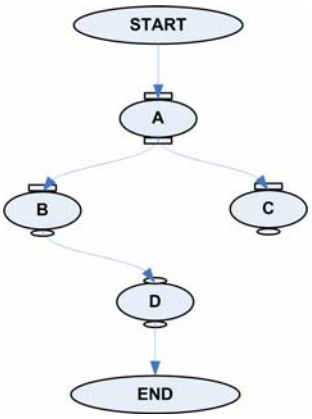

Fig. 10. An example including SINGLE_OUT and ALL_IN graphs

2.2 SINGLE_OUT and ALL_IN cases

When not all nodes in the workflow are of the SINGLE_IN/ALL_OUT type a simple transitive closure is not enough to ensure validity. See for example the workflow graph of Fig. 10. Although a transitivity check would indicate that there is nothing wrong with this workflow, a more careful examination reveals that this workflow will never lead to a successful completion of its task. Due to the fact that both nodes B and C are required for the activation of node D but only one of them may be activated by node A.

Since the transitive closure methodology can only handle correctly the case of SINGLE_IN/ALL_OUT nodes, we need to transform the workflow graphs of the more general cases to graphs comprising only SINGLE_IN/ALL_OUT nodes, so that the methodology can be applied.

As far as the SINGLE_OUT nodes are concerned, these are handled by creating multiple copies of the graph; one for each activated output. In order for the workflow to be validated, each node needs to be found to be meaningful on at least one graph and leading to the end on at least one graph; there is no need for these to happen on the same graph.

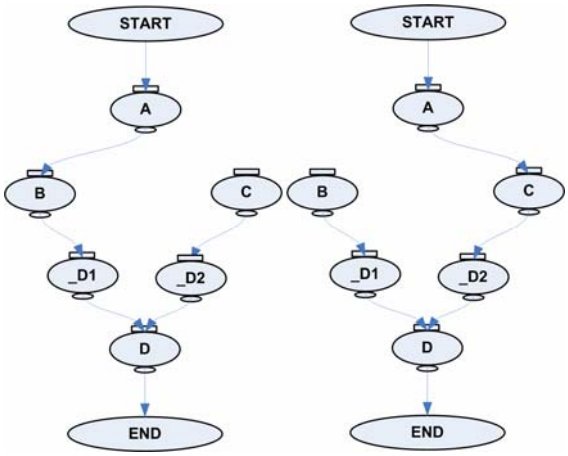

Fig. 11. The transformed multiple graph problem

When it comes to ALL_IN nodes, we need to have a means of making sure that all possible inputs are activated at the same time. In order to avoid confusions in the case of multiple output and multiple input nodes that are linked together in more than one ways, we utilize the following approach: Extra "dummy" nodes are added to the graph, one for each required input of an ALL_IN node. The node is found to be meaningful only if all of the dummy nodes associated with it are activated at the same time, i.e. in the same transitive closure graph.

The validation of the workflow presented in Fig. 10 is transformed into the multiple graph validation problem presented in Fig 11. In the transitive closure of the two graphs (Table 3) we can see that although all nodes are found to be reasonable and terminating in at least one of the two matrices, the two dummy nodes are not activated simultaneously on either of the two examined instances of the problem. Therefore, we correctly conclude that there is a problem with the activation of node D, and thus the workflow of our example is not validated.

4 Experimental results

The methodology presented in this paper has been coded using the Python programming language.Input, in the current stage of the implementation, is provided in the form of text files, such as the one presented bellow:

```
name=S;connections=A;intype=none;outype=ALL_OUT;
name=A;connections=B,C;intype=SINGLE_IN;outype=SINGLE_OUT;
name=B;connections=D;intype=SINGLE_IN;outype=ALL_OUT;
name=C;connections=D;intype=SINGLE_IN;outype=ALL_OUT;
name=D;connections=nE;intype=ALL_IN;outype=ALL_OUT;
name=E;connections=none;intype=SINGLE_IN;outype=null;
```

Table 3. Transitive closures for the multiple graph problem

	START	A	B	C	_D1	_D2	D	END
START	0	1	1	0	1	0	1	1
A	0	0	1	0	1	0	1	1
B	0	0	0	0	1	0	1	1
C	0	0	0	0	0	1	1	1
_D1	0	0	0	0	0	0	1	1
_D2	0	0	0	0	0	0	1	1
D	0	0	0	0	0	0	0	1
END	0	0	0	0	0	0	0	0

	START	A	B	C	_D1	_D2	D	END
START	0	1	0	1	0	1	1	1
A	0	0	0	1	0	1	1	1
B	0	0	0	0	1	0	1	1
C	0	0	0	0	0	1	1	1
_D1	0	0	0	0	0	0	1	1
_D2	0	0	0	0	0	0	1	1
D	0	0	0	0	0	0	0	1
END	0	0	0	0	0	0	0	0

The workflow described in this example is actually the workflow presented in Fig. 10. The output of the module for this example is provided bellow.

```
@!ERROR: ALL_IN check failed for D
@!The Workflow is not Valid
```

Using another sample workflow with the following properties our validator produces correct results

```
name=S;connections=A;intype=none;outype=ALL_OUT;
name=A;connections=B,C;intype=SINGLE_IN;outype=ALL_OUT;
name=B;connections=D,E;intype=SINGLE_IN;outype=SINGLE_OUT;
name=C;connections=F;intype=SINGLE_IN;outype=ALL_OUT;
name=D;connections=H;intype=SINGLE_IN;outype=ALL_OUT;
name=H;connections=G;intype=SINGLE_IN;outype=SINGLE_OUT;
name=F;connections=G,E;intype=ALL_IN;outype=SINGLE_OUT;
name=G;connections=E;intype=ALL_IN;outype=SINGLE_OUT;
name=E;connections=none;intype=SINGLE_IN;outype=ALL_OUT;
```

Producing the following output

```
@!ERROR: ALL_IN check failed for G Possible Workflow #0
@! Workflow is not valid
```

By changing the output type of nodes F and B to ALL_OUT, eliminating the problem of G not being activated in the previous cases, the validator produces the following result guarantying us a valid workflow.

```
@! Workflow is valid
```

5 Conclusions

In this paper we have focused on the problem of automatic workflow validation. As we have explained, the validation of workflows can be a tricky task, due to the fact that different types of nodes can be associated with a workflow, some of which are quite difficult to handle.

After providing a formal model for workflows and following a transitive closure based approach, we have provided a methodology for the automated validation of workflows comprising SINGLE_IN/ALL_OUT nodes. Then, through problem transformation, we explained how the same methodology can be utilized to also treat the more general cases. The proposed methodology has been tested theoretically, but also practically through implementation into a working stand-alone module.

What remains to be done is the integration of the developed module and methodology with the graphical workflow editor. This will also open the way for real life application and testing under heavier loads. One point that is expected to be of augmented interest is that of the operation of transitive closure, as the complexity of the conventional approach is too high to consider for real time operation under heavy load. In this direction, novel transitive closure approaches can be applied [6,7], taking advantage the sparse nature of the workflow graphs.

References

1. A. P. Barros, A. H. M. ter Hofstede, Towards the construction of workflow-suitable conceptual modelling techniques, Information Systems Journal 8 (4), 313–337, 1998.
2. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski and A. P. Barros, Workflow Patterns, Distributed and Parallel Databases 14(1), pp. 5-51, 2004.
3. W. M. P. van der Aalst, M. Weske, G. Wirtz, Advanced topics in workflow management: issues, requirements, and solutions, Journal of Integrated Design and Process Science 7(3), 2003.
4. M. M. Compton, S. Wolfe, Intelligent validation and routing of electronic forms in adistributed workflow environment, Proceedings of the Tenth Conference on Artificial Intelligence for Applications, 1994.
5. S. Sadiq, M. Orlowska, W. Sadiq, C. Foulger, Data flow and validation in workflow modeling, Proceedings of the 15th Australasian database conference, 2004.
6. M. Wallace, S. Kollias, Two Algorithms For Fast Incremental Transitive Closure Of Sparse Fuzzy Binary Relations, International Journal of Computational Methods, in press.
7. M. Wallace, Y. Avrithis, S. Kollias, Computationally efficient sup-t transitive closure for sparse fuzzy binary relations, Fuzzy Sets and Systems 157(3), pp. 341-372, 2006.