Using partially defined workflows for course modelling in a learning management system

Boris Milašinović
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
boris.milasinovic@fer.hr

Krešimir Fertalj
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
kresimir.fertalj@fer.hr

ABSTRACT

Decoupling the presentation logic, business logic and workflow in a complex system is essential to keep the system flexible. Modelling workflow in complex systems such as learning management systems (LMS) could be supported by creating partially defined workflows and producing the algorithms that will merge them into an integral workflow. This paper presents some issues that arise in the implementation of a LMS flow control module using partially defined workflows in existing workflow management software (Windows Workflow Foundation) and proposes an algorithm for its resolution. The research is still in progress and only the problems with parallel splits and synchronizations in flow control have been described.

Keywords: Workflow management, partially defined workflows, workflow merge, Windows Workflow Foundation

1. Introduction

During the development of a learning management system (LMS) and during its use there are lots of issues that require attention. The focus is put mostly on the graphical design of course materials. Course authors tend to create graphically rich and interactive materials and organize them into appropriate topics and lessons. In case that something beyond the presentational tool is being developed, it is usually the module for knowledge assessment since a system that has various assessment types entices a more active approach to learning [2].

But, after that, one of the important issues is when, to whom and in which order to present particular lesson or a course. Courses are usually organized in a tree form where the predecessors of a particular node in the tree are the courses that student had to pass in order to take the next course. As an LMS develops through time the flow logic changes continuously. As noted in [1] the design of e-learning systems is largely conducted on an intuitive, ad hoc basic, resulting in inefficient systems that defectively support the learning process. In order to make an e-learning system more process oriented, to maintain its quality and manageability, decoupling the presentation and business logic (what and how to present, how to manage data) and workflow (when and in which order to present) is in authors' opinion one of the crucial things that need to be done. One use of a workflow within an e-learning system already has already been demonstrated in [3] and [4] but was limited mostly to sequence workflows. In contrast to these practical examples SCORM [8] as a standard [9] for course material design includes SCORM Navigation and Sequencing as a schema definition for common patterns in LMS flow control, but leaves the implementation problems to the system developers.

This paper presents some issues that arise in the implementation of a LMS flow control module using existing workflow management software (Windows Workflow Foundation [10], in further text WF) and proposes an algorithm for its solution.

2. Partially defined workflows

A LMS can be extended with a module for course workflow management. In this way course creators could manage course workflow more easily. In a large LMS, or during the study curriculum design, it is reasonable to expect that more than one person will define courses and manage relationship between them.

The basic idea behind the partially defined workflows is that the system should allow particular course designers to design their own courses and to connect them with the prerequisite courses. After all users have defined their courses and their prerequisites, the system should merge them and produce the final workflow. The situation where courses C and D are required for course F, courses B and C are required for course E etc is illustrated in Figure 1. Course relationships in the picture are described using directed graphs.

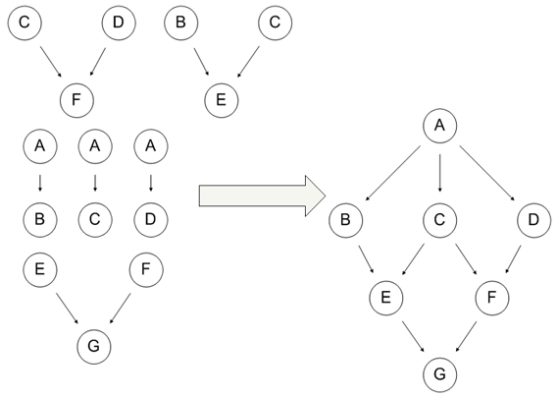

Figure 1: Partially defined relationships between courses merged into the integral graph model

Transforming the graph representation into a real workflow model is not a simple task. Although the graph shown in the Figure 1 is simple, it clearly depicts the problem. It has one parallel split (A to B, C and D) and three synchronizations (at vertices E, F and G). Although it can be supported by the custom workflow management system (as shown in [5]) most of the workflow languages expect that each parallel split is paired with the corresponding synchronization. Therefore it is impossible to support graph from the Figure 1 without modifications.

For instance, WF, similarly to other workflow languages, does not allow direct connections between the elements in two parallel branches. In this paper the authors propose cloning common courses and putting them into more than one parallel branch. Concrete implementation of the workflow model will ensure that those clones are shown as one course during the runtime. Creation of such workflow is done using the algorithm described in the following section.

3. The workflow creation algorithm

As shown in Figure 2, the algorithm has several steps. In the beginning partially defined workflows (graphs) have to be merged. Courses and their relationships are shown as a directed graph where an arc going from node A to node B means that course A is the predecessor of the course B.

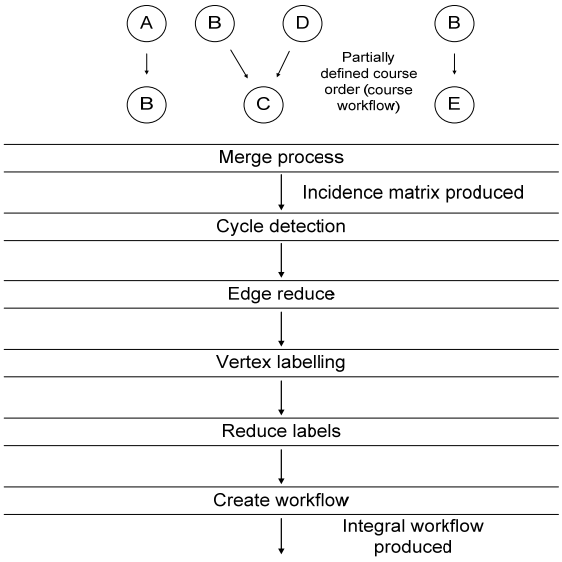

Figure 2: Phases of the workflow creation algorithm

The efficient way of doing the initial merge process is to represent the graph with an incident matrix. Course names have to be uniquely translated to the positive numbers ranging from 1 and the number of the courses. After that, the creation of the incidence matrix is quite easy. Value at row *i* and column *j* will be 0, -1, or 1 depending on whether the arc between vertices *i* and *j* does not exist,

goes from *i* to *j*, or goes from *j* to *i* respectively.

3.1 The edge reducing algorithm

After the incidence matrix has been created cycle detection using modification of the topological sort described in [6] and [7] must be done. If the directed graph does not contain cycles, the algorithm can proceed to the edge reduction process. Basically, if for some edge e = (A,B) an alternative path from vertex A to vertex B exists then the edge e can be removed because it is obsolete (edge e = (A,B) in Figure 3 is removed since A is a transitive predecessor of B via C).

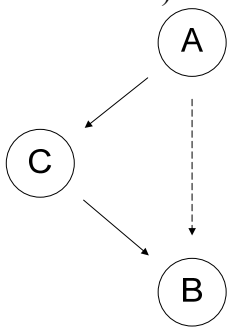

Figure 3: Removing obsolete edge from A to B

After the edge reducing algorithm, two new vertices have to be added to the graph: *Start* and *End*. *Start* vertex will be connected to all vertices having inbound degree equal to zero (in such way that *Start* is theirs predecessor) and all vertices having outbound degree equal to zero will be connected with the *End* vertex (in such way that *End* is theirs successor). That way, it has been assured that graph is connected which is essential for the labelling process illustrated in the next section.

3.2 The vertex labelling algorithm

The vertex labelling algorithm can be applied to the directed graph G=($\mathcal{V}$, E) with the set of nodes $\mathcal{V}$, and set of directed arcs E with the following presumptions:

- Graph G is connected and there are no cycles in the graph
- There is only one start vertex $v \in \mathcal{V}$ such that its inbound degree $in(v) = 0$
- There is only one end vertex $v \in \mathcal{V}$ such that its outbound degree $out(v)=0$
- For each pair of nodes v_1 and v_2 such that exist $e \in E$ and such that $e = (v_1,v_2)$ there is no any other path $v_1,v_a,v_b,...v_2$ from v_1 to v_2.

The labelling function L: $\mathcal{V} \rightarrow \mathcal{L}$ where $\mathcal{L}$ is set of labels (strings that contain only numbers and dots) will assign one or more labels to each vertex in the graph. Labelling algorithm starts from the graph end vertex. Labels of vertex that has only one predecessor are added to the set of the predecessor's labels without being changed. If a vertex has two or more predecessor then its set of labels is added (with modifications) to the set of each predecessor labels. During that process each label is concatenated with dot and order number, where the order number goes from 1 to the number of predecessors. Formally these steps can be written as follows.

1. Set the end vertex as the current vertex *curr* and label it with '1'
   (L(*curr*) = {'1'}) and set O ={ *curr* }

2. Set the *curr* vertex to be the first node from the set O. Let the
   S = { v | such that exists e =(v, *curr*)}.
   Set O = (O ∪ S) \ {*curr*}
   a. If the |S| = 1 then L(v) = L(v) ∪ L(*curr*) where v∈S
   b. If the |S| > 1 then for each $v \in$ S L(v) = L(v) ∪ concat(L(*curr*), '.i') where i goes from 1 to |S| respectively

3. Repeat step 2 until O = ∅

3.3 The label reduction algorithm

Function *level*: $\mathcal{L} \rightarrow \mathbb{N}$ is defined as the number of the dots inside a label. Label l is the parent of label m if *level*(m) = *level*(l) + 1 and m and l are the same until the last dot in both labels. The label reduction algorithm takes labels in the defined order and for each label l finds the nodes that contain all labels that have l as parent. For such nodes those labels are replaced with label l. Formally, the algorithm steps can be written as follows.

1. *curr_level* = $\max_{l \in \mathcal{L}} level(l)$ - 1

2. Let the $\mathcal{L}' = \{l \in \mathcal{L} \mid level(l) = curr\_level\}$
   For each label *l* from $\mathcal{L}'$ do
   $\mathcal{L}'' = \{m \in \mathcal{L}' \mid l \text{ is parent of } m\}$
   If $\mathcal{L}'' \neq \emptyset$ then
   $\mathcal{V}' = \{v \in \mathcal{V} \mid m \in \mathrm{L}(v), \forall m \in \mathcal{L}''\}$
   For each $v \in \mathcal{V}'$ do
   $\mathrm{L}(v) = \mathrm{L}(v) \setminus \mathcal{L}'' \cup \{l\}$

3. *curr_level* = *curr_level* – 1
   if *curr_level* ≥ 0 repeat step 2

After the labels have been reduced, a workflow model can be created. For each vertex, its labels represent the names of the parallel branches in the workflow model. Cardinality of particular labels set determines how many clones of each vertex should be created and added to corresponding parallel branches.

4. An example

For the sake of simplicity enumeration function will be skipped and courses will be named with numbers from 1 to 10. Relationships between courses are as follows.

- To take course 1 course 4 must be taken
- To take course 2 courses 1 and 4 must be taken
- To take course 3 course 5 must be taken
- To take course 5 courses 1, 2 and 8 must be taken
- To take course 6 courses 5 and 10 must be taken
- To take course 7 courses 4, 5, 8 and 10 must be taken
- To take course 8 course 1 must be taken
- To take course 10 courses 5 and 9 must be taken

Matrix shown in Figure 4 is the incidence matrix for these relationships.

$$\begin{bmatrix} 0 & -1 & 0 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 4: Incidence matrix before the edge reducing algorithm

The edge reduction algorithm removes the following edges: edge from 1 to 5 because there exits alternative path (1→8→5), edge from 4 to 2 (because of 4→1→2), edge from 4 to 7 (because of 4→1→8→5→10→7), edge from 5 to 6 (because of 5→10→6), edge from 5 to 7 (because of 5→10→7) and edge from 8 to 7 (because of 8→5→10→7). The matrix from Figure 5 is the new incidence matrix after the aforementioned edges have been removed.

$$\begin{bmatrix} 0 & -1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 5: Incidence matrix after the edge reducing algorithm

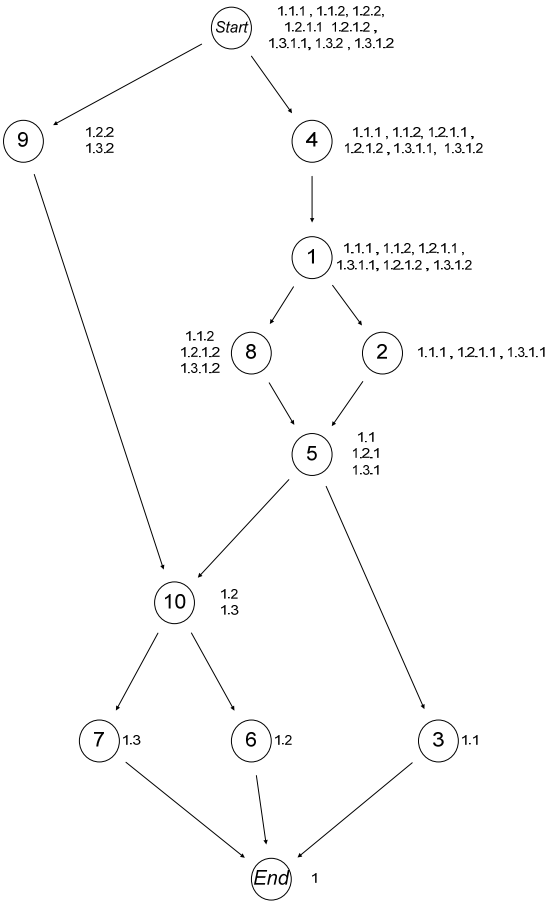

Figure 6: Labelled graph after vertex labelling algorithm (before label reduction algorithm)

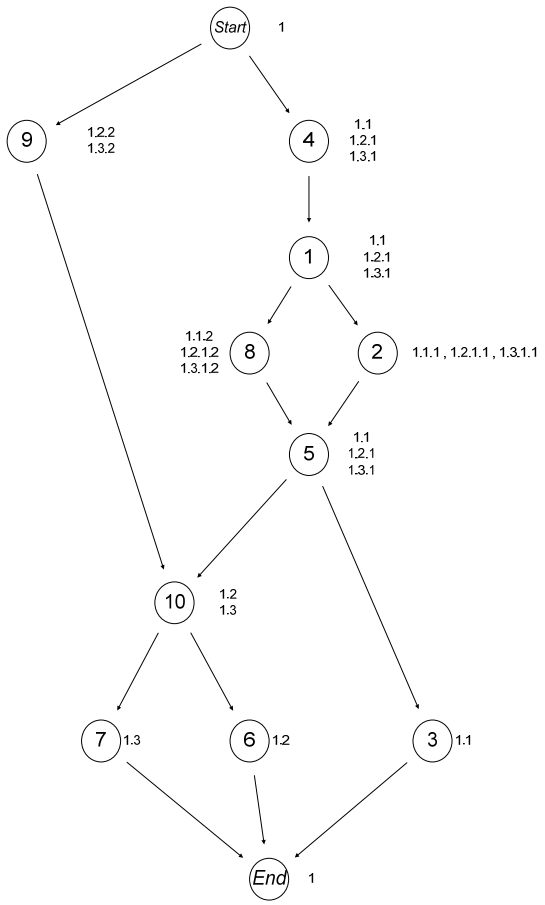

Figure 7: Labelled graph after label reduction algorithm

A newly created vertex Start is connected with vertices having inbound degree equal to zero (vertices 4 and 9) and vertices having outbound degree equal to zero (3, 6 and 7) are connected with the End vertex. One of the possible topological sorts is: Start, 4, 9, 1, 2, 8, 5, 3, 10, 6, 7, End.

Figure 6 and Figure 7 shows the labelled graph after the labelling algorithms and after the reduction of labels. The resulting WF model is presented in Figure 8 where elements in the given model are named in form *C_{node number}*. In case some nodes had to be cloned, *inst_{clone instance number}* is appended to the node name in order to have unique node names.

5. Conclusion

Dividing the prerequisite course model in an LMS or in a study curriculum into the set of partially defined workflows where each partially defined workflow represents parts of a course order helps maintaining course order and increase readability.

Nevertheless, as shown in Figure 1 for already simple models it can be impossible to directly transform them into a real workflow model in an existing workflow modelling language. The paper presented an algorithm for the integration of partially defined workflows, which define a prerequisite course model in an LMS or in a study curriculum.

Although the research is still in progress and only simple parallel splits have been used so far, the basic idea of presented algorithm, in authors' opinion, makes a solid foundation for the future developments. The future work will tend to broaden supported set of workflow patterns and to establish the framework for partially defined models.

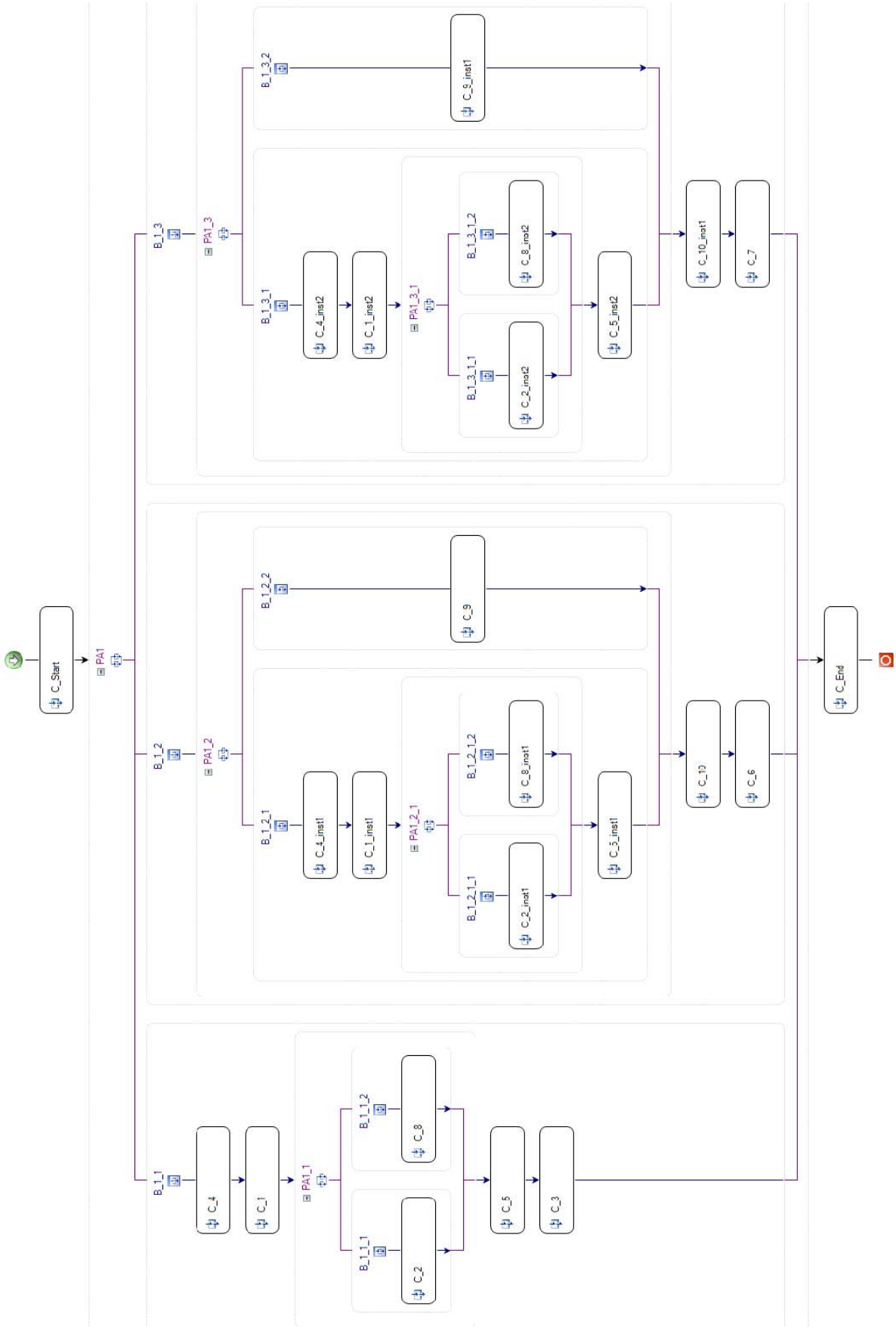

Figure 8: WF model for the graph from Figure 7.

References

[1] P. Avgeriou, S. Retalis, N. Papaspyrou, Modelling learning technology systems as business systems. Software and Systems Modelling, Volume 2, Number 2, 2003, pp 120-133

[2] I. Botički, B. Milašinović, Knowledge Assessment at the Faculty of Electrical Engineering and Computing, In the Proceedings of the 30th International Conference on Information Technology Interfaces, Cavtat, Croatia, 2008, pp 111-116

[3] M. Cesarini, M. Monga, R. Tedesco, Carrying on the e-Learning process with a Workflow Management Engine, In the Proceedings of the 2004 ACM symposium on Applied computing, Nicosia, Cyprus, 2004, pp 940-945

[4] J. Lin, C. Ho, W. Sadiq, M.E. Orlowska, Using Workflow Technology to Manage Flexible e-Learning Services, Educational Technology & Society 5(4), 2002, pp 116-123

[5] B. Milašinović, K. Fertalj, I. Nižetić, On some Problems while Writing an Engine for Flow Control in Workflow Management Software, In the Proceedings of the 29th International Conference on Information Technology Interfaces, Cavtat, Croatia, 2007, pp 489-494

[6] B.R. Preiss, Testing for Cycles in a Directed Graph, Data Structures and Algorithms with Object-Oriented Design Patterns in C#, http://www.brpreiss.com/books/opus6/html/page565.html [2009/04/24] (original content from Data Structures and Algorithms with Object-Oriented Design Patterns in C++, John Wiley & Sons, 1998)

[7] B.R. Preiss, Topological Sort, Data Structures and Algorithms with Object-Oriented Design Patterns in C#, http://www.brpreiss.com/books/opus6/html/page558.html [2009/04/24] (original content from Data Structures and Algorithms with Object-Oriented Design Patterns in C++, John Wiley & Sons, 1998)

[8] SCORM, Home Page http://www.adlnet.org/Technologies/scorm/default.aspx [2009/04/24]

[9] Standard formats for courseware, Croatian Academic and Research Network Reference Centre, Evaluation of courseware, http://wwww.carnet.hr/referalni/obrazovni/en/oca/standards [2009/04/24]

[10] Windows Workflow Foundation, http://msdn.microsoft.com/en-us/netframework/aa663328.aspx [2009/04/24]