Structural Correctness of Planning Processes in Building Engineering

Axel Klinger, Markus König, Volker Berkhahn
Institute of Computer Science in Civil Engineering
University of Hannover, Germany
{klinger, koenig, berkhahn}@bauinf.uni-hannover.de

Summary

The planning of projects in building engineering is a complex process which is characterized by a dynamical composition and many modifications during the definition and execution time of processes. For a computer-aided and network-based cooperation a formal description of the planning process is necessary. In the research project "Relational Process Modelling in Cooperative Building Planning" a process model is described by three parts: an organizational structure with participants, a building structure with states and a process structure with activities. This research project is part of the priority program 1103 "Network-Based Co-operative Planning Processes in Structural Engineering" promoted by the German Research Foundation (DFG).

Planning processes in civil engineering can be described by workflow graphs. The process structure describes the logical planning process and can be formally defined by a bipartite graph. This structure consists of activities, transitions and relationships between activities and transitions. In order to minimize errors at execution time of a planning process a consistent and structurally correct process model must be guaranteed. This contribution considers the concept and the algorithms for checking the consistency and the correctness of the process structure.

1 Introduction

The modelling of planning processes is a dynamical process, where many changes, extensions and updates are made during the planning process. A workflow graph, which describes a planning process, must be consistent and structurally correct. This article is dealing with the structural correctness of workflow graphs and methods to detect structural incorrectness.

After a short introduction of workflow graphs, criteria are shown for which a workflow graph is structurally correct. The structural correctness is checked by instance subgraphs (van der Aalst 2002) (König 2004). In this paper the definition of instance subgraphs and an algorithm to find the instance subgraphs of a workflow graph are explained in detail.

Normally, every activity in a planning process is executed only once. In this case the process structure is acyclic. If a sequence of similar activities is planned, these activities can be transformed into cycles of activities. While most authors restrict their algorithms for acyclic graphs, this paper also covers graphs with cycles.

2 Structure of Workflow Graphs

A workflow graph contains a set of activities and a set of transitions. The relations between activities and transitions are described by a bipartite graph and a set of rules.

Activity: An activity describes a set of planning tasks and is handled by one participant or a group of participants. It can only be carried out if all predecessor activities are completed.

Transition: A transition describes a relationship between activities as an object.

Bipartite Structure: A workflow graph is formally described by a bipartite graph. The activities and transitions of the workflow form the nodes of the graph. The directed relations between activities and transitions are described by edges of the graph.

$$W := (A, T; R, Q) \qquad R \subseteq T \times A \qquad Q \subseteq A \times T \tag{1}$$

A	Set of activities
T	Set of transitions
R	Set of relations between activities and transitions
Q	Set of relations between transitions and activities

Rules: In order to model workflow graphs the following rules are introduced to realize parallel or alternative execution of activities and transitions. These rules, illustrated in Figure 1, form the basis to check the structural correctness of the workflow graph.

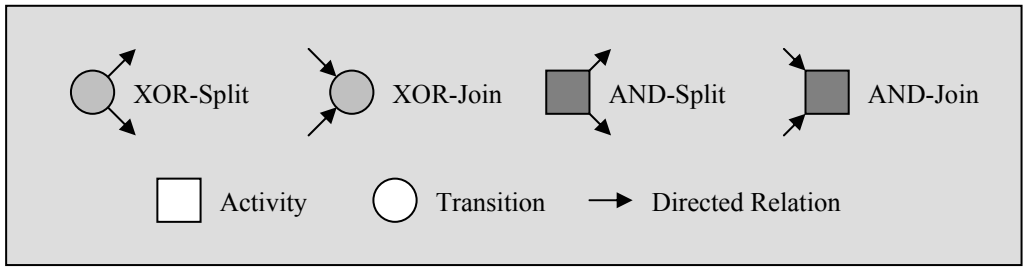

Figure 1: Rules for modelling parallel and alternative processes

1. **Decision (XOR-split):** A decision is modelled if a transition has more than one successor. In this case only one of the following activities can be chosen and will be executed.
2. **Contact (XOR-join):** A contact is modelled if a transition has more than one predecessor. In this case the execution of exactly one of the predecessors must be guaranteed.
3. **Asynchronisation (AND-split):** An asynchronisation is modelled if an activity has more than one successor. In this case all following transitions will be executed.
4. **Synchronisation (AND-join):** A synchronisation is modelled if an activity has more than one predecessor. In this case it must be guaranteed that all predecessors are executable.

Figure 2 shows the usage of rules for parallel and alternative execution of activities in a workflow graph.

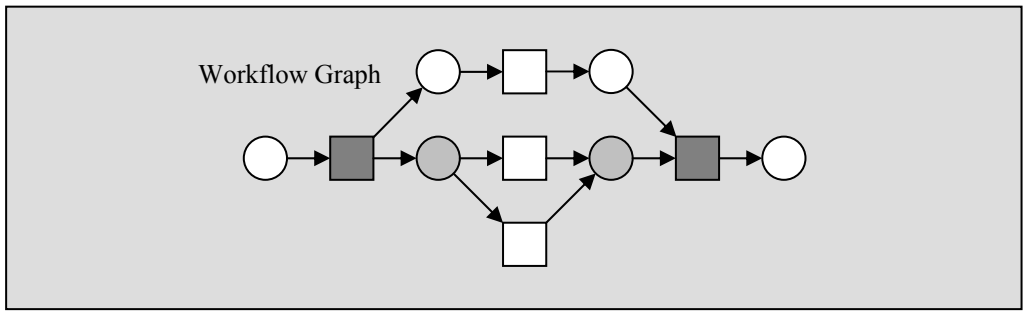

Figure 2: Usage of workflow rules

3 Structural Consistency

A structurally consistent workflow graph has a set of start transitions and a set of end transitions. The attainability of each end transitions must be guaranteed in a workflow graph.

Start transitions: A transition without predecessors is called a start transition. A workflow graph must contain at least one start transition.

End transitions: A transition without successors is called an end transition. A workflow graph must contain at least one end transition.

Attainability: Every activity or transition must be reachable from at least one start transition. From each activity or transition at least one end transition must be reachable.

Workflow component: A workflow graph can be unambiguously divided into strongly connected components. A strongly connected component is a subgraph of the workflow graph and is called a workflow component. A workflow component contains all nodes (activities and transitions), which are reachable from all other nodes of this workflow component. Every node is part of exactly one workflow component. If a workflow component contains more than one node, it is a strongly connected cyclic subgraph of the workflow graph. If every workflow component contains exactly one node, the workflow graph is acyclic.

Reduced workflow graph: A workflow graph can be mapped to a reduced workflow graph with cyclic workflow components. Every node within the reduced workflow graph represents a workflow component. If a workflow component contains exactly one activity or exactly one transition, the related node in the reduced workflow graph is an activity or a transition. If a workflow component contains more than one node, the related node in the reduced workflow graph is a transition under the rules for structural correctness (see section 6). Every relation between two nodes of different workflow components is transferred to the reduced workflow graph. The workflow graph is structurally consistent if the reduced workflow graph has a bipartite structure.

4 Structural Correctness

A consistent workflow graph is structurally correct, if from exactly one start transition exactly one end transition is reachable under the workflow rules. If a workflow graph contains decisions as well as asynchronisations, two different structural problems may arise (van der Aalst et al. 2002) (Sadiq and Orlowska 1999): Deadlock and lack of synchronisation.

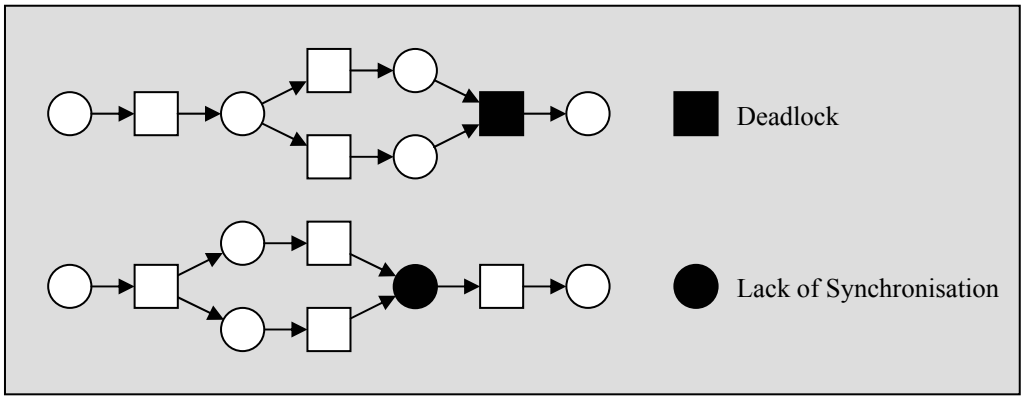

Figure 3: Deadlock and lack of synchronisation

Deadlock: A deadlock as shown in Figure 3 arises, if after a decision alternative activities are merged by a synchronisation. In this case the synchronisation activity can not be executed. Rule 4 is broken.

Lack of synchronisation: A lack of synchronisation as shown in Figure 3 arises, if asynchrony activities are merged by a contact. In this case the following activities would be executed more than once. Rule 2 is broken.

5 Instance subgraphs for acyclic workflow graphs

A consistent workflow graph describes all possible workflows. In an acyclic workflow graph every single possible workflow can be described by an instance subgraph as a subgraph of the workflow graph. The structural correctness of the workflow graph implies the structural correctness of all instance subgraphs.

Instance Subgraph: A workflow graph W can be unambiguously divided into W_i instance subgraphs. Every instance subgraph describes one possible workflow without decisions. According to this, every transition of an instance subgraph except for the end transition has exactly one successor. No instance subgraph is a subgraph of another instance subgraph.

$$W := (A, T; R, Q) \qquad W_i := (A_i, T_i; R_i, Q_i)$$

$$A_i \subseteq A \qquad T_i \subseteq T \qquad R_i \subseteq R \qquad Q_i \subseteq Q$$

$$\text{with} \quad \bigwedge_{t \in T-E} g_S(t) = 1 \tag{2}$$

$$W = \bigcup_{i=1}^{n} W_i \qquad \text{with} \quad \bigwedge_{i=1}^{n} \bigwedge_{\substack{j=1 \\ j \neq i}}^{n} \neg (W_i \subseteq W_j)$$

$g_S(t)$ Number of successors of a transition t
E Set of end transitions

Figure 4 shows a workflow graph which is divided into two instance subgraphs without decisions.

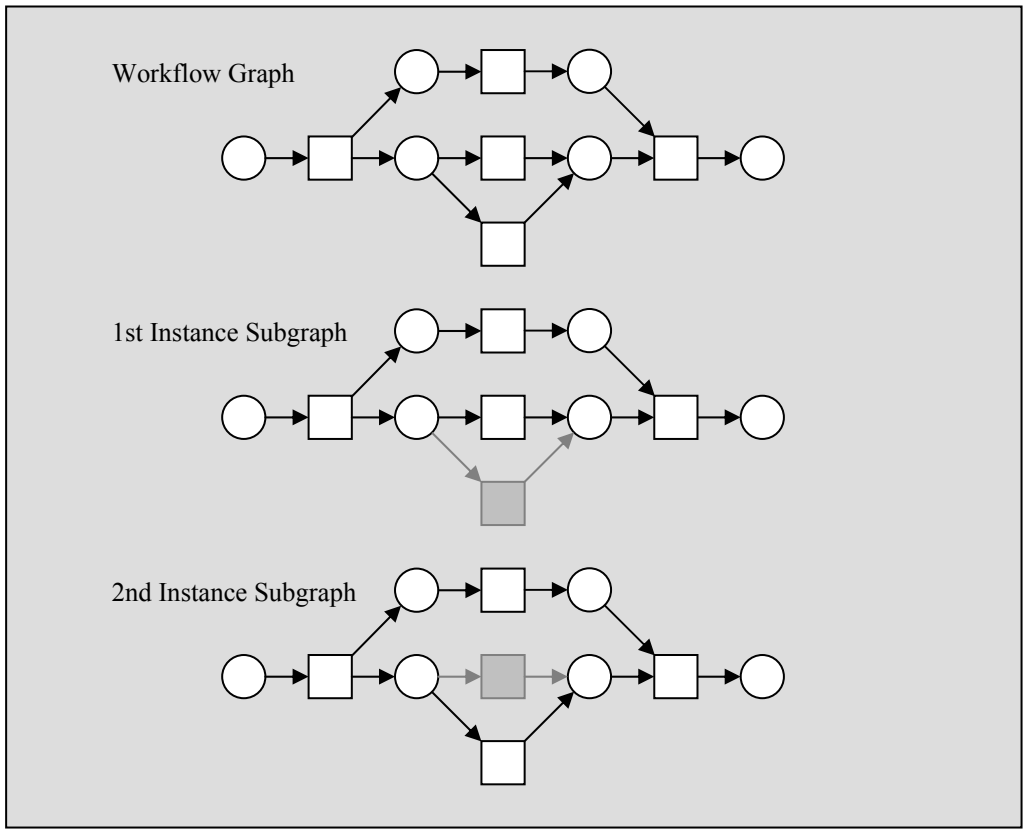

Figure 4: Instance subgraphs

Construction of Instance Subgraphs: An instance subgraph of a workflow graph can be created under the rules 1 and 3. Starting at one start transition, the workflow is traversed. If an activity with more than one successor is reached, the traversal has to be continued for every successor. If a transition with more than one successor is reached, only for one successor the traversal has to be continued. After the traversal of the workflow graph the instance subgraph contains all passed activities and transitions and their relations.

For each start transition and each possible combination of decisions exists one instance subgraph. In order to get all instance subgraphs, efficient algorithms based on breadth-first-traversal and depth-first-traversal (König 2004) can be applied.

Structural Correctness: A cyclic workflow graph is structurally correct, if every instance subgraph is structurally correct. An instance subgraph is structurally correct, if all following conditions are fulfilled:

1. Every transition in the instance subgraph, except for the start transition, has exactly one predecessor.
2. Every activity in the instance subgraph has the same predecessors as in the workflow graph.
3. Every instance subgraph has exactly one end transition.

Figure 5 shows a workflow graph and the corresponding instance subgraphs.

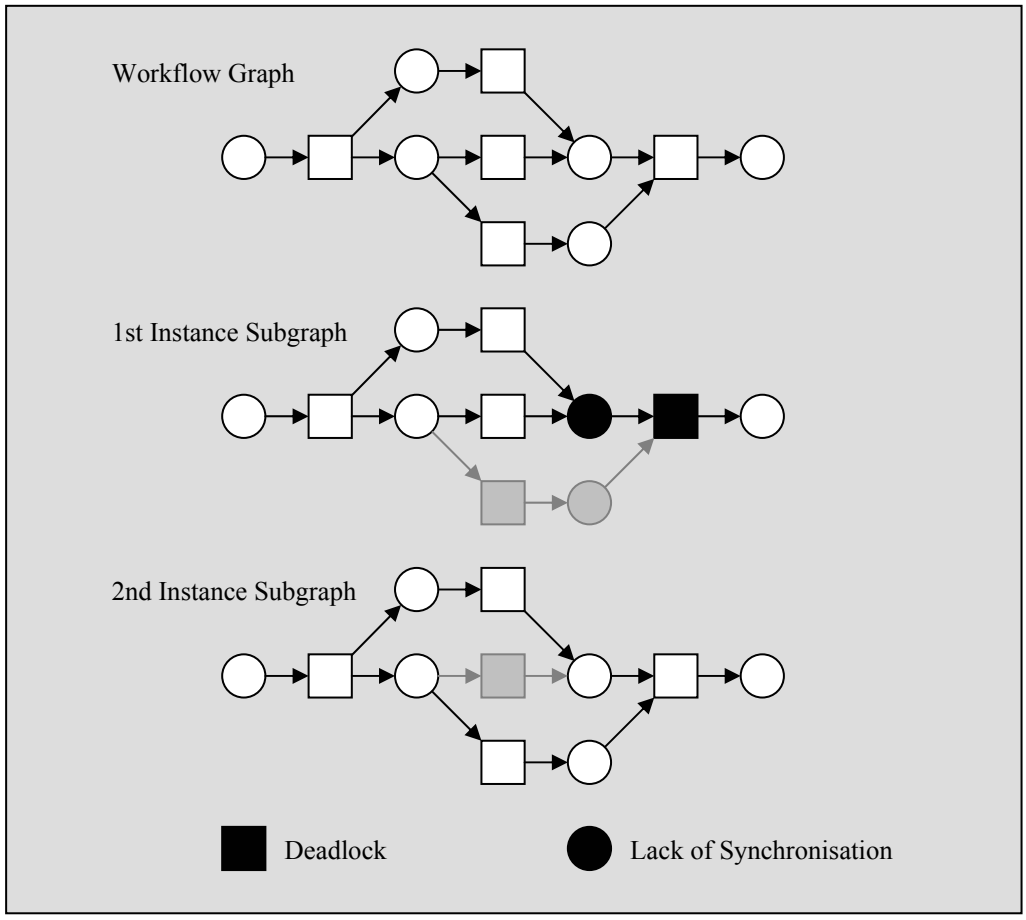

Figure 5: Instance subgraphs with deadlock and lack of synchronisation

If the first condition is not fulfilled, rule 2 for the workflow graph is broken and a lack of synchronisation exists. If the second condition is not fulfilled, rule 4 for the workflow graph is

broken and a deadlock exists. If the third condition is not fulfilled, the workflow graph has no unique end.

6 Instance subgraphs for non-acyclic workflow graphs

The structural correctness of non-acyclic workflow graphs is based on the structural correctness of acyclic workflow graphs, by decomposing the non-acyclic workflow graph into strongly connected components and mapping the graph to a reduced component graph. A non-acyclic workflow graph is structurally correct, if all following conditions are fulfilled.

1. The reduced workflow graph is structurally correct.
2. Every cyclic workflow component is structurally correct.

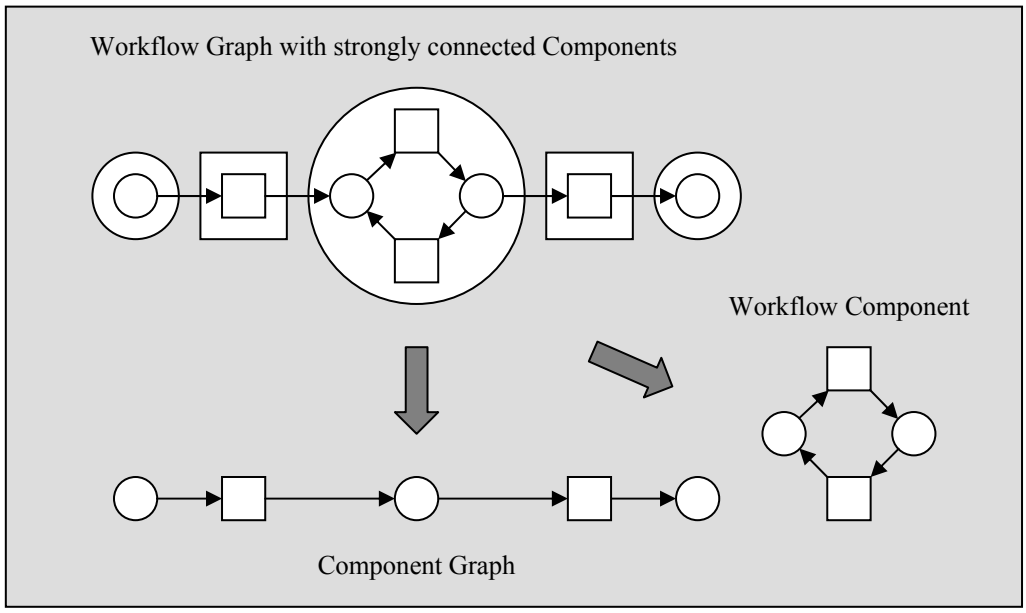

Figure 6: Extracting cycles

Cyclic Workflow Component: A cyclic workflow component is a cyclic subgraph with alternative input transitions and alternative output transitions. Input activities for cyclic workflow components are not allowed, because at least one predecessor is unreachable. Output activities are not allowed, because the following activities could be executed more than once.

Cut of a workflow component: A transition within a component with one or more predecessors from outside of this workflow component is called an input transition. A transition within the component with one or more successors from outside of this workflow component is called an output transition. Each input transition has to be cut and is replaced by one transition with all successors and one transition with all predecessors of the original input transition. Each output transition has to be cut and is replaced by one transition with all successors and one transition with all predecessors of the original output transition. Transitions without predecessors are start transitions. Transitions without successors are end transitions.

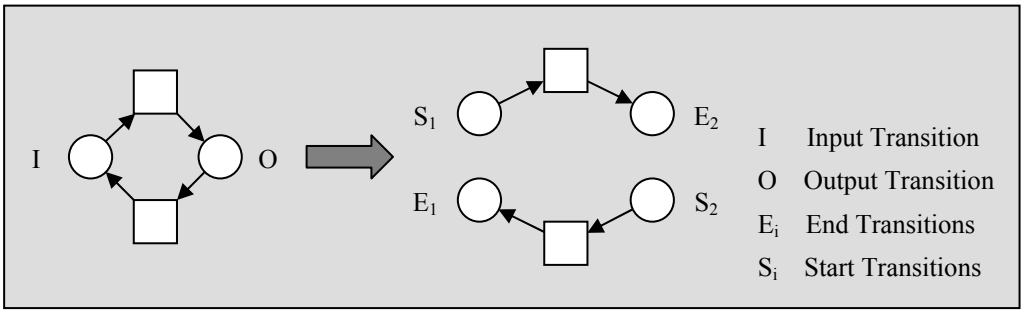

Figure 7: Cut of a cyclic workflow component

When all input and output transitions are replaced, the workflow component is a new workflow graph. If this workflow graph is acyclic, it can be subdivided into instance subgraphs. If the workflow graph is not acyclic, it can be decomposed into strongly connected components and mapped to a reduced component graph. This component graph has to be checked for structural correctness as shown above.

7 Example

The following example shows a consistent workflow graph with two strongly connected components. Both components have some input transitions and some output transitions. This workflow graph is checked with the aforementioned algorithm. The workflow graph is decomposed into strongly connected components and mapped to a reduced component graph.

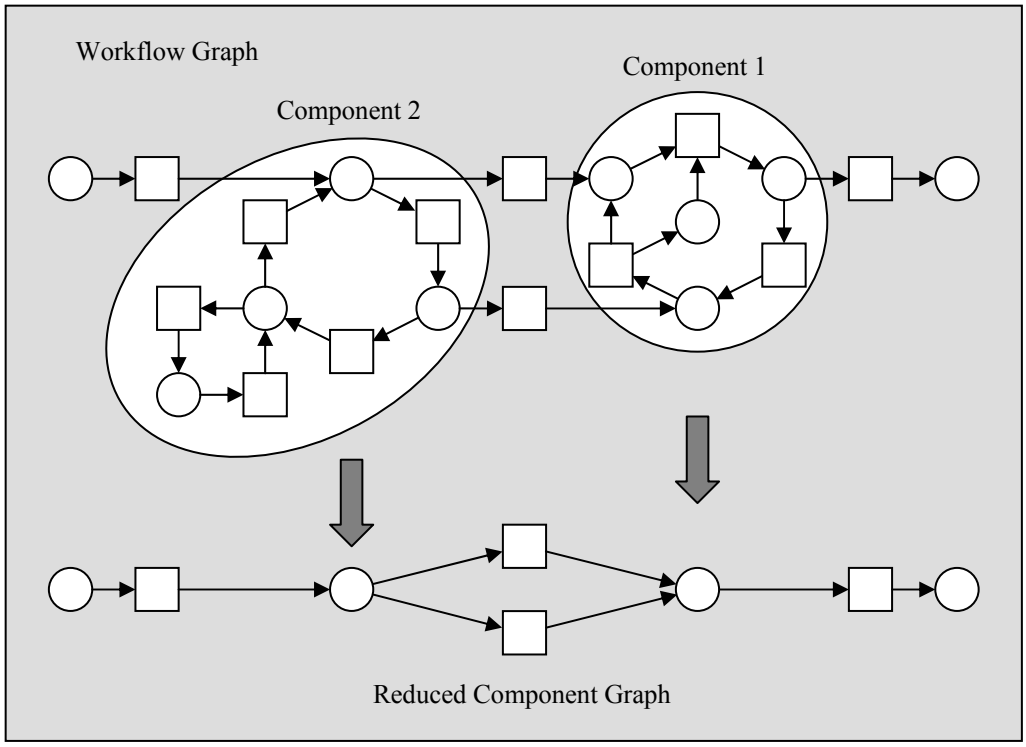

Figure 8: workflow graph and component graph

The components are cut and tested for more strongly connected components. After cutting the input and output transitions of the first component no more strongly connected components exist in this subgraph. Consequently, the component can be checked for structural correctness with instance subgraphs.

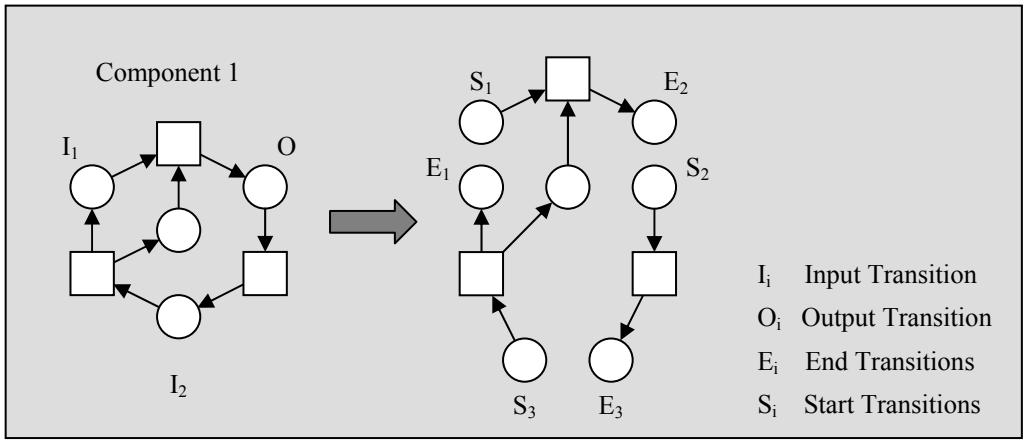

Figure 9: Cut of the first component

The cut component has three start transitions and no decisions. The aforementioned algorithm leads to three instance subgraphs.

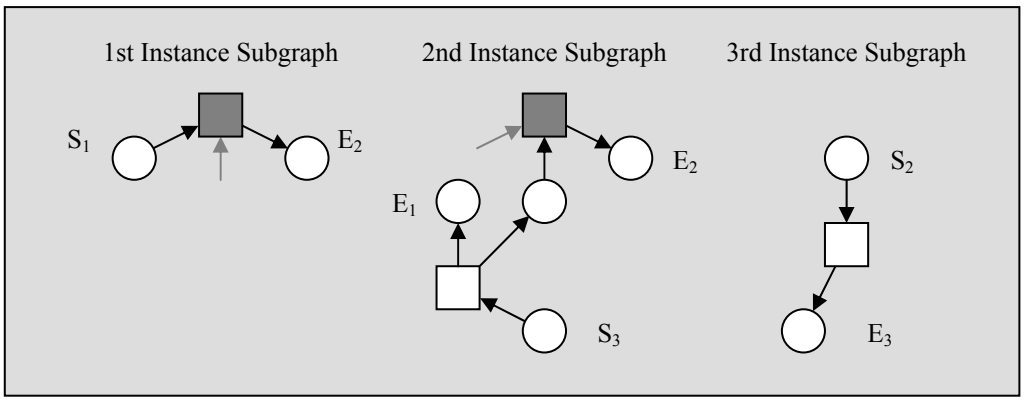

Figure 10: Instance subgraphs of the first component

The first and the second instance subgraph contain a deadlock. In addition the second instance subgraph contains two end transitions. The third instance subgraph has no structural conflict.

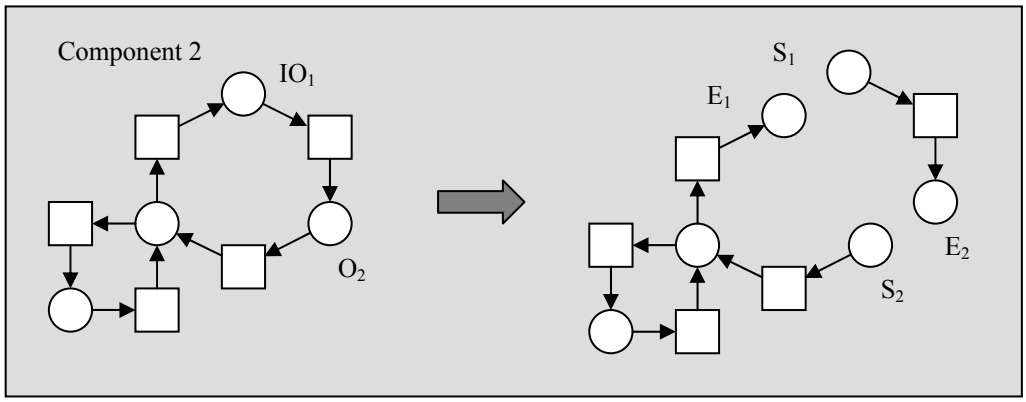

Figure 11: Cutting the second component

After replacing the input and output transitions the second component contains another cycle. This cycle is extracted and the reduced graph of this component can be decomposed into instance subgraphs.

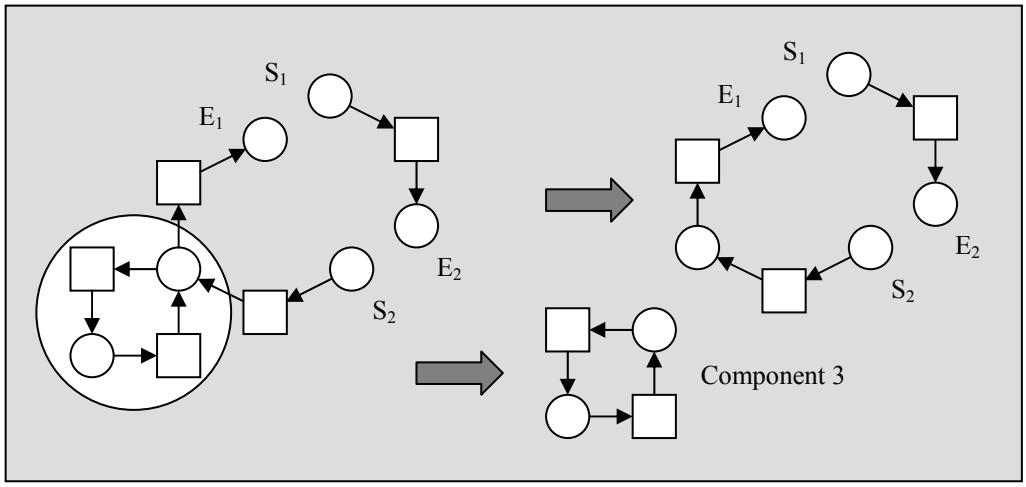

Figure 12: Extracting another cycle

The instance subgraphs of the second and third component now can be checked as shown before. They are free of any structural conflicts.

In this example the workflow graph is not structurally correct, because one of its components is not structurally correct.

8 Conclusion

This paper shows how non-acyclic workflow graphs can be checked for structural correctness by decomposing a reduced workflow graph into instance subgraphs. A reduced workflow graph is acyclic and contains one node for each strongly connected component. Each input and output transition of a strongly connected component is replaced by one start transition and one end transition. The resulting workflow graph of a cut component has to be checked for more strongly connected components and if it does not contain any cycle, the structural correctness is given by the structural correctness of its instance subgraphs. This method yields to the above mentioned algorithm.

9 References

Baumgarten, B. (1996): *Petri-Netze - Grundlagen und Anwendungen*. Spektrum Akademischer Verlag GmbH, Germany.

König, M. (2004): *Ein Prozessmodell für die kooperative Gebäudeplanung.* Dissertation, University of Hannover, Germany.

König, M. and Klinger, A. (2002): *Modellierung von Planungsprozessen mit Hilfe von Hierarchischen Strukturen.* 14. Forum Bauinformatik in Bochum, Fortschrittsberichte Reihe 4, Nr. 181, VDI-Verlag Düsseldorf, Germany.

Damrath, R. and König, M. (2002): *Relational Modelling of Planning Processes in Building Engineering.* In: Proceedings of the 9th International Conference on Computing in Civil and Building Engineering; Taipei, Taiwan.

Pahl, P. J. and Damrath, R. (2001): *Mathematical Foundations of Computational Engineering – A Handbook.* Springer-Verlag, Berlin, Germany.

Van der Aalst, W. M. P., Hirnschall, A and Verbeek, H. M. W. (2002): *An Alternative Way to Analyze Workflow Graphs.* Springer-Verlag, Proceedings of the 14th International Conference

on Advanced Information Systems Engineering (CAiSE´02), volume 2348 of Lecture Notes in Computer Science, pages 535-552, Berlin, Germany.

Desel, J. and Esparza, J. (1995): *Free Choice Petri Nets.* Volume 40 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge.

Sadiq, W. and Orlowska, M. E. (1999): *Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models.* In: M. Jarke and A. Oberweis (editors) Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE´99), volume 1626 of Lecture Notes in Computer Science, pages 195-209, Springer-Verlag, Berlin, Germany.

Lin, H., Zhao, Z., Li, H. and Chen, Z. (2002): *A Novel Graph Reduction Algorithm to Identify Structural Conflicts.* In: Proceedings of the 34th Annual Hawaii International Conference on System Science, IEEE Computer Society Press.