On Business Process Model Transformations *

Wasim Sadiq and Maria E. Orlowska

Distributed Systems Technology Centre
School of Computer Science & Electrical Engineering
The University of Queensland
Qld 4072, Australia
email: {wasim,maria}@dstc.edu.au

Abstract. A business process model represents the basic building block for a workflow-enabled enterprise information system. Generally, a process model evolves through numerous changes during its lifetime to meet dynamic and changing business requirements. It is essential that such changes are introduced systematically and their impact is clearly understood. Process model transformation is a suitable approach for this purpose. Applying pre-defined transformation operations can ensure that the modified process conforms to a given class of constraints specified in the original model. Using a generic process modelling language, we identify three classes of transformation principles – equivalent, imply, and subsume – to manage changes in process models. A simple algebraic notation for representing process graphs is also presented that can be used to reason about transformation operations.

1 Introduction

The use of workflow technology in enterprises has grown substantially during the past few years. It is now considered an appropriate platform for building and integrating component-oriented enterprise systems. The workflow management systems provide a flexible environment to manage process logic of business applications just like database management systems have been providing functionality for managing application data. Data modelling techniques like entity-relationship diagrams are applied to identify data management requirements of business applications. These entity-relationship models are mapped to relational schemas of target database management systems. Similarly, process modelling techniques and tools are applied to capture process-relevant requirements of the business applications in the form of business process models. Workflow management systems take up the responsibility for coordinating business processes on the basis of process models that define workflow tasks and associated coordination constraints.

There are several aspects of a business process model including structural / control flow, data flow, roles, application interfaces, temporal constraints, and others [8]. The structural modelling – also known as control flow modelling – of a process model

* The work reported in this paper has been funded in part by the Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

defines the way a workflow management system would order and schedule workflow tasks. This is the primary and perhaps the most important aspect of a process model. It defines the coordination constraints between workflow tasks and builds a foundation for capturing other aspects of workflow requirements.

Generally, a process model goes through several evolutionary changes during its lifetime to satisfy dynamic and changing business requirements. There are generally two approaches for changing process models to meet such new requirements. We classify them as *constrained* and *unconstrained* approaches respectively.

The unconstrained approach, as the name suggests, does not put any restrictions on the way a process model may be modified. The process designer takes the original process model as a starting point for building a new process model that would incorporate required changes. As an outcome of the modification process, we get a new version of the process model with some arbitrary changes. However, it is generally not possible to sensibly relate the modified model with the original process model it is derived from. Unconstrained approach is useful in situations where there has been significant process evolution and adaptation to business requirements [7] [13].

In contrast, the constrained approach maintains certain relationships between the original and modified process models. Using this approach, the workflow designer introduces changes in the original process model in such a way that the modified process model conforms to a specific class of coordination constraints specified in the original process model. This approach supports a systematic way of introducing process model changes and provides means to reason about their impact.

A suitable approach for introducing constrained changes in the process models is through transformation operations. In this paper, we will identify transformation principles and associated operations for introducing constrained changes in process models. Such transformation operations may be applied to a process model G to transform it into G' such that G and G' still maintain underlying structural relationship with each other. We will be concentrating only on structural / control flow aspects of process models. We will identify three classes of structural relationships – equivalent, imply, and subsume – and associated transformation operations. An analogy between process model transformations and schema transformations in database research is useful here.

Before we present these transformation principles, let us consider a few scenarios where changing a process model through transformation operations rather than arbitrary modifications would be useful.

Given a set of tasks and coordination constraints, it is possible to build two equivalent process models that may look quite different graphically on the surface but express identical functionality. Let us assume that a process designer correctly captures the process requirements of a system in the form of a process model. One of his colleagues, while looking at the model, realizes that certain changes in the process model would simplify and clarify the design and make it simpler to comprehend. However, he would not want to introduce any changes in the process model that might violate any existing coordination constraints. In such a case, transformation operations that keep the modified model semantically equivalent with the original model can be applied. A transformed equivalent model may also allow improved ability to easily introduce and reason about other changes in the process model.

It is important to point out here transformation changes in the process model result in addition or deletion of modelling objects from the process models. We are not considering changes in graph layouts here. There are several graph-drawing algo-

rithms available in the literature that can improve the visual layout of graphs. The graph layout algorithms just improve the visual representation of a process graph without making any semantic changes to the underlying modelling structures.

Another useful application of process transformations is in gradual development of a process model. Let us suppose that a process designer captures the necessary and required process tasks and associated coordination constraints in the form of a higher-level process model. This process model then may be used as a basis to define other specialized process models that would add additional tasks and coordination constraints without violating the constraints specified in the original model. In such a case, we would get several versions of the original process model with additional process modelling objects and structures.

Process transformations may also be used to effectively manage dynamic modification of process models at run-time. In workflow environments supporting dynamic changes, a process model template is used as a basis for creating new instances of business processes. Each of such process model instances could be modified at run-time to handle specialized requirements of the instance. A framework to introducing such changes through process transformations allows process designers to correctly understand the impact of their change on overall execution of the process model.

Rest of the paper is organized as follows. In section 2, we present a simple and generic process modelling language to build process models. This modelling language will be used to present three structural relationships and associated transformation operations in section 3 and 4. An algebraic notation for transformation principles will be introduced in section 5 to provide a means for formal considerations. Section 6 concludes the paper and introduces a process-modelling tool that has been developed to implement some of the ideas presented in this paper.

2 Process Modelling Structures

A process model contains tasks and associated coordination constraints to control the scheduling and execution of defined tasks. The tasks in a process model are performed to achieve some business objectives. Generally, these tasks are inter-related in such a way that the initiation of one task is dependent on the successful completion of a set of other tasks. Therefore, the order in which tasks are executed is very important. The structural / control flow modelling of a process defines the way a workflow management system would order and schedule workflow tasks.

In this section, we briefly describe the structural aspects of a typical workflow modelling language and its graphical representation. This language conforms to the generic workflow modelling concepts as described by Workflow Management Coalition [11]. The process models in this graphical language are modelled using two types of objects: node and transition. Node is classified into two subclasses: task and choice/merge coordinator. A *task*, graphically represented by a rectangle, represents the work to be done to achieve some objectives. It is also used to implicitly build sequence, fork, and synchronizer structures. It is the primary object in workflow specifications and could represent both automated and manual activities. A *choice/merge coordinator*, graphically represented by a circle, is used to build choice and merge structures. A transition links two nodes in the graph and is graphically represented by a directed edge. It shows the execution order and flow between its

head and tail nodes. By connecting nodes with transitions through modelling structures, as shown in Figure 1, we build directed acyclic graphs (DAG) called workflow graphs where vertices represent nodes and directed edges represent transitions.

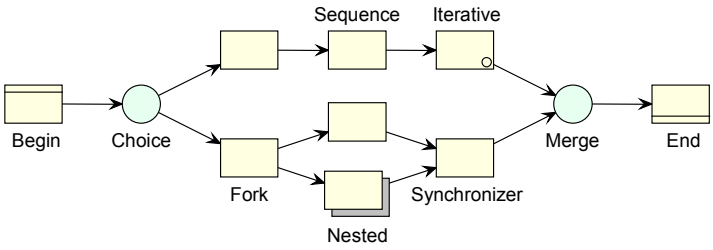

Fig. 1. Process modelling structures

Sequence is the most basic modelling structure and defines the ordering of task execution. It is constructed by connecting at most one incoming and one outgoing transition to a task. A *fork* structure is used to represent concurrent paths within a workflow graph and is modelled by connecting two or more outgoing transitions to a task. At certain points in workflows, it is essential to wait for the completion of more than one execution path to proceed further. A *synchronizer* structure, represented by attaching more than one incoming transition to a task, is applied to synchronize such concurrent paths. A task waits until all the incoming transitions have been triggered.

A *choice* structure is used to model mutually exclusive alternative paths and is constructed by attaching two or more transitions to a choice/merge coordinator object. At run-time, the workflow selects one of the alternative execution paths for a given instance of the business process by activating one of the transitions originating from the choice coordinator object. The choice structure is exclusive and complete. The exclusive characteristic ensures that only one of the alternative paths is selected. The completeness characteristic guarantees that, if a choice coordinator object is activated, one of its outgoing flows will always be triggered. A *merge* structure is "opposite" to the choice structure. It is applied to join mutually exclusive alternative paths into one path by attaching two or more incoming transitions to a merge coordinator object.

The fork and synchronizer structures are represented implicitly by directly connecting transitions to task objects. This approach keeps the resulting workflow model compact as well as graphically explicit. Nevertheless, in certain cases, it requires the use of *null* or dummy tasks to model proper coordination of flow and to conform to the syntactical correctness criteria of workflow structures.

Since a workflow model is represented by a directed acyclic graph (DAG), it has at least one node that has no incoming transitions (source) and at least one node that has no outgoing transitions (sink). We call these *begin* and *end* nodes respectively node. A workflow instance completes its execution after its end node has completed its execution. However, if a workflow graph contains more than one end node, then its instance completes its execution after executing a subset of these end nodes depending on whether choice or fork structures have been used in preceding paths of the end nodes. A bar at the top of a task or choice/merge coordinator represents a begin node. Similarly, a bar at the bottom represents an end node.

The *nesting* structure simplifies the workflow specifications through abstraction. Using this construct, we can encapsulate a workflow specification into a task and then

use that nested task in other workflow specifications. For each execution of a nested task, the underlying workflow is executed. A nested task is graphically represented through a shaded rectangle under the task rectangle. The *iteration* structure is needed to model the repetition of a group of tasks within a workflow. One way to support iteration is through exit conditions. As long as a certain condition is not met, a particular task is repeatedly executed. The nesting structure could be used if there is a need to repeat a sub graph of the workflow model. This technique for iteration, however, can only support blocked iteration.

We have come across a variety of process modelling languages both in research papers and commercial products [12]. Most of the languages support the generic modelling structures introduced here. We have defined this simple language to provide a basis for introducing process transformation relationships and rules. However, the transformation principles are applicable to other forms of process modelling languages after appropriately modifying the transformation rules.

3 Structural Transformations

In this section, we will introduce concepts of three types of structural relationships – equivalent, imply, and subsume – that two process graphs *G* and *G'* may satisfy. These relationships are progressively relaxed and subsume the restrictive ones. This means if *G* and *G'* satisfy equivalent relationship, they also satisfy imply and subsume relationships. Similarly, if *G* and *G'* satisfy imply relationship then they also satisfy subsume relationship.

While defining relationships we will use the notion of *execution nodes*. The nodes in a process graph can be classified into two groups: execution nodes and coordination nodes. In the process modelling language described in the previous section, all tasks except null tasks represent execution nodes, i.e., the nodes that directly perform some business activity. Choice/merge nodes and null tasks represent coordination nodes that are used to explicitly capture the coordination constraints graphically.

3.1 Equivalent Relationship

A workflow graph *G'* is structurally equivalent to a graph *G* if sets of execution nodes in both *G* and *G'* are equal and each one of them preserves the structural / control flow constraints specified in the other. The structurally equivalent relationship is represented by notation $G' \leftrightarrow G$. Informally, if *G* and *G'* are structurally equivalent workflow graphs then they may be used interchangeably to accomplish identical functionality.

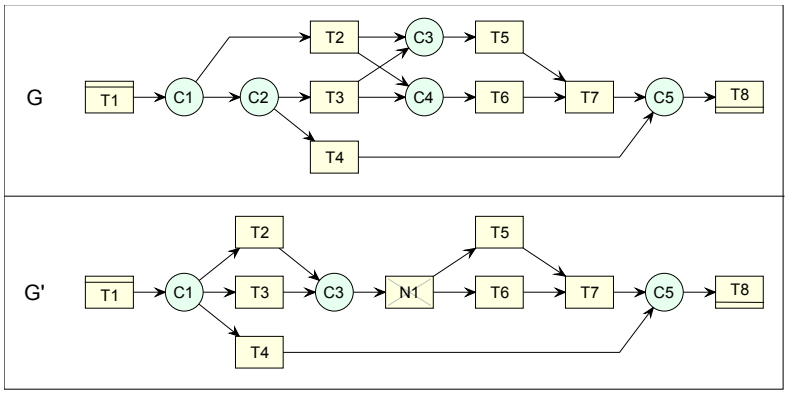

Fig. 2. Structurally equivalent relationship

In Figure 2, we show an example of two structurally equivalent workflow graphs *G* and *G'*. In *G*, after completing *T1*, a choice is made between *T2*, *T3*, and *T4* through two choice structures *C1* and *C2*. In *G'*, identical functionality is achieved through a single choice construct *C1*. In *G*, completion of either *T2* or *T3* would trigger *T5* and *T6* in parallel. The graph *G'* achieves identical functionality through different modelling structures.

3.2 Imply Relationship

A workflow graph *G'* implies *G* if sets of execution nodes in both *G* and *G'* are equal and *G'* preserves the structural / control flow constraints specified in *G*. However, G may not satisfy all of the structural / control flow constraints specified in *G*`.

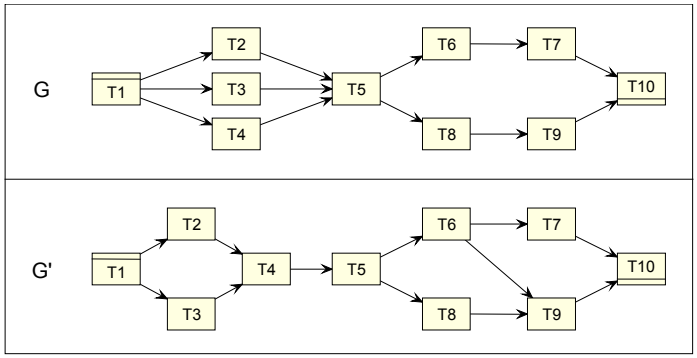

Fig. 3. Imply relationship

The imply relationship is represented by notation $G' \rightarrow G$. Informally, if a workflow graph *G'* implies *G*, then *G'* may be used to accomplish the functionality of *G* with less parallelism. However, the reverse does not hold. Additionally, if *G* and *G'* conform to equivalent relationship, then also conform to imply relationship. In graph *G* of Figure 3, *T2*, *T3*, and *T4* may be performed in any order between *T1* and *T5*.

However, *T4* in *G'* can only be performed after completing *T2* and *T3*. That means, *G'* is restrictive than *G* in specifying the order in which some of the tasks may be executed in parallel. Similarly, *T9* in *G'* will be initiated only after both *T6* and *T8* have finished. In graph *G*, however, *T9* does not have to wait for the completion of *T6*. This example shows that *G'* will perform all tasks of *G* without violating any control flow constraint specified in *G*. However, *G'* contains some additional control flow constraints that would result into less parallelism between workflow tasks.

3.3 Subsume Relationship

A workflow *G'* subsumes *G* if the set of execution nodes in *G* is a subset of the execution nodes in *G'* and *G'* preserves the structural / control flow constraints specified in *G*. However, *G* may not satisfy all of the structural / control flow constraints specified in *G`*. The subsume relationship is represented by notation $G' \supset G$. Again, informally, if a workflow graph *G'* subsumes *G*, then *G'* may be used to accomplish the functionality of *G* with less parallelism and additional tasks. However, the reverse does not hold. Additionally, if *G* and *G'* conform to either equivalent or imply relationship, then they also conform to subsume relationship.

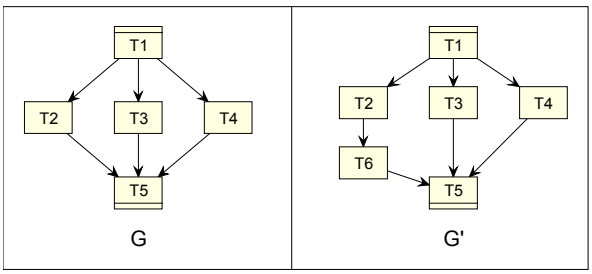

Fig. 4. Subsume relationship

Figure 4 shows an example of subsume relationship. In graph *G*, after *T2*, *T3* and *T4* are completed, *T5* may be started. However, in *G'*, *T6* has to be completed after *T2* as well before *T5* can be performed. This means, *G'* is performing all the tasks in *G* without violating any control flow constraints and in addition is performing an additional task *T6*.

4 Transformation Operations

In this section, we will identify a set of transformation operations that can be applied to transform a workflow graph *G* into a workflow graph *G'* such that both *G* and *G'* conform to the three types of relationships introduced in previous section. Accordingly, three sets of transformation rules will be presented that maintain these relationships. The transformation rules are defined on the basis of following assumptions:

– Each execution node in the workflow graph is unique and it is not possible to identify if two distinct nodes perform exactly the same function.

– The workflow graph is free of errors and conforms to the correctness criteria. It does not contain deadlock and lack of synchronization structural conflicts [9].
– The transformation rules take into account only the semantics of adjacent modelling structures.

4.1 Structurally Equivalent Transformation

We will define a set of transformation rules that may be applied in any order to transform a graph *G* into *G'* such that both *G* and *G'* are structurally equivalent. It may seem that some of the transformation rules add or remove redundant modelling structures from workflow graphs. The addition or removal of such structures in workflow graphs is also used in applying the transformation rules for imply and subsume relationships.

SE1: Choice/merge child node of a single parent choice/merge node
A child choice/merge coordinator node of a single parent choice/merge node may be removed from the graph after moving outgoing transitions of the child choice/merge node to the parent choice/merge node. Inversely, a choice/merge node may be added as a child of another choice/merge node and a subset of the outgoing transitions of the parent choice/merge node may be moved as outgoing transitions of the newly added choice/merge node.

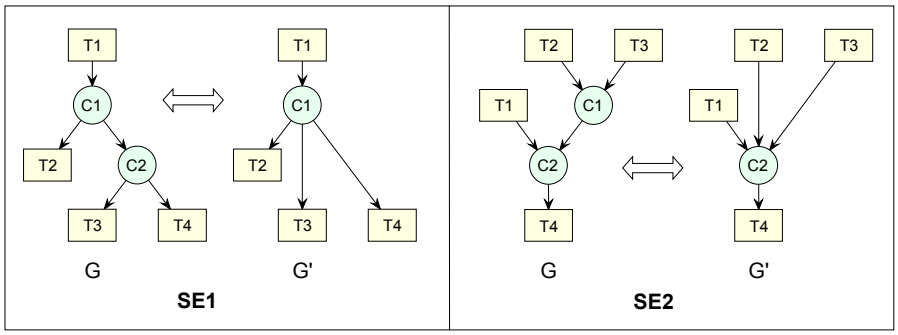

Fig. 5. Rules SE1 and SE2

In Figure 5 (SE1), *C2* has a single parent choice/merge node *C1*. By applying this transformation rule, we can move the two outgoing transitions from *C2* to *C1* and remove *C2* from the workflow graph. It is assumed that when this transformation is applied, the underlying transition conditions for the choice structure are also appropriately modified.

SE2: Choice/merge parent node of single child choice/merge node
A parent choice/merge node of a single child choice/merge node may be removed from the graph after moving incoming transitions of the parent choice/merge node to the child choice/merge node. Inversely, a choice/merge node may be added as a parent of another choice/merge node and a subset of the incoming transitions of the child

choice/merge node may be moved as incoming transitions of the newly added choice/merge node.

In Figure 5 (SE2), *C1* has a single child choice/merge node *C2*. By applying this transformation rule, we can move the two incoming transitions from *C1* to *C2* and remove *C1* from the graph.

SE3: Child null task node of a single parent task node

A child null task node of a single parent task node may be removed from the graph after moving outgoing transitions of the null task to the parent task. Inversely, a null task node may be added as a child of another task and a subset of the outgoing transitions of the parent task may be moved as outgoing transitions of the newly added null task.

In Figure 6 (SE3), *N1* is a null task that has a single parent task node *T1*. By applying this transformation rule, we can move the two outgoing transitions from *N1* to *T1* and remove *N1* from the graph. The use of a null task in following example may seem redundant. However, it could be used to clarify the design of a workflow model. This transformation rule is also quite useful when applied in conjunction with imply and subsume transformation operations that will be explained later.

SE4: Parent null task node of a single child task node

A parent null task node of a single child task node may be removed from the graph after moving incoming transitions of the null task to the child task. Inversely, a null task node may be added as a parent of another task and a subset of the incoming transitions of the child task may be moved as incoming transitions of the newly added null task.

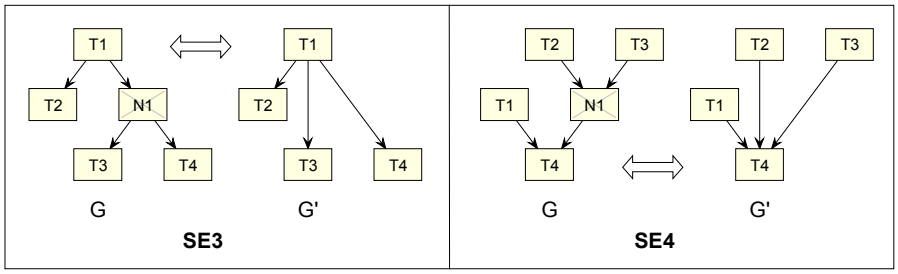

Fig. 6. Rules SE3 and SE4

In Figure 6 (SE4), *N1* is a null task that has a single child task node *T4*. By applying this transformation rule, we can move the two incoming transitions from *N1* to *T4* and remove *N1* from the graph.

SE5: Overlapping synchronizer and fork structures

Overlapping implicit synchronizer and fork structures are formed whenever two or more task nodes have identical sets of two or more parent task nodes. Such structures may be merged into explicit structures using a null task. Inversely, the null task may be removed to transform explicit synchronizer and fork structures into structurally equivalent implicit representation.

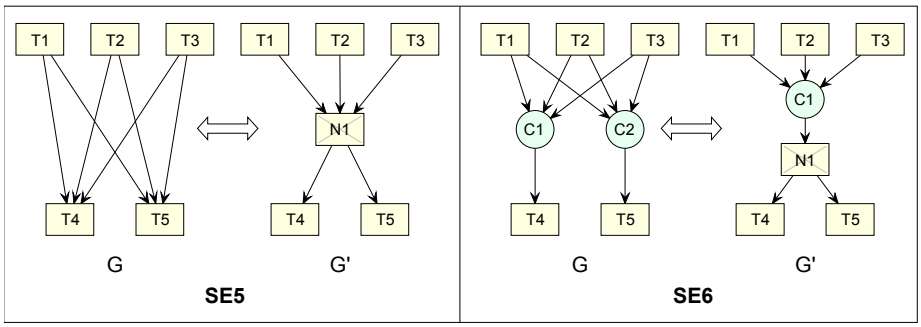

Fig. 7. Rules SE5 and SE6

In graph *G* of Figure 7 (SE5), *T4* and *T5* can be started in parallel as soon *T1*, *T2*, and *T3* have been completed. This synchronization and fork behavior has been modelled implicitly in *G*. By applying SE5 transformation operation, we can introduce a null task *N1* in *G'* that explicitly model the synchronization *T1*, *T2*, and *T3* and parallel fork of *T4* and *T5*.

SE6: Overlapping merge and fork structures

Overlapping implicit merge and fork structures are formed whenever two or more choice/merge nodes have identical sets of two or more parent task nodes. Such structures may be merged into explicit structures using a single explicit merge structure and a single explicit fork construct using a null task. Inversely, the null task and merge construct may be removed to transform explicit constructs into structurally equivalent implicit representation.

In graph *G* of Figure 7 (SE6), *C1* and *C2* are two merge nodes that have identical sets of parent task nodes *T1*, *T2*, and *T3*. By applying this transformation rule, we can remove *C2*, insert a null task *N1* and move outgoing transitions of *C1* and *C2* to *N1*.

SE7: Redundant synchronizing transitions

A redundant synchronizing transition may be added between two workflow tasks only if it does not introduce a deadlock structural conflict. Inversely, a redundant synchronizing transition may be removed from the workflow graph.

In graph *G* of Figure 8, a transition is added between *T1* and *T5*. This new transition adds a constraint that *T5* cannot start until *T1*, *T3*, and *T4* have finished. However, completion of *T3* and *T4* transitively ensures that *T1* has also been completed. Therefore, a transition between *T1* and *T5* is redundant since it does not affect the execution order of the workflow tasks.

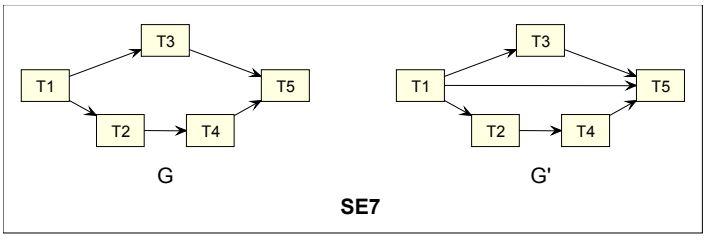

Fig. 8. Rule SE7

4.2 Imply Transformation

The transformation rules for structurally equivalent relationship can be applied since they maintain the imply relationship. In addition, the following transformation rule may also be applied.

IM1: Adding non-redundant synchronizing transitions
A non-redundant transition may be added between two workflow tasks only if it does not introduce a deadlock structural conflict. The addition of a non-redundant transition between two workflow tasks introduces some new restrictions on the possible parallel execution of one or more tasks in the workflow definition. It may also make some other transitions in the model redundant that may be removed by applying SE7.

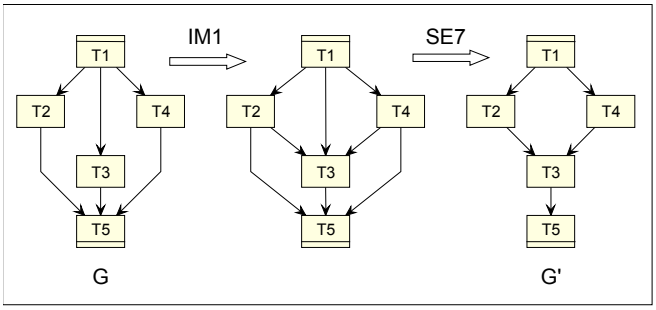

Fig. 9. Rule IM1

In graph *G* of Figure 9, IM1 is applied twice to add two new transitions, from *T2* to *T3* and from *T4* to *T3*. Then SE7 is applied to remove three transitions from *T1* to *T3*, *T2* to *T5* and *T4* to *T5* that have become redundant after applying IM1.

4.3 Subsume Transformation

The transformation rules for structurally equivalent relationship and imply relationship can be applied since they maintain the subsume relationship. In addition, the following transformation rule may also be applied.

SU1: Insert a subgraph in place of a transition
A transition in a workflow graph may be replaced by a workflow graph by adding the subgraph between the head and tail node of the transition. However, the added subgraph must conform to certain properties to ensure that it does not introduce errors in workflow graph and for all possible executions of the subgraph the tail node of the replaced transition is always triggered.

In graph *G* of Figure 10, SU1 is applied to replace the transition from *T3* to *T4* with a subgraph starting from *C1* and ending at *C2*. It is also applied to insert a task *T7* between *T2* and *T4*.

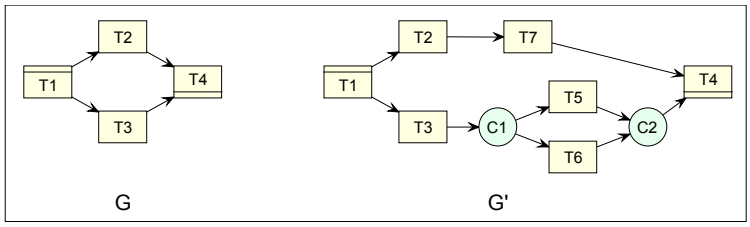

Fig. 10. Rule SU1

5 Formal Considerations

In this section, we will introduce an algebraic notation to represent process model structures. Using the notation, a process graph may be represented through a set of triggering constraints.

A triggering constraint (TC) is of the form $X \rightarrow Y$, implying that the completion of an instance of X triggers the start of an instance of Y. Here X and Y represent algebraic terms building upon workflow nodes and transitions.

Given a process graph, we can represent it through a set of simple triggering constraints of the form:

Simple triggering constraints:

Triggering Constraint: Node $\rightarrow$ Term | Term $\rightarrow$ Node
Term: Node | (Node $\vee$ Node …) | (Node $\wedge$ Node …)

We can also transform a set of simple triggering constraints to a set of composite triggering constraints by removing non-executing nodes and merging associated triggering constraints. This transformation is performed using a set of triggering constraint transformation principles that will be presented later in this section.

Composite triggering constraints:

Triggering Constraint: Term $\rightarrow$ Term
Term: Node | (Term $\vee$ Term …) | (Term $\wedge$ Term …)

Figure 11 shows a mapping of process modelling structures to simple triggering constraints. The notation used for triggering constraints is simple and self-explanatory.

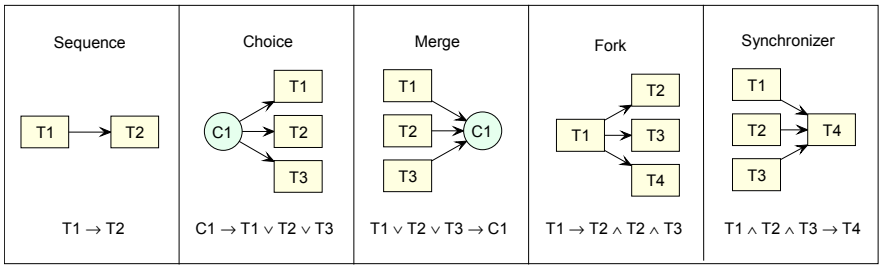

Fig. 11. Mapping modelling structures to triggering constraints

The underlying principles for transforming simple triggering constraints (STCs) to composite triggering constraints (CTCs) are shown in Table 1. In these principles, Rx represent triggering constraint terms, Cx represent choice / merge nodes, and Nx represent null task nodes. By applying these principles, we can reason about the transformation operations presented in section 4.

Table 1. Transformation principles for triggering constraints

$R1 \vee R2 \Leftrightarrow R2 \vee R1$ $R1 \wedge R2 \Leftrightarrow R2 \wedge R1$	(P1) (P2)
$R1 \vee (R2 \vee R3) \Leftrightarrow R1 \vee R2 \vee R3 \Leftrightarrow (R1 \vee R2) \vee R3$ $R1 \wedge (R2 \wedge R3) \Leftrightarrow R1 \wedge R2 \wedge R3 \Leftrightarrow (R1 \wedge R2) \wedge R3$	(P3) (P4)
$R1 \rightarrow R3,\ R2 \rightarrow R3 \Leftrightarrow R1 \vee R2 \rightarrow R3$ $R1 \rightarrow R2,\ R1 \rightarrow R3 \Leftrightarrow R1 \rightarrow R2 \wedge R3$	(P5) (P6)
$R1 \rightarrow C1,\ C1 \rightarrow R2 \Leftrightarrow R1 \rightarrow R2$ $R1 \rightarrow N1,\ N1 \rightarrow R2 \Leftrightarrow R1 \rightarrow R2$	(P7) (P8)
$R1 \rightarrow C1,\ C1 \vee R2 \rightarrow R3 \Leftrightarrow R1 \vee R2 \rightarrow R3$ $R1 \rightarrow N1,\ N1 \wedge R2 \rightarrow R3 \Leftrightarrow R1 \wedge R2 \rightarrow R3$	(P9) (P10)
$R1 \rightarrow C1 \vee R2,\ C1 \rightarrow R3 \Leftrightarrow R1 \rightarrow R2 \vee R3$ $R1 \rightarrow N1 \wedge R2,\ N1 \rightarrow R3 \Leftrightarrow R1 \rightarrow R2 \wedge R3$	(P11) (P12)

6 Concluding Remarks

The main contribution of the paper is to identify three useful classes of structural relationships for transforming process models and to introduce a framework for systematically introducing changes in process models. We have presented three structural relationships – equivalent, imply, and subsume – and associated transformation operations that may be used to systematically changing process models. We have also introduced a simple algebraic notation to represent and reason about process model transformations.

The structural relationships and transformation principles have been introduced using a generic process modelling language conforming to the process definition con-

cepts defined by Workflow Management Coalition. The language makes use of core modelling structures – sequence, fork, synchronizer, choice, merge, iteration, and nesting – to build structural / control flow specifications of process models.

The concepts introduced in this paper have been implemented in a workflow modelling and verification tool called FlowMake. The tool allows both constrained and unconstrained approaches for introducing changes in the process models. It provides workflow analysts and designers a well-defined framework to model and reason about various aspects of workflows. It has been designed to augment workflow products with enhanced modelling capabilities. More information about FlowMake is available at http://www.dstc.edu.au/Research/Projects/FlowMake/.

References

1. Sadiq W and Orlowska ME (1997). On Correctness Issues in Conceptual Modeling of Workflows. In Proceedings of the 5th European Conference on Information Systems (ECIS '97), Cork, Ireland, June 19-21, 1997.
2. Carlsen S (1997). Conceptual Modeling and Composition of Flexible Workflow Models. PhD Thesis. Department of Computer Science and Information Science, Norwegian University of Science and Technology, Norway, 1997.
3. Casati F, Ceri S, Pernici B and Pozzi G (1995). Conceptual Modeling of Workflows. In M.P. Papazoglou, editor, Proceedings of the 14th In-ternational Object-Oriented and Entity-Relationship Modeling Conference, volume 1021 of Lecture Notes in Computer Science, pages 341-354. Springer-Verlag.
4. Georgakopoulos D, Hornick M and Sheth A (1995) An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Journal on Distributed and Parallel Databases, 3(2):119-153.
5. Hofstede, AHM ter, Orlowska ME and Rajapakse J (1998). Verification Problems in Conceptual Workflow Specifications. Data & Knowledge Engineering, 24(3):239-256, January 1998.
6. Rajapakse J (1996). On Conceptual Workflow Specification and Verification. MSc Thesis. Department of Computer Science, The University of Queensland, Australia, 1996.
7. Reichert M and Dadam P (1997). ADEPTflex - Supporting Dynamic Changes of Workflow without loosing control. Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management.
8. Sadiq W and Orlowska ME (1999). On Capturing Process Requirements of Workflow Based Information Systems. In Proceedings of the 3rd International Conference on Business Information Systems (BIS '99), Poznan, Poland, April 14-16, 1999.
9. Sadiq W and Orlowska ME (1999). Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99), Heidelberg, Germany, 14-18 June 1999. Lecture Notes in Computer Science 1626. pp. 195-209. Springer-Verlag.
10. Workflow Management Coalition (1996) The Workflow Management Coalition Specifications - Terminology and Glossary. Issue 2.0, Document Number WFMC-TC-1011.
11. Workflow Management Coalition (1998). Interface 1: Process Definition Interchange, Process Model, Document Number WfMC TC-1016-P.
12. Butler Report (1996). Workflow: Integrating the Enterprise. The Butler Group, 1996.
13. Sadiq S (2000). Handling Dynamic Schema Change in Process Models. In Proceedings of Australian Database Conference, Canberra, Australia. January, 2000.