Activity Relations: A Dataflow Approach to Workflow Design

Sherry Sun
*University of Arizona*

Leon Zhao
*University of Arizona*

Leon Zhao
*University of Arizona*

Follow this and additional works at: [http://aisel.aisnet.org/icis2006/44](http://aisel.aisnet.org/icis2006/44)

**Recommended Citation**
[http://aisel.aisnet.org/icis2006/44](http://aisel.aisnet.org/icis2006/44)

This material is brought to you by the International Conference on Information Systems (ICIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ICIS 2006 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
ACTIVITY RELATIONS: A DATAFLOW APPROACH TO WORKFLOW DESIGN

Sherry X. Sun
Department of MIS,
Eller College of Management
University of Arizona
xiaoyun@email.arizona.edu

J. Leon Zhao
Department of MIS,
Eller College of Management
University of Arizona
jlzhao@email.arizona.edu

Abstract

A key step in workflow design is to determine the activity sequences, which are often driven by the dataflow constraints in a business process. Therefore, the literature has suggested that workflow design can start with dataflow analysis. However, no formalism exists for deriving activity sequences from a set of identified activities and their input and output data. In this paper, we formalize the problem of workflow design on the basis of dataflow analysis. We tackle the problem by using the concept of “activity relations” as an intermediate step for identifying the possible activity execution sequences from dataflow. We investigate how to derive activity relations from dataflow and discuss their implication in workflow design.

Keywords: Activity relations, workflow design, dataflow analysis, data dependencies

Introduction

Designing a workflow model is a complicated task where many factors, such as resources sharing and business policies, have to be taken into consideration (Stohr and Zhao 2001). Among these factors, one dominant factor is dataflow, namely what data are needed as input by activities and what data are produced as output by activities. Dataflow can drive the constraints that control activity sequencing (Kwan and Balasubramanian, 1997). If the constraints derived from dataflow are violated, dataflow errors will occur, leading to unexpected workflow termination and high debugging cost at run time (Sun et. al. 2004).
The information about dataflow can be collected from existing forms and documents, such as product specification, without knowing the sequences of activity execution (Reijers et al. 2003). Therefore, the dataflow in a business process can be determined before the model of activity sequences is identified. The information contained in a dataflow model can help determine activity sequences in the workflow to be developed. Figure 1 shows a workflow design framework based on dataflow analysis. This framework starts with identification of a set of business activities and their input and output data. After a dataflow model is created, activity relations can be derived from the dataflow model and then used to identify the sequences of activity execution. The result of workflow design is a set of routing activities including ANDSplit, ANDJoin, XORSplit, and XORJoin, and the execution sequences among business activities and routing activities.

This paper focuses on deriving activity relations from a dataflow model. We first extend the activity-based workflow modeling to incorporate the dataflow aspect. Further, we formally define the concept of activity relations and provide design principles to derive activity relations from a dataflow model.

Literature Review

While a significant amount of research in the workflow area has focused mainly on modeling, verification, and architecture issues (Ellis and Maltzahn 1997; Aalst and Hee 2002; Bi and Zhao 2004), a formal approach to workflow design was outside the scope of most workflow research until very recently (Stohr and Zhao 2001). The method of Product-Based Workflow Design (Reijers et al. 2003) uses the relationship among data elements derived from product specifications as a starting point for workflow design. Moreover, cost and flow time are considered as criteria for selection of workflow models, and breadth-first and depth-first search are used at the step of determining activity sequences. However, the principles on using parallel routing and conditional routing are not emphasized.

Research in workflow design can benefit from the stream of work in business process redesign. Business process redesign deals with both technical issues and socio-cultural issues related to restructuring a business process for improvement in cost, quality, speed, and service. For instance, the analytical model proposed by Aalst (2001) focuses on minimizing time and maximizing resource utilization through sequential and parallel routing of tasks in a particular type of processes where each activity can produce only two possible results. As suggested by Reijers et al. (2003), the multiple optimization criteria found in business process redesign can be used as criteria for workflow model evaluation in the workflow design process.

The basic idea of applying dataflow analysis to workflow design was proposed in Sun and Zhao (2004). In this paper, we extend our previous research with a new concept called “activity relations”, which is the foundation for building a formal workflow design procedure. In addition, we provide the criteria for deciding whether a dataflow model provides sufficient information for workflow design. The activity relations we propose are different from “log-based ordering relations” used in (Aalst et al. 2004) in that activity relations describe the structures of a workflow model and log-based ordering relations describe the event sequences recorded in an event log.

Dataflow Analysis

In this section, we present the basic dataflow concepts (Sun and Zhao 2004). Further, several dataflow analysis instruments are devised, including direct requisite set and full requisite set, to serve the purpose of workflow design.

Preliminaries – Data Dependencies and Activity Dependencies

**Definition 1** (Data Dependency) Activity $v_i$ depends on a set of input data $I_{v_i}$ to produce a set of output data $O_{v_i}$, which is referred to as the data dependency for $v_i$ and is denoted as $\lambda_\rightarrow(I_{v_i},O_{v_i})$.

Note that in this paper, we only consider the data that each activity indispensably depends on. The optional data dependencies, i.e., an activity needs some data just for reference and can go forward even without it, are not considered.

**Definition 2** (Conditional Routing Constraint) A conditional routing constraint $c$ specifies that when a condition clause $f(D)$ is evaluated to be true, a set of activities $V$ will be executed, denoted as $c=f(D) \rightarrow \text{Execute}(V)$, where $D$ is a set of data items and $f(D)$ is a logic expression on $D$. 
An example of conditional routing constraint is that when the condition “travel expense is greater than $5,000” is evaluated to be true, the activity “Approve the travel application by a director” will be executed.

There are three types of data dependencies: mandatory, conditional, and execution dependencies.

**Definition 3 (Mandatory, Conditional, Execution Data Dependency)** Data dependencies for activity $v_j$ can be categorized into three types, mandatory, conditional, and execution data dependencies, denoted as $\lambda_{vi}^{\text{m}}[I_{vi}^{m},O_{vi}^{m}]$, $\lambda_{vi}^{\text{c}}[I_{vi}^{c},O_{vi}^{c}]$, and $\lambda_{vi}^{\text{e}}[I_{vi}^{e},O_{vi}^{e}]$. Note the set of input data for $v_j$ is decomposed into three subsets, i.e., $I_{vi}=I_{vi}^{m}\cup I_{vi}^{c}\cup I_{vi}^{e}$, and

- $I_{vi}^{m}$ is the set of input data that $v_j$ must use, i.e., $\forall d\in I_{vi}^{m}$, if $d$ is null at run time then $v_j$ will not be activated,
- $I_{vi}^{c}$ represents the set of input data that $v_j$ conditionally depends on, i.e., $\forall d\in I_{vi}^{c}$, under some conditions, $v_j$ may be executed without using $d$,
- $I_{vi}^{e}$ represents the set of data that $\forall d\in I_{vi}^{e}$, there exists a conditional routing constraint $c=f(D):\text{Execute}(V)$ such that $d\in D$ and $v_j\in V$, and the data dependency on $d$ occurs only when activity $v_j$ is executed.

Note that for simplicity, we use $\lambda_{vi}^{\text{m}}[I_{vi}^{m}|I_{vi}^{c}|I_{vi}^{e},O_{vi}^{m}]$ as an abbreviation for the totality of $\lambda_{vi}^{\text{m}}[I_{vi}^{m},O_{vi}^{m}]$, $\lambda_{vi}^{\text{c}}[I_{vi}^{c},O_{vi}^{c}]$, and $\lambda_{vi}^{\text{e}}[I_{vi}^{e},O_{vi}^{e}]$.

**Example 1 (Order processing workflow)** Assume that we are to design an order processing workflow. Figure 2 shows the relevant activities and data items. In this workflow, activity $v_2$, update product availability, always uses the product quantities ordered by a customer as input in order to update the product availabilities. Therefore, data item $d_3$, quantities ordered, is the mandatory data input of $v_2$ and data item $d_4$, updated availability, is the output of $v_2$, i.e., $d_3\in I_{v_2}$ and $d_4\in O_{v_2}$.

Further, if a customer orders more than what is available, the order process should not proceed. Instead, a replenish order should be sent to the manufacturer and a back order notice should be sent to the customer. Therefore, the routing constraints are $c_7=(d_5=d_2: \text{Execute}(v_2, v_3))$ and $c_9=(d_6=d_3: \text{Execute}(v_2, v_4, v_5))$. As such, $d_1, d_2\in I^{e}$ for $v_2, v_3, v_4, v_5$, and $v_6$.

Moreover, $d_7$, order confirmation No., is one of the final outputs from this process, i.e., $d_7$ is the input and output of the end activity. However, when quantities ordered ($d_5$) are more than the quantities available ($d_3$), the order cannot be confirmed. As such, $d_7$ can be null. The end activity will still be activated even if $d_7$ is null, i.e., $d_7\in I^{e}$ for the end activity. Table 1 shows the data dependencies of each activity in the order processing workflow.

---

**Table 1** Data dependencies in the order processing workflow

<table>
<thead>
<tr>
<th>Activities</th>
<th>Data Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$: process order</td>
<td>$d_2$: quantities available</td>
</tr>
<tr>
<td>$v_2$: update product availability</td>
<td>$d_1$: product IDs</td>
</tr>
<tr>
<td>$v_3$: send replenishment order</td>
<td>$d_3$: customer ID</td>
</tr>
<tr>
<td>$v_4$: confirm order</td>
<td>$d_4$: quantities ordered</td>
</tr>
<tr>
<td>$v_5$: send back order notice</td>
<td>$d_5$: updated availability</td>
</tr>
<tr>
<td>$v_6$: make shipment</td>
<td>$d_6$: replenishment quantities</td>
</tr>
<tr>
<td>$e$: end activity</td>
<td>$d_7$: confirmation No.</td>
</tr>
<tr>
<td>$s$: start activity</td>
<td>$d_8$: shipping date</td>
</tr>
<tr>
<td></td>
<td>$d_9$: back order notice status</td>
</tr>
</tbody>
</table>

---

1 Without loss of generality, we assume that the set of external data $E$ does not overlap with any input data set $I_v$, that is, $\forall v\in V$, $E\cap I_v=\emptyset$. So, if $d\in I_v$ then $d\notin E$ hereafter. Further, we assume that the external data $E$ is necessary and sufficient.
activity dependency between two activities \( v_i \) and \( v_j \), we denote the non-dependency between the two activities as \( v_i \not\rightarrow v_j \). Further, if \( d \in I_{vi} \) and \( d \in O_{vj} \), \( v_i \) has a mandatory dependency on \( v_j \), denoted as \( v_j \Rightarrow m_{vi} \). If \( d \in I_{vi} \) and \( d \in O_{vj} \), \( v_i \) has a conditional dependency on \( v_j \), denoted as \( v_j \Rightarrow c_{vi} \). If \( d \in I_{vi} \) and \( d \in O_{vj} \), \( v_i \) has an execution dependency on \( v_j \), denoted as \( v_j \Rightarrow e_{vi} \).

<table>
<thead>
<tr>
<th>Activities</th>
<th>( \Lambda_v \cdot [I_v, I_{vi}, O_{vi}] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>s: Start activity</td>
<td>( \Lambda_0 [ {d_0 } \cup {d_0 }] )</td>
</tr>
<tr>
<td>e: End activity</td>
<td>( \Lambda_v [ {d_i, d_j, d_s, d_d, d_b } \cup {d_i, d_j, d_s, d_d, d_b }] )</td>
</tr>
<tr>
<td>( v_1 ): process order</td>
<td>( \Lambda_{v1} [ {d_i, d_j, d_s } \cup {d_d }] )</td>
</tr>
<tr>
<td>( v_2 ): update product availability</td>
<td>( \Lambda_{v2} [ {d_i } \cup {d_i, d_j } \cup {d_j }] )</td>
</tr>
<tr>
<td>( v_3 ): send replenishment order</td>
<td>( \Lambda_{v3} [ {d_i } \cup {d_i, d_j } \cup {d_d } \cup {d_s }] )</td>
</tr>
<tr>
<td>( v_4 ): confirm order</td>
<td>( \Lambda_{v4} [ {d_i, d_j } \cup {d_i, d_j } \cup {d_j } \cup {d_d }] )</td>
</tr>
<tr>
<td>( v_5 ): send back order notice</td>
<td>( \Lambda_{v5} [ {d_i } \cup {d_i, d_j } \cup {d_d }] )</td>
</tr>
<tr>
<td>( v_6 ): make shipment</td>
<td>( \Lambda_{v6} [ {d_i, d_j } \cup {d_d } \cup {d_s }] )</td>
</tr>
</tbody>
</table>

**Dataflow Analysis Concepts**

**Definition 5** (Dataflow) Given a set of business activities \( V \), dataflow \( \Lambda \) is the set of data dependencies for activities in \( V \), denoted as \( \Lambda = \{ \lambda_v \cdot [I_v, I_{vi}, O_{vi}] \mid v \in V \} \) or \( \Lambda = \{ \lambda_v \cdot [I_v, O_{vi}] \mid v \in V \} \).

In essence, Table 1 shows the dataflow for the order processing workflow.

**Definition 6** (Direct Requisite Set \( \Delta_v \)) A set of activities \( \Delta_v \) is the direct requisite set for activity \( v \) if for any activity \( x \in \Delta_v \), there exists a data item \( d \) such that \( d \in O_x \), \( d \in I_v \), where \( I_v \) is the input data set of activity \( v \), i.e., \( I_v = \bigcup_{v \in V} I_v \cup \bigcup_{v \in V} O_{vi} \). \( O_x \) is the output data set of \( x \), and \( x \neq v \).

**Definition 7** (Completeness of \( \Delta_v \)) Given activity \( v \) and \( \Delta_v \), if \( \forall d \in I_v \), there exists \( v_j \) such that \( v_j \in \Delta_v \) and \( d \in O_{vj} \), then \( \Delta_v \) is complete.

**Definition 8** (Full Requisite Set \( \Gamma_v \)) Given a set of activities \( V \), their data dependencies \( \Lambda = \{ \lambda_v \cdot [I_v, O_{vi}] \mid v \in V \} \), and activity \( v \in V \), the full requisite set \( \Gamma_v \) for \( v \) is a subset of \( V \) such that if \( u \not\Rightarrow \Gamma_v \), then \( u \not\Rightarrow v \).
Definition 9 (Independent sets): Given multiple sets of activities $V_1, V_2, ..., V_n$, if for any $v_i \in V_j$, $v_j \in V_2, ..., v_n \in V_n$, $v_i$ and $v_j$ (where $i, j = 1, 2, 3, ..., n$ and $i \neq j$) do not depend on each other, we say $V_i \cap V_j \cap \ldots \cap V_n$ are independent of each other, denoted as $V_i \cap V_j \cap \ldots \cap V_n$. Informally, we refer to $V_i, V_j, ..., V_n$ as independent sets. Note if $V_i \cap V_j \cap \ldots \cap V_n$, then $(V_i \cup V_j) \cap \ldots \cap V_n$.

In the order processing workflow, $\{v_2\}, \{v_3, v_4\}, \{v_5, v_6\}$ are independent sets. Moreover, by Definition 9, $\{v_2, v_3\} \cap \{v_4, v_5\}$ and $\{v_2, v_3\} \cap \{v_4, v_5\}$ hold. The full requisite set $\Gamma_v$ can be constructed from the direct requisite set $\Delta_v$, and the independent sets can be created given $\Gamma_v$ for every activity $v$. Due to space limit, the algorithms for deriving the full requisite set $\Gamma_v$ and the independent sets are omitted, but will be reported elsewhere (Sun and Zhao, 2006).

The Workflow Model

A workflow model includes both its dataflow and control flow. The control flow represents a set of activities and their execution sequences in a workflow (Aalst and Hee 2002; Bi and Zhao 2004). In this section, we formalize the concept of activity relations to represent the activity execution structures in the control flow. We extend activity based modeling, since it is used in most existing information systems (Lin et al. 2002).

Definition 10. (Workflow Model) A workflow model $W$ is a 7-tuple $<A, s, e, R(\text{type}), L, A, C>$, where

- $A$ is a finite set of business activities and $\forall v \in (A \setminus \{s, e\})$, $\text{InDegree}(v) = \text{OutDegree}(v) = 1$,
- $s \in A$ is the start activity and $\text{InDegree}(s) = 0$, $\text{OutDegree}(s) = 1$,
- $e \in A$ is the end activity and $\text{InDegree}(e) = 1$, $\text{OutDegree}(e) = 0$,
- $R(\text{type})$ is a finite set of routing activities where $\text{type} \in \{\text{XORsplit, XORJoin, ANDSplit, ANDJoin}\}$,
- $L_{\subseteq}(A \cup R(\text{type})) = (A \cup R(\text{type}))$ is a set of directed arcs among activities,
- $A$ is the dataflow of $A$, i.e., $A = \{\lambda_{v_i} [f_{v_i}^m | f_{v_i}^n | O_{v_i}] | v_i \in A\}$,
- $C = \{c = f(D) : \text{Execute}(V_c) \text{ and } D \subseteq \{O_v | v \in A\} \text{ and } V_c \subseteq A\}$ is a set of conditional routing constraints.

In Definition 10, $\text{InDegree}(v)$ is the number of arcs coming to $v$ and $\text{OutDegree}(v)$ is the number of arcs leaving from $v$. Figure 3 shows the graphic representation of a workflow model, where $v_1, v_2, v_3, v_4,$ and $v_5$ are business activities and $r_1, r_2, r_3, r_5$ are routing activities.

| Table 2. Activity Dependencies, $\Delta_v$ and $\Gamma_v$ in Example 1 |
|-----------------|-----------------|-----------------|
| Activity        | $\Delta_v$      | $\Gamma_v$      |
| $s$: Start activity | $\emptyset$      | $\emptyset$      |
| $e$: End activity    | $\{v_2, v_3, v_4, v_5\}$ | $\{s, v_1, v_2, v_3, v_4, v_5\}$ |
| $v_1$: process order     | $\emptyset$      | $\emptyset$      |
| $v_2$: update product availability | $\{s, v_1\}$      | $\{s, v_1\}$      |
| $v_3$: send replenishment order    | $\{s, v_1\}$      | $\{s, v_1\}$      |
| $v_4$: confirm order       | $\{s, v_1\}$      | $\{s, v_1\}$      |
| $v_5$: send back order notice | $\{s, v_1, v_3\}$ | $\{s, v_1, v_3\}$ |
| $v_6$: make shipment       | $\{s, v_1, v_4\}$ | $\{s, v_1, v_4\}$ |
We define five types of activity relations. Relations “∗”, “>”, and “>>” indicate sequential relations. vi∗vj indicates that vi is to be fired right after vj is fired, and vi>>vj and vi>>w, vj indicate that vi precedes vj but vj does not necessarily follow vi immediately. The difference between “>>” and “>>” is that “>>” indicates a strong precedence and “>>” indicate a week precedence. vi>>vj indicates that vi must be executed before vj every time vj is executed. vi>>vj indicates that vi must be executed before vj when vi and vj are both executed. For example, in Figure 3, v1 must be executed before v4 every time when v4 is executed, therefore v1>>v4. Since v3 precedes v4 only when both of them are executed, we have v3>>v4. Relation “∧” indicates parallelism described by Property 1 and “∨” indicates conditional routing described by Property 2. These two properties can be easily verified; thus, we give them without proof.

Property 1. Let W=<A, s, e, R(type), L, A, C> be an acyclic workflow model. Let vi, vj, x∈A∪R(type), vi∧vj, y∈R(ANDJoin), y>x, and σ(vi, e)∩σ(vj, e)=σ(y, e). We have:
- σ(vi, vj)=∅.
- For every σ(s, x), if y∈σ(s, x), then y-vi∈σ(s, x).

Property 2. Let W=<A, s, e, R(type), L, A, C> be an acyclic workflow model. Let vi, vj, x∈A∪R(type), vi∨vj, y∈R(XORJoin), y>x, and σ(vi, e)∩σ(vj, e)=σ(y, e). We have:
- σ(vi, vj)=∅.
- For every σ(s, x), if y∈σ(s, x), then y-vi∈σ(s, x).

We borrow the concepts of dot notation, firing rule, and firing sequence from Petri nets but redefine them in the context of activity-based modeling.
Workflow Design Principles

Workflow design issues include the verification of dataflow and the identification of activity relations. Figure 4 summarizes the symbols used in this section.

Verification of Dataflow

Given a set of activities and their data dependencies, the first step of workflow design is to examine if the set of data dependencies is complete and concise. We need to make sure that the dataflow is complete, i.e., any input data not provided by external resources is produced as output by an activity in the workflow. Further, the dataflow has to be concise, i.e., all the output data are useful and no more than one activity produces the same output data. The purpose of dataflow verification is to remove potential data errors before a control flow model is to be generated.

Definition 15 (Completeness of Dataflow): Given a set of activities $V$ and its dataflow $A=\{\lambda_v[I_v, O_v]\ | v \in V\}$, if $\forall v \in V, \Delta_v$ is complete, then $A$ is complete.

Definition 16 (Conciseness of Dataflow): Given a set of activities $V$ and its dataflow $A=\{\lambda_v[I_v, O_v]\ | v \in V\}$, $A$ is concise if the following two conditions are satisfied: 1) for each $d \in O_v$, where $v \in V$, there exists $v_j \in V$ such that $d \in I_{v_j}$; and 2) for each $d \in I_v$, where $v \in V$, there exists only one $v_j \in V$ such that $d \in O_{v_j}$.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c, c'$</td>
<td>conditional routing constraint</td>
</tr>
<tr>
<td>$s$</td>
<td>start activity</td>
</tr>
<tr>
<td>$e$</td>
<td>end activity</td>
</tr>
<tr>
<td>$x, y, v, v_i, v_j$: any activity</td>
<td></td>
</tr>
<tr>
<td>$r, r_i$: any routing activity</td>
<td></td>
</tr>
<tr>
<td>$A, V, V_i$: a set of business activities</td>
<td></td>
</tr>
<tr>
<td>$W$: a workflow model</td>
<td></td>
</tr>
<tr>
<td>$I_v$: the set of data items as input for activity $v$</td>
<td></td>
</tr>
<tr>
<td>$d, d_i$: data item</td>
<td></td>
</tr>
<tr>
<td>$D$: a set of data items</td>
<td></td>
</tr>
<tr>
<td>$E$: the overall set of external data to $W$</td>
<td></td>
</tr>
<tr>
<td>$P_{v'}$</td>
<td>the set of input data activity $v$ must use</td>
</tr>
<tr>
<td>$P_{v'}$: the set of input data activity $v$ conditionally depends on</td>
<td></td>
</tr>
<tr>
<td>$P_{v'}$: the set of input data activity $v$ has execution dependency on</td>
<td></td>
</tr>
<tr>
<td>$O_v$: the set of data items as final output from $W$</td>
<td></td>
</tr>
<tr>
<td>$O_{v'}$: the set of data items as output of activity $v$</td>
<td></td>
</tr>
</tbody>
</table>

$\lambda: $ dataflow

$C$: workflow routing constraint set

$\lambda = \lambda_m, \lambda_c, \lambda_e$: mandatory, conditional, and execution data dependencies for activity $v$ respectively, and $v$ is any activity including $s$ and $e$

$\Rightarrow, \Rightarrow_m, \Rightarrow_c, \Rightarrow_e$: activity dependency, mandatory activity dependency, conditional activity dependency, and execution dependency, respectively

$\Delta_v$: direct requisite set for activity $v$

$\Gamma_v$: full requisite set for activity $v$

$\infty$: no dependency

$v_i \ast v_j$ | $v_i$ is to be fired right after $v_j$ is fired |

$v_i \triangleright v_j$: $v_j$ must be executed before $v_i$ when $v_i$ and $v_j$ are both executed |

$v_i \triangleright \triangleright v_j$: $v_j$ must be executed before $v_i$ every time $v_j$ is executed |

$v_i \wedge v_j$: $v_i$ and $v_j$ are executed in parallel |

$v_i \lor v_j$: either $v_i$ or $v_j$ is executed

Identification of Sequential Relations

In this section, we present the principles for identifying sequential relations. The basic idea is as follows. If activity $v_i$ uses some input data $d$ produced by activity $v_j$, then $v_j$ cannot be executed before $v_i$ is executed. Otherwise, data $d$ will not be available for $v_j$ to use.

Proposition 1. Let $W=\langle A, s, e, R(\text{type}), L, A, C \rangle$ be an acyclic workflow model. Let $A$ be complete and concise. For any $v_i, v_j \in A$: $v_i \Rightarrow v_j$ implies $v_i \triangleright v_j$.

3 It is possible that under different conditions, various activities produce the same output. For simplicity, we consider the data produced by different activities as different data outputs.
Proof: \( v_i \Rightarrow v_j \) indicates two possibilities: (1) there exists a data item \( d \) such that \( d \in O_{iv} \) and \( d \in I_{iv} \); or (2) there exists another activity \( v_m \) such that \( v_i \Rightarrow v_m \) and \( v_m \Rightarrow v_j \), namely \( \exists d_1, d_2 \) that \( d_1 \in O_{iv}, d_1 \in I_{iv}, d_2 \in O_{im}, \) and \( d_2 \in I_{iv} \).

a) We first prove that under Condition (1), \( v_i \Rightarrow v_j \) implies \( v_i \Rightarrow v_j \). Assume that \( v_i \Rightarrow v_j \) does not hold, then by Definition 14, there exists no firing sequence \( \sigma(v_i, v_j) \). Therefore, \( v_i \notin \sigma(s, v_j) \) holds for every \( \sigma(s, v_j) \). Since \( A \) is a complete and concise workflow, there exists no other \( v \in A \) that can produce \( d \) as output. Then, \( d \) will not be available for \( v_j \) to use when it is executed. Therefore, \( v_i \Rightarrow v_j \) must hold such that \( d \) can be produced when \( v_j \) is executed.

b) We then prove that under Condition (2) \( v_i \Rightarrow v_j \) implies \( v_i \Rightarrow v_j \). Since there exists activity \( v_m \) such that \( \exists d_1, d_2 \) that \( d_1 \in O_{iv}, d_1 \in I_{iv}, d_2 \in O_{im}, \) and \( d_2 \in I_{iv} \), according to the proof above \( v_i \Rightarrow v_m \) and \( v_m \Rightarrow v_j \) must hold. Therefore, there exist \( \sigma(v_i, v_m) \) and \( \sigma(v_m, v_j) \). We can construct \( \sigma(v_i, v_j) = \sigma(v_i, v_m) \cup \sigma(v_m, v_j) \). Therefore, \( v_i \Rightarrow v_j \) holds.

By a) and b), we conclude \( v_i \Rightarrow v_j \) implies \( v_i \Rightarrow v_j \).

**Corollary 1.** Let \( v_i \) and \( v_j \) be two activities. \( v_i \in \Gamma_{vj} \) implies \( v_i \Rightarrow v_j \).

**Proof:** Because of \( v_i \in \Gamma_{vj} \) by Definition 8, \( v_i \Rightarrow v_j \). By Proposition 1, we conclude \( v_i \Rightarrow v_j \).

**Proposition 2.** Let \( W=<A, s, e, R(type), L, A, C> \) be an acyclic workflow model. Let \( A \) be complete and concise. For any \( v_i, v_j \in A \), \( v_i \Rightarrow v_j \) or \( v_j \Rightarrow v_i \) must hold.

Proof: Because of \( v_i \Rightarrow v_j \) or \( v_j \Rightarrow v_i \), by Definition 4, there exists a data item \( d \) such that \( d \in O_{iv}, d \in I_{iv} \) or \( d \in \Gamma_{iv} \). Then for every \( \sigma(s, v_j) \), \( v_i \in \sigma(s, v_j) \) holds because otherwise \( d \) will not be available for \( v_j \) to use and by Definition 3, \( v_j \) cannot be executed because \( d \in \Gamma_{iv} \) or \( d \in \Gamma_{iv} \). By Definition 14, \( v_j \Rightarrow v_i \).

**Corollary 2.** Let \( W=<A, s, e, R(type), L, A, C> \) be an acyclic workflow model. Let \( v_i, v_j \in A \), if \( v_i \Rightarrow v_j \) and there exists no \( v_k \in A \) such that \( v_i \in \Gamma_{vk} \) and \( v_k \in \Gamma_{vj} \), then \( v_i \) and \( v_j \) can be arranged as \( v_i \Rightarrow v_j \) unless a routing activity must be placed between \( v_i \) and \( v_j \).

Proof: We prove by contradiction. Assume that there exists activity \( v_k \) that has to be placed between \( v_i \) and \( v_j \). Then \( v_i \Rightarrow v_k \) and \( v_k \Rightarrow v_j \) must hold. Then by Proposition 1 and Corollary 1, \( v_i \Rightarrow v_k \) and \( v_k \Rightarrow v_j \) must hold because otherwise \( v_i \Rightarrow v_k \) and \( v_k \Rightarrow v_j \) would be unnecessary. Therefore, \( v_i \in \Gamma_{vk} \) and \( v_k \in \Gamma_{vj} \), which contradicts the condition that there exists no \( v_k \in A \) such that \( v_i \in \Gamma_{vk} \) and \( v_k \in \Gamma_{vj} \). Therefore, \( v_i \) and \( v_j \) can be arranged as \( v_i \Rightarrow v_j \).

Proposition 1 and Corollary 1 provide the principles on how to design the precedence relations “\( \Rightarrow \)” based on data dependencies. Proposition 2 describes when the strong precedence relations “\( \Rightarrow \)” should be used. Corollary 3 describes the conditions when a direct arc may be added.

**Identification of Parallelism and Conditional Routing**

If two activities \( v_i \) and \( v_j \) are executed in parallel or in two different conditional routing branches, \( v_j \) cannot use any data \( d \) produced by activity \( v_i \) as input. Otherwise, it is possible that data \( d \) will not be available for \( v_j \) to use at the time of the execution of \( v_j \).

**Proposition 3.** For any two activities \( v_i \) and \( v_j \), \( v_i \land v_j \) or \( v_i \lor v_j \) implies \( v_i \Rightarrow v_j \).

Proof: We prove by contradiction. Assume \( v_i \land v_j \) or \( v_i \lor v_j \), and \( v_i \Rightarrow v_j \). Given \( v_i \Rightarrow v_j \) or \( v_j \Rightarrow v_i \) by Proposition 1, we conclude that \( v_i \Rightarrow v_j \) or \( v_j \Rightarrow v_i \), i.e., there exist \( \sigma(v_i, v_j) \neq \emptyset \) or \( \sigma(v_j, v_i) \neq \emptyset \). As such, it is not possible that \( v_i \land v_j \) or \( v_i \lor v_j \) implies \( v_i \Rightarrow v_j \). This contradicts with the assumption. Therefore, \( v_i \land v_j \) or \( v_i \lor v_j \) implies \( v_i \Rightarrow v_j \).

**Corollary 3.** Let \( W=<A, s, e, R(type), L, A, C> \) be an acyclic workflow model. Let \( S=\{v_{ix}\} \mid x \in \{1, 2, \ldots, m\} \) and \( v_{ix} \in A \) and \( v_{ix} \in T \) and \( T=\{v_{ij} \mid y \in \{1, 2, \ldots, n\} \) and \( v_{ij} \in A \) and \( v_{ij} \in S \} \).

1. If \( v_{ix} \land v_{ij} \) holds for every pair of \( v_{ix} \in S \) and \( v_{ij} \in T \), then \( S \preceq T \).

2. If \( v_{ix} \lor v_{ij} \) holds for every pair of \( v_{ix} \in S \) and \( v_{ij} \in T \), then \( S \preceq T \).

Proof: Given \( v_{ix} \land v_{ij} \) or \( v_{ix} \lor v_{ij} \) where \( x \in \{1, 2, \ldots, m\} \) and \( y \in \{1, 2, \ldots, n\} \), by Proposition 2, \( v_{ix} \Rightarrow v_{ij} \) holds for every pair of \( v_{ix} \in S \) and \( v_{ij} \in T \). By Definition 9, we conclude \( S \preceq T \).

Proposition 3 and Corollary 3 suggest that two independent sets can have the relations of “\( \land \)” or “\( \lor \)”, which corresponds to parallelism and conditional routing. Next, we differentiate the conditions under which parallelism should be used from those under which conditional routing should be used.
Proposition 4. Let \( v_i, v_j, \) and \( v_k \) be activities. Given \( v_i \Join v_j, v_i \implies v_k \) or \( v_i \ag v_k \), and \( v_j \implies v_k \) or \( v_j \ag v_k \), the activity relation between \( v_i \) and \( v_j \) cannot be \( v_i \Join v_j \).

Proof: We prove by contradiction. Assume that \( v_i \Join v_j \) holds. Given \( v_i \implies v_k \) or \( v_i \ag v_k \), and \( v_j \implies v_k \) or \( v_j \ag v_k \), by Proposition 2, \( v_i \implies v_k \) and \( v_j \implies v_k \) must hold, which we refer to as “strong precedence”. Given \( v_i \Join v_j \), by Property 2, for each \( \sigma(s, e) \), if \( v_j \leq \sigma(s, e) \), then \( v_j \geq \sigma(s, e) \), i.e., for every \( v_j \leq \sigma(s, e) \) if \( v_j \geq \sigma(s, e) \) then \( v_j \geq \sigma(s, e) \), which contradicts with the notation of “strong precedence”. Therefore, given \( v_i \Join v_j \), \( v_i \implies v_k \) or \( v_i \ag v_k \), and \( v_j \implies v_k \) or \( v_j \ag v_k \), \( v_i \Join v_j \) cannot be true.

Proposition 5. Let \( v_i \) and \( v_j \) be two activities and \( v_i \Join v_j \). Let \( c_1 = f_1(D_1) : \text{Execute} (V_1) \) and \( c_2 = f_2(D_2) : \text{Execute} (V_2) \) be the conditional routing constraints for \( v_i \) and \( v_j \) to be executed, i.e., \( v_i \in V_1 \) and \( v_j \in V_2 \).

1. If there exist \( d_{a} \in D_1 \) and \( d_{b} \in D_2 \) such that \( f_1(D_1) \) and \( f_2(D_2) \) are both true, the activity relation between \( v_i \) and \( v_j \) cannot be \( v_i \Join v_j \).

2. If there exist \( d_{a} \in D_1 \) and \( d_{b} \in D_2 \) such that \( f_1(D_1) \) is true and \( f_2(D_2) \) is false, the activity relation between \( v_i \) and \( v_j \) cannot be \( v_i \Join v_j \).

Proof: First we prove 1. Let \( d_{a} \in D_1 \) and \( d_{b} \in D_2 \) such that \( f_1(D_1) \) and \( f_2(D_2) \) are both true. Then, there are situations that \( v_i \) and \( v_j \) both have to be executed. By Property 2, the activity relation between \( v_i \) and \( v_j \) cannot be \( v_i \Join v_j \). Then, we prove 2. Let \( d_{a} \in D_1 \) and \( d_{b} \in D_2 \) such that \( f_1(D_1) \) is true and \( f_2(D_2) \) is false, there exists at least a situation where \( v_i \) and \( v_j \) cannot both be executed. By Property 1, the activity relation between \( v_i \) and \( v_j \) cannot be \( v_i \Join v_j \).

Proposition 6. Let \( W = A, s, e, R(\text{type}), L, A, C > \) be an acyclic workflow model. Given \( v_i \Join v_j \), there exists \( r \in R(XORSplit) \) such that \( r \in \sigma(v_i, v_j) \) for every \( \sigma(v_i, v_j) \).

Proof: Given \( v_i \Join v_j \), by Definition 4 there exists a data item \( d \) such that \( d \in \Gamma_{v_i}, d \in \Omega_{v_j} \). By Definition 3, there exists \( c = f(D) : \text{Execute} (V) \) such that \( d \in D \) and \( v_j \in V \). Because of \( v_i \Join v_j \), by Proposition 2, \( v_i \Join v_j \). Then, for every \( \sigma(s, e) \), if \( v_j \in \sigma(s, e) \) then \( v_j \in \sigma(s, e) \). When an instantiation of \( d \) makes \( f(D) \) false, \( v_j \) is executed and \( v_j \) is not executed, i.e., \( v_j \in \sigma(s, e) \) and \( v_j \not\in \sigma(s, e) \). By Definition 12, we conclude that there exists \( r \in R(XORSplit) \) such that \( r \in \sigma(v_i, v_j) \) for every \( \sigma(v_i, v_j) \).

Propositions 4, 5, and 6 describe the situations where parallelism can be used and where conditional routing must be used in a workflow model. Table 3 summarizes the results of Propositions 1-6. Essentially Propositions 1-6 provide the principles on how to derive activity relations from dataflow. Once activity relations are known, we can decide where to place routing activities and where to add direct arcs as listed in the column of Implications in Table 3. Due to space limits, the related algorithms are reported in another paper (Sun and Zhao 2006).
### Table 3. Summary of Design Principles

<table>
<thead>
<tr>
<th>Activity Relations</th>
<th>Design Principles</th>
<th>Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_i \ast v_j$</td>
<td>May occur if $v_i \Rightarrow v_j$ and there exists no $v_n \in A$ such that $v_i \in \Gamma_{v_n}$ and $v_n \in \Gamma_{v_j}$ (By Corollary 2)</td>
<td>$v_i$ immediately precedes $v_j$</td>
</tr>
<tr>
<td>$v_i \triangleright v_j$</td>
<td>Occurs if $v_i \Rightarrow v_j$ or $v_i \in \Gamma_{v_j}$ (By Proposition 1 and Corollary 1)</td>
<td>There exists a firing sequence from $v_i$ to $v_j$</td>
</tr>
<tr>
<td>$v_i \triangleright\triangleright v_j$</td>
<td>Occurs if $v_i \Rightarrow v_j$ or $v_i \in \Gamma_{v_j}$ (By Proposition 2)</td>
<td>$v_i$ is included in all firing sequences to $v_j$</td>
</tr>
<tr>
<td>$v_i \land v_j$</td>
<td>Occurs if $v_i \land v_j$ and there exists no conditional routing constraints where $v_i$ is executed and $v_j$ is not (By Proposition 3, Proposition 5)</td>
<td>An ANDSplit can be placed before $v_i$ and $v_j$, and an ANDJoin can be placed after $v_i$ and $v_j$</td>
</tr>
<tr>
<td>$v_i \lor v_j$</td>
<td>Occurs if 1) $v_i \lor v_j$, 2) no other activity has mandatory or execution dependency on both $v_i$ and $v_j$, 3) there exists no routing constraints where $v_i$ and $v_j$ can be both executed, and 4) $v_i$ and $v_j$ both have execution dependency on other activities. (By Propositions 3, 4, 5, and 6).</td>
<td>An XORSplit is before $v_i$ and $v_j$, and an XORJoin is after $v_i$ and $v_j$</td>
</tr>
</tbody>
</table>

### Table 4. Activity Relations in the Order Processing Workflow

<table>
<thead>
<tr>
<th>Activity relations</th>
<th>Application to the order processing workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_i \ast v_j$</td>
<td>$v_j \ast v_3$ and $v_j \ast v_6$ are possible</td>
</tr>
<tr>
<td>$v_i \triangleright v_j$</td>
<td>$v_i \triangleright v_2, v_i \triangleright v_3, v_i \triangleright v_5, v_i \triangleright v_6$</td>
</tr>
<tr>
<td>$v_i \land v_j$</td>
<td>${v_2} \land {v_3, v_5}$, ${v_3, v_5} \lor {v_4, v_6}$, ${v_3, v_5} \lor {v_2, v_3, v_5}$, ${v_3, v_5} \lor {v_3, v_5}$</td>
</tr>
<tr>
<td>$v_i \lor v_j$</td>
<td>${v_2} \lor {v_3, v_5}$, ${v_3, v_5} \lor {v_4, v_6}$, ${v_3, v_5} \lor {v_2, v_3, v_5}$</td>
</tr>
</tbody>
</table>

We apply Propositions 1-6 and Corollaries 1-3 to design the order processing workflow. Considering the conditional routing constraints $c_1 = (d_2 \leq d_6; \text{Execute}(v_2, v_3))$ and $c_2 = (d_0 \geq d_3; \text{Execute}(v_2, v_4, v_6))$ and the independent sets $\{v_2\} \not\approx \{v_2, v_3\}$, $\{v_2\} \not\approx \{v_2, v_5\}$, $\{v_2, v_3\} \not\approx \{v_4, v_6\}$, $\{v_2, v_5, v_3\} \not\approx \{v_4, v_6\}$, and $\{v_2, v_3\} \not\approx \{v_3, v_5\}$, we can get the workflow design results shown in Table 4 and Figure 5. Note that $\{v_3, v_5\} \lor \{v_2, v_4, v_6\}$ dominates $\{v_2\} \lor \{v_3, v_5\}$ and $\{v_3, v_5\} \lor \{v_2, v_4, v_6\}$.

![Figure 5. Workflow Design Based on Dataflow Analysis](image-url)
Conclusions

In this paper, we proposed the concept of activity relations and advocated the use of activity relations in designing workflow models through dataflow analysis. Our overall goal is to address the workflow design problem in two steps: deriving the activity relations from dataflow first, and then identifying the possible control flow structures. As the foundation of a formal workflow design methodology, this paper focused on the first step and provided design guidelines for deriving activity relations.

Note that dataflow analysis may help generate more than one candidate control flow model. Other factors such as resource limitations and cost optimization need to be taken into consideration when determining the final model. Moreover, we assume that the set of activities in a process are known at the beginning of the design process. In future research, we plan to focus on issues related to the identification of candidate activity sets. We will also extend our work in several other directions: the automation of the workflow design process, an empirical comparison with existing workflow design methods, and an investigation of the mechanisms for handling cyclic workflows.

References
