Decision Support Systems 55 (2013) 270–283

Contents lists available at SciVerse ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss

Formal workflow design analytics using data flow modeling

Sherry X. Sun *, J. Leon Zhao

Department of Information Systems, College of Business, City University of Hong Kong, Hong Kong

ARTICLE INFO

Article history:
Received 24 August 2011
Received in revised form 25 September 2012
Accepted 21 January 2013
Available online 16 February 2013

Keywords:
Workflow design
Data dependency
Activity dependency
Activity relations
Business process automation

ABSTRACT

Workflow design has become a critical function in enterprise information management. However, only scant research attention has been paid to formal workflow design methodologies. As a result, existing design methods in business process management remain a manual and experiential effort and result in inefficiency in design tasks and potential errors in workflow models. Considering that there are hundreds and thousands of business processes in organizations worldwide, overcoming this deficiency will have an enormous technical and economic impact on enterprise information management. In this paper, we investigate the possibility of incorporating formal analytics into workflow design, thus alleviating the intellectual challenge faced by business analysts when creating workflow models. The workflow design analytics we propose helps construct a workflow model based on information about the relevant activities and their associated data. In addition, our workflow design approach also helps determine whether the given information is sufficient for generating a workflow model and ensures the avoidance of certain workflow anomalies. The significance of our study is to enable the transformation of workflow design from a manual and experiential effort into a more systematic and rigorous approach.

1. Introduction

Over the last decade during the drive towards e-business, many organizations have realized the importance of Internet-based business process automation as they face the challenge of obtaining competitiveness in the global marketplace. To achieve high levels of business process efficiency, many organizations resort to workflow management systems to integrate business activities that span multiple functional units, such as human resources, marketing, and manufacturing [31,33]. As a prerequisite for effective workflow management, a workflow model is required to specify the execution sequence of activities needed to support certain business functions, such as producing a product or offering a service. Thus, it is critical to design correct workflow models efficiently according to specific business requirements.

The existing workflow design approaches, such as the participative approach widely used in practice, mainly focus on collecting business requirements rather than providing a rigorous procedure for generating workflow models. The lack of formal design approaches often causes design problems. Recent empirical studies have shown an error rate of 10%–20% from over 2000 process models used in different industry practice, which include the widely used SAP reference model, the models used in a process reengineering project in the service sector in Germany, the models documented in the financial industry in Austria, the models used by consulting firms, and the models from IBM projects [20–22]. With the empirical evidence of the design problems, the importance of formal design methods starts to be recognized.

The process of designing workflow models can be partitioned into three stages. The first stage is *business analysis* consisting of design activities to understand business objectives, system environment, domain knowledge, and process management rules [37,38]. The second stage is *process analysis* that deals with the issues of activity identification and dataflow specification [14,32]. Activities can be identified by workers and managers in various business units since they generally know what they do on the daily basis. Input and output data for each task, i.e. dataflow, can be identified from documents, forms, and databases used in the business process [15,27,35].

The third stage, namely *model construction* to derive the correct activity sequencing, is an intellectually challenging step. First, no clear guidelines exist to help determine whether a given set of activities and related data are sufficient for designing a workflow. Moreover, due to the human cognitive limit, great effort is required to figure out all interrelations of different steps in a complex process with no guarantee for the resulting model to be error-free [21,22]. These challenges are compounded by the fact that a workflow can involve multiple business units and different organizations located in different geographic locations.

The common wisdom for dealing with potential errors in the stage of model construction has been to confirm the workflow models through simulation and verification before deploying them in practice [2,7,17]. However, it is a fundamental principle in software engineering that design errors should be prevented as early as possible [21,23]. The later the errors are identified, the more cost and effort are needed to fix the errors. This principle also holds for designing and automating business processes [5]. Consequently, we believe that once a set of activities and their

* Corresponding author.
E-mail addresses: sherry.sun@cityu.edu.hk (S.X. Sun), jlzhao@cityu.edu.hk (J.L. Zhao).

0167-9236/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dss.2013.01.028

input and output data have been identified in the stage of *process analysis*, it is possible to derive a workflow model through analyzing the data dependencies among activities, namely dataflow analysis, so as to avoid certain potential design errors in the first place. Of course, this assumes the knowledge of all the activities and their input and output data can be obtained in the stage of *process analysis* given that the necessary information can be identified from interviews, discussion sessions, and documents by using the existing design methods [13,15,27,35]. This simplification makes it manageable to develop a first-cut workflow design method.

The scope of our research is that starting from a dataflow to derive a corresponding acyclic workflow consisting of five basic constructs [1,2,28]. A dataflow is specified with a set of business activities, the related routing constraints defined on the given business activities, and input and output data items for each business activity. In this paper, we focus on determining relations between business activities from the dataflow and then developing a control flow consistent with the dataflow. The basic idea is to decide how activities should be sequenced in a workflow through examining the dataflow in a process, i.e., the transformation from input data to output data via a sequence of activities. First, we define certain dataflow properties, including well-connectedness, completeness, and conciseness, to verify if a dataflow specification provides correct and sufficient information for constructing a workflow model. Moreover, we develop design principles for deriving potential activity execution steps, referred to as "activity relations", from data dependency analysis. Third, we develop an analytical method to enable the creation of complete workflow models from activity relations. Fourth, we simplify workflow design by applying the concept of inline block and decoupling the issue of model correctness from the issue of workflow efficiency. Comparing with the existing workflow design methods that involve largely manual and experiential efforts, our method converts the design of complex workflow models from a task that depends on the knowledge and experience of workflow developers to a task that is supported with a systematic and mathematically rigorous approach. Therefore, we refer to such an analytical approach as workflow design analytics.

The impact of workflow design analytics is significant. First, workflow design analytics builds the foundation for a formal approach to deal with the complexity of workflow model construction, thus alleviating the intellectual challenge faced by business analysts. Second, analytical workflow design can help verify the sufficiency of the given process information, including activities and associated data, for workflow model construction. Moreover, our approach can help eliminate certain potential workflow flaws caused by violation of data dependencies [30] and inappropriate model structures [7]. This feature of our approach will reduce the costs of generating error-free workflow models. As our research goal is to enable the transformation of workflow design from a manual and experiential effort into a more systematic and rigorous approach, the contribution of our research goes well beyond the computational formalism presented in this paper. Successful application of our approach will have considerable economic implications for companies with complex business processes.

The remainder of this paper is organized as follows. Section 2 reviews the relevant literature. Section 3 presents the data dependency concepts needed for workflow design. Section 4 defines the principles for deriving basic workflow structures from analyzing data dependencies. Section 5 demonstrates how to generate complex workflow models step by step. Finally, Section 6 concludes the paper by outlining our contributions and future directions.

2. Literature review

Most work in the area of workflow management focuses on representing activity sequences through various graph-oriented modeling languages. For example, both Event-driven Process Chains (EPC), used to describe the SAP reference model [9], and Business Process Modeling Notation (BPMN), developed by the Business Process Management Initiative (BPMI), are notations intended not only for technical developers but also for business analysts. As efforts to make the two graphic notations formal, different formalisms have been added to both EPC and BPMN [11,20]. Another popular graphic tool, UML activity diagram, plays an important role in workflow modeling [12]. Due to the lack of formal foundation [19] in UML activity diagrams, propositional logic has been applied to verifying UML activity diagrams [7].

As a formal modeling language, Petri nets [1,2] can help determine if an activity sequence contains errors, such as deadlock, activities without termination or activation, and infinite cycles, due to its abundance of analysis techniques. Based on Petri nets, a new workflow modeling language, Yet Another Workflow Language (YAWL), has recently been developed with some extended expressive power for specifying different workflow structures [4]. Metagraphs, a graphic structure extended from directed graphs and hypergraphs, has also been used as a formal workflow modeling approach to supports analysis of the data transformation through a sequence of activities [6].

While a significant amount of research in the workflow area has focused on modeling, verification, and analysis issues [2,7,17], formal methods to generate workflow models, namely the workflow design approaches, have not received enough attention in the literature [31]. The most well-known workflow design approach is the participative approach [13], which adopts the principle of joint application design (JAD) consisting of a variety of methods for conducting workshops where users and technical developers work together to define requirements of information systems [10]. This approach is useful for collecting information needed for workflow design, such as business documents, business rules, relevant business activities, and data elements critical to execute the activities. However, the participative approach does not offer formalism for generating workflow models in a rigorous manner.

The method of Product-Based Workflow Design uses the relationship among data elements derived from product specifications as a starting point for workflow design [27,35]. Moreover, cost and flow time are considered as criteria for selection of workflow models. The policy-driven workflow mining method [15] focuses on deriving a workflow model from documented business policies. However, formal procedures are still needed on how to determine activity sequences and how to use parallelism and conditional routing on the basis of analyzing relationships among data.

As an important step in workflow management, dataflow analysis can help discover dataflow errors hidden in a workflow model after the model has been created [25,29,32,34]. However, it should be more efficient to prevent dataflow errors by taking dataflow into consideration at the time the workflow model is created. Therefore, in this paper, we propose to start workflow design with analyzing data dependency, a type of flow dependency that arises whenever one activity uses a resource produced by another activity [18]. Moreover, we propose a new concept called "activity relations" as a foundation for a formal workflow design method. The concept of activity relations is inspired by the concept of "log-based ordering relations" used in [3]. However, the concept of "activity relations" is different from "log-based ordering relations" in that the former describes the structures of a workflow model and the latter describes the event sequences found in an event log.

The detection and verification of errors have also been investigated in the area of rule-based systems [24,26,30,37,38]. Validation of rule-based systems requires the identification of anomalies such as redundancy, ambivalence, circularity, and deficiency at a large-scale since a rule-based system can have thousands of rules. However, workflow design faces a different challenge given that each step in a business process typically produces a data output while a rule in a rule-based system may or may not produce any data. Further, consideration of the two intertwined aspects, i.e., dataflow and control flow, is not explicit in rule-based systems but critical for preventing structural errors in workflow systems.

As shown above, the number of publications related to workflow design is still quite small, particularly in comparison with the area of database design. However, workflow design is becoming increasingly

important given that many organizations are rushing to automate their business processes via workflow systems, indicating that more research is needed in this area.

3. Data and activity dependencies analysis

In this section, we first extend the concepts of data dependency analysis, originally developed for verifying dataflow anomalies in a workflow model [32], for the purpose of workflow design. We then introduce an example of order processing and show how data dependencies can be decided using this example. Further, we explain how to derive activity dependencies from data dependencies and briefly discuss how to determine whether a given set of data dependencies is sufficient for deriving a proper workflow model.

3.1. Concepts of data dependencies

This section classifies data dependencies into three types, unconditional, conditional, and execution data dependencies. Following the convention in relational databases [34], we consider a data item to be an atomic data element that can be modeled as an attribute in a relational database. An activity v can perform read/write operations on a data item d which is an input/output of v [32]. To handle changes in a workflow, multiple data items can be used to capture a change rather than simply overwriting the existing data because the history of how a process is conducted usually needs to be kept. For example, we may need to track when the order is placed, fulfilled, and shipped. The order status can be specified using "order_placed", "order _fulfilled", and "order_shipped", each of which can be set "Yes" or "No". To track all states of the same data object, a Structure data type $d'\{d_1, \ldots, d_i, d_{i+1}, \ldots, d_n\}$ [34] should be used where d_i is a data item capturing a state of d'. An example of Structure is Order{order_placed, order_fulfilled, order_shipped}. Therefore, we do not consider the operation of "update" in this paper.

Next, we define the concept of routing constraint because it is useful for analyzing data dependencies.

Definition 1. (Routing Constraint) A routing constraint c is denoted as $c = f(D)$:*Execute*(V), where D is a set of data items and $f(D)$ is a logic expression of D.

For example, when the condition "travel expense d is greater than $5000" is true, the activity v *approve the travel application* will be executed. This is written as $c = (d > 5000)$: *Execute*(v).

Definition 2. (Data Dependency) In general, each activity v takes in a set of input data items I_v and produces a set of output data items O_v. This is called a data dependency for v, denoted as $\lambda_v(I_v, O_v)$.

Definition 3. (Unconditional, Conditional, and Execution Data Dependency) Data dependencies for activity v can be further categorized into three types, namely unconditional, conditional, and execution data dependencies and denoted as $\lambda^u_v(I^u_v, O_v)$, $\lambda^c_v(I^c_v, O_v)$, and $\lambda^e_v(I^e_v, O_v)$, respectively, where I^u_v is the unconditional input data set always needed by activity v, I^c_v is the conditional input data set required by activity v when certain routing constraints are satisfied, and I^e_v is the execution input data set determining whether activity v needs to be executed.

Note that the total set of input data is the sum of all three types of input data, denoted as $I_v = I^u_v \cup I^c_v \cup I^e_v$ where $I^u_v \cap I^c_v = \varnothing$. An example is given next to illustrate these data dependencies.

3.2. Order processing example

As a running example, the relevant activities and key data items for order processing are identified in Fig. 1. Table 1 shows several business rules and the corresponding routing constraints. We determine unconditional, conditional, and execution data dependencies as explained below.

- *Example 1* (*Unconditional data dependency*): In the order processing example, activity v_9, *update product inventory*, always uses data items d_0, *quantity available*, d_4, *quantity ordered*, and d_{14}, *shipping date*, as input unconditionally in order to produce data item d_{10}, *updated availability*. Thus the unconditional data dependency for v_9 is written as $\lambda^u_{v9}([d_0, d_4, d_{14}], [d_{10}])$.
- *Example 2* (*Conditional data dependency*): Further, data item d_8, *approved by manager*, is needed only if a customer does not have a good credit rating. Otherwise, activity v_6, *process down payment*, is enacted to generate output d_9, *down payment paid*, regardless of the value of d_8. Thus, we have a conditional data dependency $\lambda^c_{v6}([d_8], [d_9])$.
- *Example 3* (*Execution data dependency*): In Table 1, the routing constraint $c_1 = \{d_5 =$ "*No*": *Execute*$(v_3, v_8)\}$ says that a replenishment order should be sent when a customer requests more than what is available. That is, the input d_5, *product availability confirmed*, determines whether to execute activity v_3, *send replenishment order*. Thus, the output from v_3, d_{11}, *replenishment quantity*, and d_{12}, *replenishment date*, have execution dependency on d_5, i.e., $\lambda^e_{v3}([d_5], [d_{11}, d_{12}])$.

Table 2 shows all unconditional, conditional, and execution input data for the order processing example. Note that the start and end activities are dummy activities, indicating the starting and ending points of a workflow. For the purpose of data dependency analysis, we assume that the start activity provides all the initial data available when the workflow starts. This initial data set comes from other sources external to the workflow, including other workflows, given that a workflow often interacts with other workflows through information exchange in an organizational setting. Moreover, we model the final output data produced by a workflow as the input data for the end activity because the final output data from the workflow needs to be produced before the end activity can be executed. The final output data can be sent to other workflows for process collaboration. In Table 2, the output of the start activity, d_0, *quantity available*, can be provided by a workflow for inventory management and the final output data of the order processing d_1, d_5, d_6, d_9, d_{10}, d_{11}, d_{12}, d_{13}, d_{14}, d_{15}, d_{16}, d_{17}, and d_{18}, modeled as the input to the end activity, can be input for workflows of accounting and shipping management.

For brevity, we use $\lambda_v(I_v, O_v)$ to represent the sum of $\lambda^u_v(I^u_v, O_v)$, $\lambda^c_v(I^c_v, O_v)$, and $\lambda^e_v(I^e_v, O_v)$ and $\Lambda = \{\lambda_v(I^u_v | I^c_v | I^e_v, O_v) | v \in V\}$ or simply $\Lambda = \{\lambda_v(I_v, O_v) | v \in V\}$ to represent all the data dependencies for a set of activities V, which is the *dataflow* of V as exemplified in Table 2.

3.3. Concepts of activity dependencies

Next, we discuss how to derive activity dependencies from data dependencies. Activity dependency occurs when certain input data of an activity is derived from the output data of another activity directly or indirectly. Further, when a data item d, an output from activity u, decides if the execution of activity v is needed, execution dependency occurs between u and v. When d is needed as an unconditional or conditional input by v, v must depend on u for data item d and we consider the dependency between u and v mandatory.

Definition 4. (Activity Dependency): Activity v_i is directly dependent on another activity v_j, denoted as $v_j \rightarrow v_i$, if $O_{vj} \cap I_{vi} \neq \varnothing$. If $O_{vj} \cap I^e_{vi} \neq \varnothing$ we say that v_i has an execution dependency on v_j, denoted as $v_j \rightarrow_e v_i$; otherwise, we say v_i has a mandatory dependency on v_j, denoted as $v_j \rightarrow_m v_i$, if either $O_{vj} \cap I^u_{vi} \neq \varnothing$ or $O_{vj} \cap I^c_{vi} \neq \varnothing$. Furthermore, v_i is indirectly dependent on v_j if $v_j \rightarrow x_1, x_1 \rightarrow x_2, \ldots, x_{i-1} \rightarrow x_i$ and $x_i \rightarrow v_i$ where x_i is some activity and $i \geq 1$, denoted as $v_j \Rightarrow v_i$.

Activities	v_{12}: notify of the sales manager's	d_6: down payment amount
v_1: process order	"No" decision	d_7: customer credit rating
v_2: check product availability	v_{13}: notify order fulfillment	d_8: approved by manager
v_3: send replenishment order	e: end activity	d_9: down payment paid
v_4: verify credit	s: start activity	d_{10}: updated availability
v_5: order approval by sales manager		d_{11}: replenishment quantity
v_6: process down payment	**Data Items**	d_{12}: replenishing date
v_7: confirm order	d_0: quantity available	d_{13}: confirmation number
v_8: send back order notice	d_1: product IDs	d_{14}: shipping date
v_9: update product inventory	d_2: order ID	d_{15}: back order notice status
v_{10}: make shipment	d_3: customer ID	d_{16}: payment made
v_{11}: process payment	d_4: quantity ordered	d_{17}: order fulfilled
	d_5: product availability confirmed	d_{18}: approval notified

Fig. 1. The relevant activities and key data items in the order processing example.

Note that, in Definition 4, when both mandatory and execution dependencies can occur between two activities, execution dependency takes precedence over mandatory dependency and a mandatory dependency between two activities v_i and v_j indicates that there is no execution dependency between v_i and v_j. The dependency $s \rightarrow v$ indicates that activity v uses external data as input and the dependency $v \rightarrow e$ indicates that activity v directly produces the final output from a workflow.

Example 4 (*Activity dependency*): We illustrate Definition 4 by deriving all direct activity dependencies in the order processing example (Fig. 2). Note that a data item d can be both an unconditional (or conditional) input and an execution input for an activity. For instance, given that the customer will be notified of the sales manager's decision if the order is disapproved by the sales manager, i.e., $c_6 = \{(d_8 = \text{"No"})$: *Execute*$(v_{12})\}$ (Table 1), the execution of v_{12} depends on the value of d_8, an output from activity v_5, leading to the execution activity dependency between v_5 and v_{12}. Moreover, data item d_8, *approved by manager*, is also an unconditional input data for activity v_{12}, *notify of the sales manager's "No" decision*. Therefore, the activity dependency between v_{12} and v_5 can be mandatory. Given the dominance of execution activity dependency over mandatory activity dependency, we have $v_5 \rightarrow_e v_{12}$. Similarly, the routing constraint $c_5 = \{(d_8 = \text{"Yes"})$: *Execute*$(v_6)\}$ in Table 1 indicates that the execution of v_6 depends on the value of d_8, the output data from v_5. Thus, we have $v_5 \rightarrow_e v_6$ (Fig. 2) regardless of other data dependencies for v_5 and v_6. In addition to the direct activity dependencies, activity dependency can be indirect. For instance, v_6 has an indirect dependency on v_2, i.e., $v_2 \Rightarrow v_6$, through activity v_4.

Note that, in Fig. 2, $s \rightarrow v_2$ and $s \rightarrow v_9$ indicate that activity v_2 and v_9 use some external data as input. Moreover, the activity e directly depends on v_1, v_2, v_3, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, and v_{13}, which directly produce the final output data of the workflow.

Definition 5. (Direct Requisite Set Δ_v) Given a set of activities V and their data dependencies $\Lambda = \{\lambda_v\ (I_v, O_v) |\ v \in V\}$, $\Delta_v = \{u |\ u \rightarrow v\}$ is the direct requisite set for activity v.

In Fig. 2, we can easily find the set of activities that each activity directly depends on in the order processing example. For instance, activity v_{11} directly depends on activities v_1, v_4, v_5, v_7, and v_{10}. This set is called the direct requisite set for activity v_{11}. The set of activities depended directly or indirectly by activity v is referred to as the *full requisite set* of v, as defined below.

Definition 6. (Full Requisite Set Γ_v) Given a set of activities V and their data dependencies $\Lambda = \{\lambda_v(I_v, O_v) |\ v \in V\}$, $\Gamma_v = \{u |\ u \rightarrow v \text{ or } u \Rightarrow v\}$ is the full requisite set of v.

Table 3 lists the full requisite set Γ_v for all the activities in the order processing example.

3.4. Properties of dataflow

Next, we define four properties of dataflow important for workflow design. Given a set of activities V and its dataflow $\Lambda = \{\lambda_v(I_v, O_v) |\ v \in V\}$, let $O_s = I_0$, $I_e = O_0$, and $I_s = O_e = null$ hold for $s, e \in V$ where I_0 is the initial

Table 1
Routing constraints for the order processing example.

Business rule	Routing constraint
If a customer orders more than what is available, a replenish order should be sent to the manufacturer and a back order notice should be sent to the customer.	$c_1 = \{d_5 = \text{"No"}$: *Execute*$(v_3, v_8)\}$
If there are enough products in the inventory, the customer's credit will be checked.	$c_2 = \{d_5 = \text{"Yes"}$: *Execute*$(v_4)\}$
If the credit rating of a customer is bad, the order needs to be approved by a sales manager	$c_3 = \{d_7 = \text{"bad"}$: *Execute*$(v_5)\}$
The down payment can be processed only if the credit rating of a customer is good or the order is approved by a sales manager.	$c_4 = \{(d_7 = \text{"good"})$: *Execute*$(v_6)\}$ $c_5 = \{(d_8 = \text{"Yes"})$: *Execute*$(v_6)\}$
If the order is disapproved by the sales manager, the customer will be notified of the sales manager's decision.	$c_6 = \{(d_8 = \text{"No"})$: *Execute*$(v_{12})\}$

Table 2
Data dependencies in the order processing example.

Activity (v)	Unconditional input data (I^u_v)	Conditional input data (I^c_v)	Execution input data (I^e_v)	Output data (O_v)
s	∅	∅	∅	d_0
e	d_1, d_5, d_6	$d_9, d_{10}, d_{11}, d_{12}, d_{13}, d_{14}, d_{15}, d_{16}, d_{17}, d_{18}$	∅	∅
v_1	∅	∅	∅	d_1, d_2, d_3, d_4, d_6
v_2	d_0, d_2, d_4	∅	∅	d_5
v_3	d_1	∅	d_5	d_{11}, d_{12}
v_4	d_3	∅	d_5	d_7
v_5	d_1, d_2, d_3, d_4	∅	d_7	d_8
v_6	d_1, d_2, d_3, d_4, d_6	d_8	d_7, d_8	d_9
v_7	d_2, d_3, d_7, d_9	d_8	∅	d_{13}
v_8	d_1, d_2, d_3	∅	d_5	d_{15}
v_9	d_0, d_4, d_{14}	∅	∅	d_{10}
v_{10}	$d_1, d_2, d_3, d_4, d_{13}$	∅	∅	d_{14}
v_{11}	$d_1, d_2, d_3, d_4, d_6, d_7, d_{13}, d_{14}$	d_8	∅	d_{16}
v_{12}	d_2, d_7, d_8	∅	d_8	d_{18}
v_{13}	$d_1, d_2, d_3, d_{14}, d_{16}$	∅	∅	d_{17}

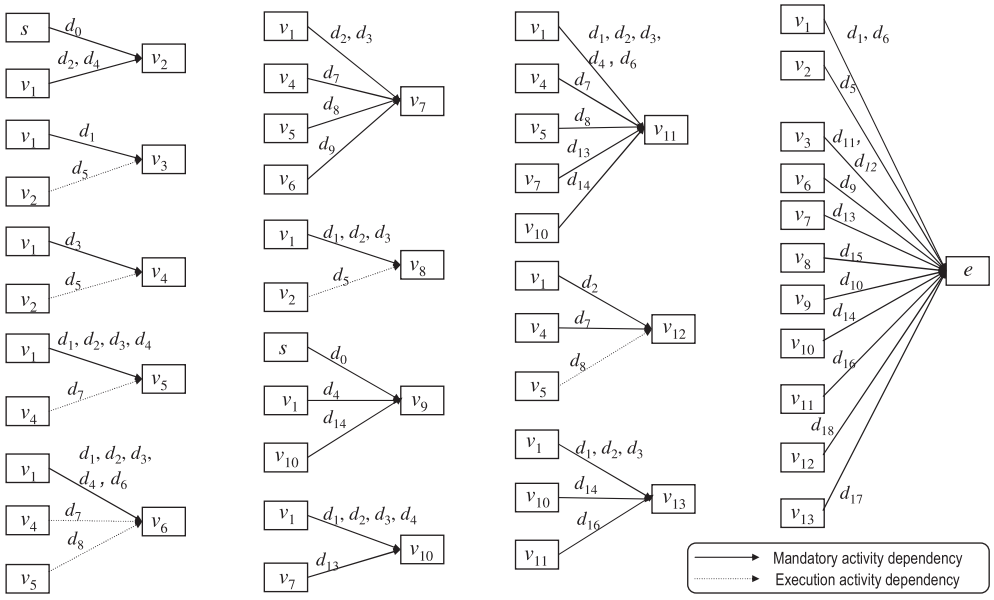


Fig. 2. Direct activity dependencies in the order processing example.

input to V provided by external resources and O_0 is the final output from V. The acyclicity, well-connectedness, completeness, and conciseness of the dataflow are defined as follows.

Definition 7. (Acyclicity of Dataflow) A dataflow is acyclic if $\forall v \in V$, $v \notin \Gamma_v$ holds.

Definition 8. (Well-connectedness of Dataflow) If for $\forall v \in V$ we have $v \in \Gamma_e$, then Λ is well-connected.

Definition 9. (Completeness of Dataflow) Λ is complete if the following two conditions hold: 1) $\forall v \in V$ and $d \in I_v$, there exists $u \in V$ such that $u \in \Delta_v$ and $d \in O_u$; 2) For any Structure $d'\{d_1, \ldots, d_i, d_{i+1}, \ldots, d_n\}$ identified in Λ, $d_y \in I^u{}_v$ always holds if $d_y \in d'$ is a required state before reaching state $d_x \in d'$ and $d_x \in O_v$.

Definition 10. (Conciseness of Dataflow) Λ is concise if the following two conditions hold: 1) $\forall d \in O_v$ where $v \in V$, there exists $u \in V$ such that $d \in I_u$; and 2) $\forall d \in I_v$ where $v \in V$, there exists only one $u \in V$ such that $d \in O_u$.

The acyclicity help avoid loops in a dataflow and the other three properties can prevent the occurrence of dataflow errors [32] and guarantee a dataflow specification that contains sufficient information for proper workflow design. The well-connectedness property proposed in this paper ensures that the activities in consideration all contribute to the production of final output. The completeness property guarantees the source to provide every data item used as an input and thus helps avoid missing data errors [32] in dataflow specifications. Moreover, the completeness property also ensures that an activity producing a new state of a data object always refers to the prior state(s) of the object. The conciseness property checks if all of the output data are used as input by an activity to prevent redundant data errors and if every output data item is produced only once to avoid conflicting errors [32]. The order processing example has been verified to satisfy all the four properties.

Table 3
Full requisite set (Γ_v) in the order processing example.

Activity	Full requisite set (Γ_v)	Activity	Full requisite set (Γ_v)
s	$\varnothing$	v_8	$\{s, v_1, v_2\}$
v_1	$\varnothing$	v_9	$\{s, v_1, v_2, v_4, v_5, v_6, v_7, v_{10}\}$
v_2	$\{s, v_1\}$	v_{10}	$\{s, v_1, v_2, v_4, v_5, v_6, v_7\}$
v_3	$\{s, v_1, v_2\}$	v_{11}	$\{s, v_1, v_2, v_4, v_5, v_6, v_7, v_{10}\}$
v_4	$\{s, v_1, v_2\}$	v_{12}	$\{s, v_1, v_2, v_4, v_5\}$
v_5	$\{s, v_1, v_2, v_4\}$	v_{13}	$\{s, v_1, v_2, v_4, v_5, v_6, v_7, v_{10}, v_{11}\}$
v_6	$\{s, v_1, v_2, v_4, v_5\}$	e	$\{s, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}\}$
v_7	$\{s, v_1, v_2, v_4, v_5, v_6\}$		

4. Derivation of activity relations from data and activity dependencies

This section discusses three types of activity relations between two activities as well as three propositions particularly for deriving those activity relations.

4.1. Workflow modeling and activity relations

The design framework we propose is not constrained to a particular modeling language since the well-known workflow languages such as EPC, BPMN, Petri nets, and YAWL all support modeling the basic workflow structures [4]. We choose UML activity diagram because it has been more widely adopted in the software industry [12,28] than other alternatives. EPC is a company standard developed by SAP. As emerging languages for workflow modeling, BPMN and YAWL are not well adopted yet. Petri nets are used only by a few workflow products, such as Woflan [2]. Consequently, UML activity diagram is a natural choice for us. Given that UML diagrams are not based on any formalism [19], we formally define workflow modeling before introducing three types of activity relations.

Definition 11. (Workflow) A workflow W is a 7-tuple$<A, s, e, R, L, \Lambda, C>$, where

1. A is a finite set of business activities,
2. $s \in A$ is the start activity where $InDegree(s) = 0$ and $OutDegree(s) >= 1$,
3. $e \in A$ is the end activity where $InDegree(e) >= 1$and $OutDegree(e) = 0$,
4. R is a finite set of routing activities and $R = R^{XS} \cup R^{XJ} \cup R^{PS} \cup R^{PJ}$ where R^{XS} is a set of XOR-Splits, R^{XJ} is a set of XOR-Joins, R^{PS} is a set of AND-Splits, and R^{PJ} is a set of AND-Joins,

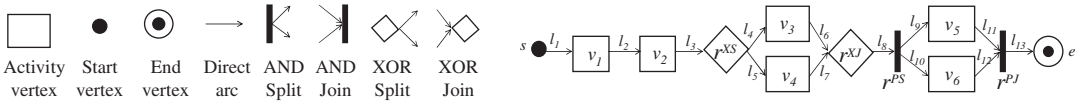


Fig. 3. Activity based workflow modeling in UML activity diagram.

5. $L=\{l_i|i=1,\ldots,n\}$ is a set of directed arcs among activities and each l_i is an ordered pair $<v^t, v^h>$ where $v^t \in A\cup R$ is the tail of l_i, $v^h \in A\cup R$ is the head of l_i, and $v^t \neq v^h$,
6. Λ is the dataflow of A, i.e., $\Lambda=\{\lambda_v\ [I^u_v|I^c_v|I^e_v,\ O_v] \mid v\in A\}$,
7. $C=\{c=f(D):Execute(V_c) \mid D\ \{I^e_v \mid v\in A\}$ and $V_c\ A\}$ is a finite set of routing constraints.

If $InDegree(v_i)=1$ holds for every $v_i\in A|\{s\}$ and $OutDegree(v_i)=1$ for every $v_i\in A|\{e\}$, W is a standardized workflow.

Fig. 3a shows the basic workflow elements in the UML activity diagram notation and Fig. 3b shows a standardized workflow with the business activities v_1, v_2, …, v_6, the routing activities r^{XS}, r^{XJ}, r^{PS}, and r^{PJ}, and the arcs $l_1=<s, v_1>$, $l_2=<v_1, v_2>$, …, $l_{13}=<r^{PJ}, e>$ linking all the activities.

For simplicity, we denote hereafter a workflow model $W=<A, s, e, R, L, \Lambda, C>$ as W. Note that to execute W correctly, the dataflow Λ has to be satisfied, namely each input data item defined in the dataflow must be made available when an activity using it as input is to be executed. Next, we formally define this property of a workflow.

Definition 11A. (Flow-Consistency Property) Given a workflow W with dataflow Λ being acyclic, well-connected, complete, and concise, if each input data item in Λ is produced before it is needed, we say that the control flow of W, as specified with R and L, is consistent with dataflow Λ.

Next, we discuss the semantics of a workflow model.

Definition 12. (Execution Rules) Let W be a workflow model.

1. If $v_i\in R^{PS}$, every v_j directly following v_i can be executed after v_i is executed,
2. If $v_i\in R^{PJ}$, v_i can be executed only after every v_j immediately preceding v_i is executed,
3. If $v_i\in R^{XS}$, only one activity of those directly following v_i can be executed immediately after the execution of v_i when the corresponding routing constraint is satisfied,
4. If $v_i\in R^{XJ}$, v_i can be executed once and only once and the execution of v_i is triggered after any activity of those immediately preceding v_i is executed,
5. If $v_i\in A$ and $InDegree(v_i)=1$, v_i can be executed after the activity immediately preceding v_i is executed, and
6. s is executed unconditionally when the workflow starts.

Intuitively, all the branches starting with the same AND-Split can be executed concurrently and an AND-Join can be executed only after all the incoming branches have been executed. An XOR-split indicates that only one of the following branches can be executed and an XOR-Join can be executed once and only once after any of its incoming branches has been executed. As business activities with multiple incoming or outgoing arcs can appear only in intermediate steps during the design process and a non-standardized workflow is not subject to execution, we do not define execution rules for such business activities.

Deriving a workflow model is a refinement process. In order to systematically determine joins in intermediate steps, we define a new type of graphs called SJ-Graph based on the concept of path borrowed from the theory of digraph [5]. In a workflow W, a sequence of arcs $<v^t_1, v^h_1>\ldots<v^t_p, v^h_p>\ldots<v^t_n, v^h_n>$ is called a path from v_i to v_j if and only if $v^t_1=v_i$, $v^h_n=v_j$ and $v^h_p=v^t_{p+1}$ for $p=1, 2, \ldots, n$-1. The set of all paths from v_i to v_j is denoted by L_{ij} and the set of all nodes covered by these paths is denoted by V_{ij}. Then, SJ-Graph, a type of graph starting with a Split and ending with a node that has multiple incoming links, can be defined as follows.

Definition 13. (SJ-Graph) Given a workflow W, a SJ-Graph, denoted as $SJG(v_i, v_j)$, is a 4-tuple $<v_i, v_j, V_{ij}, L_{ij}>$ where $v_i\in R^{PS}\cup R^{XS}$, $v_j\in A\cup R$ and $InDegree(v_j)>1$, L_{ij} is the set of all paths from v_i to v_j and V_{ij} $A\cup R$ is the set of all nodes covered by L_{ij}. If v_i and v_j match, i.e., either $v_i\in R^{PS}$ and $v_j\in R^{PJ}$ or $v_i\in R^{XS}$ and $v_j\in R^{XJ}$, we say that $SJG(v_i, v_j)$ is a proper SJ-Graph.

Note that a SJ-Graph may contain other SJ-Graphs. Given $SJG(v_i, v_j)=<v_i, v_j, V_{ij}, L_{ij}>$, if we can find another $SJG(v_p, v_q)=<v_p, v_q, V_{pq}, L_{pq}>$ such that $v_p\in V_{ij}$, $v_q\in V_{ij}$, $V_{pq}\subset V_{ij}$, $L_{pq}\subset L_{ij}$, $SJG(v_i, v_j)$ is a complex SJ-Graph. Otherwise, $SJG(v_i, v_j)$ is a simple SJ-Graph. In Fig. 3b, there are three SJ-Graphs, i.e., the proper simple SJ-Graphs $SJG(r^{XS}, r^{XJ})$ and $SJG(r^{PS}, r^{PJ})$ and the complex SJ-Graph $SJG(r^{XS}, r^{PJ})$.

Definition 14. (Activity Relations) Given an acyclic workflow model W and v_i, $v_j\in A\cup R$, three types of activity relations are defined as follows:

1. Immediate precedence (*): $v_i{}^*v_j$ iff $<v_i, v_j>\in L$,
2. Conditional precedence ($>^k_C$): $v_i>^k_C v_j$ iff $v_i{}^*r^{XS}$, $r^{XS}{}^*v_j$, and $r^{XS}\in R^{XS}$, the superscript k is used to specify the unique routing constraint between v_i and v_j, and
3. AND-parallel ($\wedge$): $v_i\wedge v_j$ iff there exists $r^{PS}\in R^{PS}$ such that $<r^{PS}, v_i>\in L$ and $<r^{PS}, v_j>\in L$.

The immediate precedence and the conditional precedence are mutually exclusive. If $v_i{}^*v_j$, v_j is to be executed right after the execution of v_i. If $v_i>^k_C v_j$, v_j may or may not be executed right after the execution of v_i depending on whether C is satisfied or not and thus v_i and v_j are separated by an XOR-Split, indicating that v_j is the start of a conditional routing branch. Relation "$\wedge$" indicates the beginning of parallel branches. If $v_i\wedge v_j$, then v_i and v_j are executed in parallel.[1]

For example, in Fig. 3b, v_2 must be executed right after v_1, i.e., $v_1{}^*v_2$. Further, v_2 is followed by an XOR-split, which immediately precedes v_3 and v_4. According to Definition 14, v_2 conditionally precedes both v_3 and v_4, i.e., $v_2>^k_C v_3$ and $v_2>^l_C v_4$. Given that v_5 and v_6 immediately follow an AND-split, v_5 and v_6 are in AND-parallel with each other, i.e., $v_5\wedge v_6$.

4.2. Identification of sequential execution

This section discusses how to identify immediate precedence relations from data and activity dependencies. The basic idea is straightforward. If activity v_j uses some data d produced by activity v_i, then v_i should be executed before v_j to make d available for v_j to use. Further, if there exists activity v_p such that v_p depends on v_i and v_j depends on v_p, v_p needs to be executed after v_i and before v_j, thus the two activities, v_i and v_j, cannot be in immediate precedence relation.

[1] For simplicity, other possible types of activity relations are omitted since they are not used in our design method.

Note that v_j cannot have execution dependency on v_i in order to place the two activities in immediate precedence relation because v_j may not be executed after the execution of v_i when the execution of v_j is decided by a data item produced by v_i.

Proposition 1. (Immediate Precedence Rule) Let W be a workflow and Λ be acyclic, well-connected, complete, and concise. Given v_i, $v_j \in A$, $v_i {}^{*} v_j$ occurs if 1) $v_i \rightarrow_m v_j$ and 2) we cannot find $v_i \Rightarrow v_j$.

Example 5 (*Determining immediate precedence*): From Fig. 2, we know $v_1 \rightarrow_m v_2$. From Table 3, we cannot find an activity v_x, through which v_1 depends on v_2 indirectly, given $\Gamma_{v1} = \varnothing$ and $\Gamma_{v2} = \{s, v_1\}$. Thus, no activity needs to be placed between v_1 and v_2. By Definition 4, $v_1 \rightarrow_m v_2$ implies that v_2 does not have execution dependency on v_1. Therefore, v_2 can be executed immediately after v_1, i.e., $v_1 {}^{*} v_2$. Further applying Proposition 1 in the order processing example, we have $v_6 {}^{*} v_7$, $v_7 {}^{*} v_{10}$, $v_{10} {}^{*} v_9$, $v_{10} {}^{*} v_{11}$, and $v_{11} {}^{*} v_{13}$.

Proposition 1 describes condition for determining an immediate precedence relation "*" between two activities. If v_i does not depend on any activities and is therefore the leading activity in a workflow, we can always execute v_i right after the start activity s, i.e., $s {}^{*} v_i$. Moreover, if v_i is depended directly by the end activity e and there exists no other indirect dependencies between v_i and e, i.e., we cannot find $v_i \Rightarrow e$, then we have $v_i {}^{*} e$ since no activity needs to be executed between v_i and e. The proofs for Proposition 1 and others propositions discussed later in this paper can be found in Appendix A.

4.3. Identification of conditional routing

Next, we present the principles for deriving the conditional precedence "$>^{k}_{C}$" between two activities, v_i and v_j. If the execution of activity v_j depends on some data d produced by activity v_i, i.e., there exists a routing constraint $c_k = f(D)$:*Execute* (V) such that $d \in D$ and $v_j \in V$, and no other activities must be executed after the execution of v_i and before the execution of v_j when c_k is met, then v_i and v_j should be in conditional precedence relation, i.e., $v_i >^{k}_{C} v_j$. Proposition 2 below describes the conditions for the conditional precedence to occur between two activities.

Proposition 2. (Conditional Precedence Rule) Let W be a workflow and Λ be acyclic, well-connected, complete, and concise, and $V_p = \{v_p | v_p \in \Gamma_{vj}, v_p \notin \Gamma_{vi}$, and $v_i \in \Gamma_{vp}\}$. For any v_i, $v_j \in A$, $v_i >^{k}_{C} v_j$ occurs if there exists $c_k = f(D)$:*Execute* (V) where $d \in O_{vi}$, $d \in D$, and $v_j \in V$ and either (1) $V_p = \varnothing$ or (2) $v_i \rightarrow_e v_p$, ($v_p \rightarrow v_j$ or $v_p \Rightarrow v_j$), and $O_{vp} \cap I^{u}_{vj} = \varnothing$ hold for each $v_p \in V_p$.

Example 6 (*Determining conditional precedence*): In the order processing example, we know the routing constraint $c_3 = \{d_7 =$ "*bad*": *Execute*$(v_5)\}$ (Table 1) and $d_7 \in O_{v4}$ (Table 2), i.e., $v_4 \rightarrow_e v_5$ (Fig. 2). As Table 3 shows, $\Gamma_{v4} = \{s, v_1, v_2\}$ and $\Gamma_{v5} = \{s, v_1, v_2, v_4\}$. Thus we can only find $v_4 \in \Gamma_{v5}$ and $v_4 \notin \Gamma_{v4}$. As such, there is no other activity between v_4 and v_5, i.e., $V_p = \varnothing$ as defined in Proposition 2, and then v_5 can be executed after v_4 when c_3 is satisfied. In the workflow model, there should be an XOR-Split between v_4 and v_5, i.e., $v_4 >^{3}_{c} v_5$. Further, we know $c_4 = \{d_7 =$ "*good*": *Execute*$(v_6)\}$ (Table 1) and $d_7 \in O_{v4}$ (Table 2), i.e., $v_4 \rightarrow_e v_6$ (Fig. 2). Given $\Gamma_{v4} = \{s, v_1, v_2\}$, $\Gamma_{v5} = \{s, v_1, v_2, v_4\}$, and $\Gamma_{v6} = \{s, v_1, v_2, v_4, v_5\}$ (Table 3), we have $v_5 \in \Gamma_{v6}$, $v_5 \notin \Gamma_{v4}$, and $v_4 \in \Gamma_{v5}$. Given $v_4 \rightarrow_e v_5$ and $v_5 \rightarrow v_6$ (Fig. 2), but $O_{v5} \cap I^{u}_{v6} = \varnothing$ (Table 2), v_5 needs to be executed before v_6 only conditionally. Therefore, no activity must be executed between v_4 and v_6 and v_6 can be executed right after v_4 when c_4 is satisfied. In the workflow model, there should be an XOR-Split between v_4 and v_6, i.e., $v_4 >^{4}_{C} v_6$. Applying Proposition 2 further, six more conditional relations are identified, i.e., $v_2 >^{1}_{c} v_3$, $v_2 >^{1}_{c} v_8$, $v_2 >^{2}_{c} v_4$, $v_5 >^{5}_{c} v_6$, and $v_5 >^{6}_{c} v_{12}$.

4.4. Identification of parallel routing

To determine the start of parallel branches, we need to identify the activities with no dependency on each other, which can be executed simultaneously. Proposition 3 is developed for this purpose.

Proposition 3. (AND-parallel Rule) Let W be a workflow and Λ be acyclic, well-connected, complete, and concise. Given two activities v_i, $v_j \in A$ $v_i \notin \Gamma_{vj}$, and $v_j \notin \Gamma_{vi}$, an AND-parallel relation $v_i \wedge v_j$ occurs if either (1) there exists $v_x \in A$ such that $v_x {}^{*} v_i$ and $v_x {}^{*} v_j$ or (2) there exists $v_x \in A$ such that $v_x >^{k}_{c} v_i$ and $v_x >^{h}_{c} v_j$ where $c_k = f_k(D_k)$:*Execute*(V_k), $c_h = f_h(D_h)$:*Execute*(V_h), $v_i \in V_k$, $v_j \in V_h$, and $f_k(D_k)$ is true if and only if $f_h(D_h)$ is true.

Example 7 (*Determining AND-parallel*): From Example 5, we $v_{10} {}^{*} v_9$ and $v_{10} {}^{*} v_{11}$, i.e., both v_9 and v_{11} can be executed right after v_{10}. Given $v_9 \notin \Gamma_{11}$ and $v_{11} \notin \Gamma_9$ (Table 3), the execution of v_9 does not need to precede that of v_{11} or vice versa. Therefore, there should be an AND-Split between v_{10} and $\{v_9, v_{11}\}$, i.e., $v_9 \wedge v_{11}$. Moreover, we know $v_2 >^{1}_{c} v_3$ and $v_2 >^{1}_{c} v_8$ from Example 6 and $c_1 = \{d_5 =$ "*No*": *Execute*$(v_3, v_8)\}$ from Table 1. Thus, whenever v_3 is executed, v_8 is executed as well. Given $v_3 \notin \Gamma_{v8}$ and $v_8 \notin \Gamma_{v3}$ (Table 3), the execution of v_3 does not need to precede that of v_8 or vice versa. In the workflow model, there should be an AND-Split between v_2 and $\{v_3, v_8\}$, i.e., $v_3 \wedge v_8$.

Proposition 3 says that if we can find activity v_x that can immediately precede two activities v_i and v_j, v_i and v_j can be structured in AND-parallel. Moreover, if we can find activity v_x that conditionally precedes both v_i and v_j and no routing constraints prevent v_i and v_j from being both executed in the workflow, v_i and v_j can be structured in AND-parallel.

4.5. Activity relation matrix

A matrix can be used to store all the activity relations derived from Propositions 1–3. This matrix is referred to as partial relation matrix as formally defined below.

Definition 15. (Partial Relation Matrix) Given $V = \{s, v_1, v_2, v_3, \ldots, v_n, e\}$, a partial relation matrix P is a $(n+1) \times (n+1)$ matrix, where the cell p_{ij} shows the activity relation between v_{i-1} and v_j where $i = 1,2, \ldots, n$, $j = 1,2,\ldots,(n+1)$, $v_0 = s$, and $v_{n+1} = e$. If $p_{ij} =$ "$\wedge$", then $p_{ji} = p_{ij}$. Otherwise, if $p_{ij} =$ "$*$", or "$>^{k}_{C}$", $p_{ji} = null$.

Table 4 shows the partial relation matrix for the order processing example, which contains all activity relations derived from data dependency analysis. When the dataflow of V is well-connected each row and each column in the partial relation matrix contains at least one "$*$" or "$>^{k}_{C}$" because well-connectedness ensures each activity links to the end activity e directly or indirectly by linking to some other activities through data dependencies defined in Section 3.1.

5. Generation of workflow models

In this section, we show how to derive a control flow based on a given set of activity relations. To simplify the design process, we first ignore parallelism without any loss of correctness. Second, we add efficiency by considering the parallelism. Finally, we standardize the workflow by adding Join nodes to the model.

5.1. Generate a correct workflow model without parallelism

A correct workflow model can be derived from two important activity relations, immediate precedence and conditional precedence. To reduce the complexity of workflow design, we propose two special techniques, sequential inline blocks and sequentialization.

Table 4
Partial relation matrix for the order processing example.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	e
s	*													
v_1	NA	*												
v_2		NA	$>^1_c$	$>^2_c$				$>^1_c$						
v_3			NA					∧						*
v_4				NA	$>^3_c$	$>^4_c$								
v_5					NA	$>^5_c$						$>^6_c$		
v_6						NA	*							
v_7							NA			*				
v_8			∧					NA						*
v_9									NA		∧			*
v_{10}									*	NA	*			
v_{11}									∧		NA		*	
v_{12}												NA		*
v_{13}													NA	*

5.1.1. Sequential inline blocks[2]

WfMC [34] defines an inline block as a collection of activities that can be condensed into a block activity. A sequential inline block is a single chain of activities starting with activity v_s and ending with activity v_e. Activities v_s and v_e are called the *source* and *sink* of the block, respectively.

A sequential inline block can be identified as follows. Given that activity v_i immediately precedes activity v_j, v_i and v_j can form a sequential inline block if v_i does not immediately precede any other activities and no other activities immediately precede v_j. Activities v_i and v_j can be business activities or block activities corresponding to sequential inline blocks. The workflow design is simplified as the activities in immediate precedence relation are condensed into a block activity. The activity relations between the block and the activities outside the block are derived from the existing activity relations associated with the source and sink of the given block.

Example 8 (*Inline block analysis*): From Table 4, we know v_1 immediately precedes v_2. Thus, we have a sequential inline block $V^s{}_1 = \{v_1, v_2\}$. We can determine $s^* V^S{}_1$ based on $s^* v_1$ (Table 4) and $V^s{}_1 = \{v_1, v_2\}$. Applying the concept of sequential inline blocks further to Table 4, we find two more blocks $V^s{}_2 = \{v_6, v_7, v_{10}\}$, and $V^s{}_3 = \{v_{11}, v_{13}\}$. Note that v_9 and v_{11} cannot be included in $V^s{}_2$ because v_{10} immediately precedes both v_9 and v_{11} (Table 4). In addition, we have $v_4 >^4_c V^s{}_2$, and $v_9 \wedge V^s{}_3$ based on the activity relations associated with the source and sink activities in the blocks $V^s{}_2$ and $V^s{}_3$.

5.1.2. Sequentialization of AND-parallel relations

To ignore parallelism temporarily, the activities in AND-parallel relation can be sequentialized such that more sequential inline blocks can be identified to further simplify the design process. As illustrated in Fig. 4, without loss of correctness, the AND-parallel relations can be replaced by immediate precedence relations. In this way, v_y, v_{y+1}, v_{y+2}, …, v_{y+n} can be condensed into a sequential inline block.

Example 9 (*Dealing with parallelism*): In the order processing example, given $v_3 \wedge v_8$ (Table 4), we sequentialize v_3 and v_8, thus resulting in another sequential inline block $V^s{}_4 = \{v_3, v_8\}$. Moreover, given $v_9 \wedge V^s{}_3$ from Example 8, we randomly sequence v_9 and $V^s{}_3$ and get another block $V^s{}_5 = \{v_9, V^s{}_3\} = \{v_9, v_{11}, v_{13}\}$. Due to $v_{10} * v_9$ and $v_{10} * v_{11}$ (Table 4), we have $v_{10} * V^s{}_5$. Given $V^s{}_2 = \{v_6, v_7, v_{10}\}$, $V^s{}_2$ and $V^s{}_5$ can form a bigger block $V^s{}_6 = \{V^s{}_2, V^s{}_5\} = \{v_6, v_7, v_{10}, v_9, v_{11}, v_{13}\}$. In total, there are three sequential inline blocks, i.e., $V^s{}_1 = \{v_1, v_2\}$, $V^s{}_4 = \{v_3, v_8\}$, and $V^s{}_6 = \{v_6, v_7, v_{10}, v_9, v_{11}, v_{13}\}$. We derive Table 5 by replacing the activities v_1, v_2, v_3, v_6, v_7, v_8, v_9, v_{10}, v_{11}, and v_{13} in Table 4 with $V^s{}_1$, $V^s{}_4$, and $V^s{}_6$ and then deciding the activity relations of $V^s{}_1$, $V^s{}_4$, and $V^s{}_6$ with other activities. Note that Table 5 is much simpler than Table 4.

[2] Other types of inline blocks are also used when Joins are determined in Sections 5.3.

5.1.3. Determination of XOR-Splits

Now, we can determine where to use XOR-Splits. Proposition 4 is developed for this purpose. Essentially, if activity v_x conditionally precedes a set of activities V and only one activity in V can be chosen for execution after v_x, an XOR-Split is needed between v_x and the set of activities V.

Proposition 4. (XOR-Split rule) Let W be a workflow and Λ be acyclic, well-connected, complete, and concise. Given activity $v_x \in A$ and a set of activities $V \subset A$, if 1) $v_x >^k_c v_i$ always holds for every $v_i \in V$ and 2) the routing constraint $c_i = f_i(D_i)$:*Execute* (V_i) for every $v_i \in V$ is mutually exclusive where $v_i \in V_i$, an XOR-Split, namely r^{XS}, should be placed between v_x and V such that $v_x * r^{XS}$ and $r^{XS} * v_i$ for every $v_i \in V$.

Example 10 (*Determining XOR-Splits*): Applying Proposition 4 to Table 5, we can add three XOR-Split nodes, $r^{XS}{}_1$, $r^{XS}{}_2$, and $r^{XS}{}_3$ as follows:

- $r^{XS}{}_1$ is used between $V^s{}_1$ and $\{v_4, V^s{}_4\}$ because $V^s{}_1 >^2_c v_4$ and $V^s{}_1 >^1_c V^s{}_4$ (Table 5) where $c_1 = \{d_5 =$ "*No*": *Execute*$(v_3, v_8)\}$ and $c_2 = \{d_5 =$ "*Yes*": *Execute*$(v_4)\}$ are mutually exclusive (Table 1);
- $r^{XS}{}_2$ is used between v_4 and $\{v_5, V^S{}_6\}$ because $v_4 >^3_c v_5$ and $v_4 >^4_c V^S{}_6$, where $c_3 = \{d_7 =$ "*bad*": *Execute*$(v_5)\}$ and $c_4 = \{(d_7 =$ "*good*"): *Execute*$(v_6)\}$ are mutually exclusive; and
- $r^{XS}{}_3$ is used between v_5 and $\{v_{12}, V^s{}_6\}$ because $v_5 >^6_c v_{12}$ and $v_5 >^5_c V^s{}_6$ where $c_5 = \{(d_8 =$ "*Yes*"): *Execute*$(v_6)\}$ and $c_6 = \{(d_8 =$ "*No*"): *Execute*$(v_{12})\}$ are mutually exclusive.

Note that if the routing constraints in the above situation are not mutually exclusive, an OR-Choice should be used. As UML activity diagram has no notation for OR-Choice, one possible solution is to use the combination of XOR and AND structures for a given OR-Choice [8]. Table 6 is the result of extending Table 5 with the three XOR-Splits. An intermediate workflow model can be created by graphically illustrating the immediate precedence relations in Table 6, leading to Fig 5. When we expand the workflow model by exploding the three sequential inline blocks $V^S{}_1 = \{v_1, v_2\}$, $V^S{}_4 = \{v_3, v_8\}$, and $V^S{}_6 = \{v_6, v_7, v_{10}, v_9, v_{11}, v_{13}\}$, we get the model in Fig. 6.

5.2. Addition of parallelism to achieve efficiency

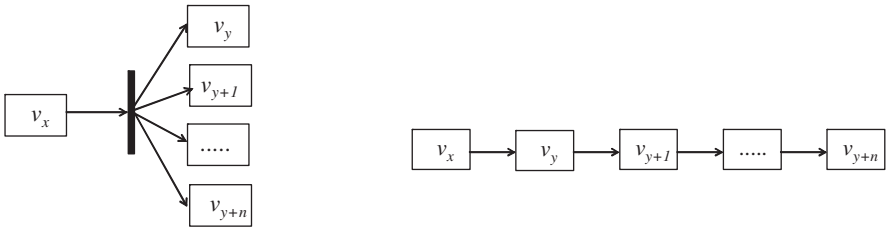
To make the workflow efficient, parallelism ignored when applying sequentialization should now be restored. Proposition 5 is developed for determining AND-Splits based on the AND-parallel relations.

Proposition 5. (AND-Split rule) Let W be a workflow and Λ be acyclic, well-connected, complete, and concise. Given a set of activities $V \subset A$, if $v_i \wedge v_j$ always holds for every pair $(v_i, v_j) \in V$, then an AND-Split r^{PS} should be placed before V such that $r^{PS} * v$ for every $v \in V$.

Example 11 (*Determining AND-Splits*): From Table 4, we know that $v_3 \wedge v_8$ and $v_9 \wedge v_{11}$. That is, v_3 and v_8 should be executed in parallel and it is the same for v_9 and v_{11}. Note that the need for parallelism is confined within two inline blocks, $V^S{}_4 = \{v_3, v_8\}$ and $V^S{}_6 = \{v_6, v_7, v_{10}, v_9, v_{11}, v_{13}\}$. For $V^S{}_4$, we need to place an AND-Split activity between $\{v_3, v_8\}$ by Proposition 5. Given $V^S{}_6 = \{V^s{}_2, V^s{}_5\}$, $V^s{}_2 = \{v_6, v_7, v_{10}\}$, $V^s{}_5 = \{v_9, V^s{}_3\}$, $V^s{}_3 = \{v_{11}, v_{13}\}$, $v_9 \wedge v_{11}$ (Table 4), we conclude $v_9 \wedge V^s{}_3$ because v_{11} is the source of $V^s{}_3$. By Proposition 5, we can place an AND-Split between v_9 and $V^s{}_3$, leading to the workflow model in Fig. 7 that is both correct and efficient. However, the model in Fig. 7 is not yet standardized as some activities, such as v_6, have more than one incoming arc.

5.3. Model standardization by adding AND-Joins and XOR-Joins

To standardize a workflow model, Joins need to be added to ensure that every business activity has one incoming arc and one outgoing arc.



a) Activities in parallel execution b) Activities in sequential execution

Fig. 4. An example of sequentialization.

Table 5
A simplified partial activity relation matrix.

	v_4	v_5	v_{12}	e	V^s_1	V^s_4	V^s_6
s					*		
v_4	NA	$>^3_c$					$>^4_c$
v_5		NA	$>^6_c$				$>^5_c$
v_{12}			NA	*			
V^s_1	$>^2_c$				NA	$>^1_c$	
V^s_4				*		NA	
V^s_6				*			NA

We use the concept of SJ-Graph defined in Section 4 to help determine Joins in an intermediate workflow model. If a simple SJ-Graph $SJG(v_i, v_j)$ starts with a Split activity v_i and ends with a business activity v_j, a Join should be added before v_j.

Proposition 6. (Join rule) If $v_i \in R^{PS} \cup R^{XS}$ and $v_j \in A$ holds for a simple $SJG(v_i, v_j) = <v_i, v_j, V_{ij}, L_{ij}>$ in an acyclic workflow W, a Join activity r is needed between $V = \{v_p | <v_p, v_j> \in L_{ij}\}$ and v_j such that $v_p {}^* r$ for each $v_p \in V$ and $r^* v_j$. To make $SJG(v_i, r)$ a proper SJ-Graph, we require $r \in R^{PJ}$ if $v_i \in R^{PS}$ and $r \in R^{XJ}$ if $v_i \in R^{XS}$.

Example 12 (*Determining Joins in Simple SJ-Graphs*): In Fig. 7, there are three simple SJ-Graphs, $SJG(r^{PS}{}_1, e)$, $SJG(r^{PS}{}_2, e)$, and $SJG(r^{XS}{}_2, v_6)$. To make these SJ-Graphs proper, by Proposition 6, we add AND-Join $r^{PJ}{}_1$ between $\{v_3, v_8\}$ and e, AND-Join $r^{PJ}{}_2$ between $\{v_9, V^S{}_3\}$ and e, and XOR-Join $r^{XJ}{}_1$ before v_6.[3] as shown in Fig. 8.

When $InDegree(v_j) > 1$ and $v_j \in R^{PS} \cup R^{XS}$ in intermediate steps during the design process, in order to apply Proposition 6, we need to add a dummy activity v_{dummy} before v_j such that each $v_p {}^* v_j$ is replaced by $v_p {}^* v_{dummy} {}^* v_j$, where v_p is an activity immediately preceding v_j before adding the dummy activity. In case of a complex SJ-Graph, the concept of inline blocks can be used to reduce the complex SJ-Graph into a simple one. A SJ-Graph $SJG(v_i, v_j)$ can be condensed to a block activity if every path from the outside of $SJG(v_i, v_j)$ to any activities in $SJG(v_i, v_j)$ goes through v_i and every path from any activities in $SJG(v_i, v_j)$ to the outside of $SJG(v_i, v_j)$ goes through v_j. $SJG(v_i, v_j)$ is a parallel inline block if v_i is an AND-Split and v_j is an AND-Join or a conditional routing inline block if v_i is an XOR-Split and v_j is an XOR-Join (WfMC, 1999).

Example 13 (*Handling Complex SJ-Graphs*): In Fig. 8, $SJG(r^{XS}{}_3, e)$ is a complex SJ-Graph, which cannot be subject to Proposition 6 directly. However, after we condense $SJG(r^{PS}{}_2, r^{PJ}{}_2)$ into a block activity, $SJG(r^{XS}{}_3, e)$ becomes a simple SJ-Graph. By Proposition 6, we add XOR-Join $r^{XJ}{}_2$ between $\{v_{12}, r^{PJ}{}_2\}$ and e. Then, we condense $SJG(r^{XS}{}_2, r^{XJ}{}_2)$ and $SJG(r^{PS}{}_1, r^{PJ}{}_1)$ into block activities and obtain a simple SJ-Graph $SJG(r^{XS}{}_1, e)$. By Proposition 6, we add XOR-Join $r^{XJ}{}_3$ between $\{r^{PJ}{}_1, r^{XJ}{}_2\}$ and e, leading to Fig. 9.

5.4. Handling workflow models with overlapping structures

While a simple workflow has clearly identifiable one-to-one correspondence between splits and joins [16,28], some complex workflows may have splits and joins not paired up straightforwardly but overlapped. Next, we show how to handle overlapping structures in which parallel branches are intertwined with conditional routing branches.

Table 7 shows a partial relation matrix with two mutually exclusive routing constraints, c_1 and c_2. For simplicity, we skip Phase 1, i.e., the analysis of data dependencies and activity relations. We show how a workflow model with overlapping structures can be generated from Table 7 step by step in Phase 2.

Step 1 Generate a correct workflow model without parallelism
At this step, we omit parallelism by sequentializing $v_4 \wedge v_5$ and $v_6 \wedge v_7$ into $v_4 {}^* v_5$ and $v_6 {}^* v_7$ and ignoring $v_2 {}^* v_5$ and $v_3 {}^* v_7$, which results in $v_2 {}^* v_4$, $v_4 {}^* v_5$ and $v_3 {}^* v_6$, $v_6 {}^* v_7$, respectively. Now, since we have $v_1 >^1_c v_2$, $v_1 >^2_c v_3$, and c_1 and c_2 mutually exclusive, by Proposition 4, an XOR-Split is placed between v_1 and $\{v_2, v_3\}$ (see Fig. 10a).

Step 2 Add parallelism to achieve efficiency
We now consider parallelism. Given $v_4 \wedge v_5$, $v_2 {}^* v_4$, $v_2 {}^* v_5$ (Table 7), we need an AND-Split between v_2 and $\{v_4, v_5\}$ by Proposition 5. Similarly, $v_6 \wedge v_7$, $v_3 {}^* v_6$, $v_3 {}^* v_7$ in Table 7 indicates an AND-Split is needed between v_3 and $\{v_6, v_7\}$, resulting in Fig. 10b.

Step 3 Standardize the model by adding AND-Joins and XOR-Joins
Now, we add Joins before v_8, v_9, and v_{10} in Fig. 10b, respectively. By Proposition 6, we add AND-Join activity r^{PJ} between $\{v_8, v_9\}$ and v_{10}. Similarly, we add XOR-Join $r^{XJ}{}_1$ between $\{v_4, v_6\}$ and v_8 and XOR-Join $r^{XJ}{}_2$ between $\{v_5, v_7\}$ and v_9. As shown in Fig. 11, we now have four simple proper SJ-Graphs $SJG(r^{XS}, r^{XJ}{}_1)$, $SJG(r^{XS}, r^{XJ}{}_2)$, $SJG(r^{PS}{}_1, r^{PJ})$, and $SJG(r^{PS}{}_2, r^{PJ})$. The model construction is complete for the overlapping workflow.

Table 6
A simplified partial activity relation matrix with XOR-Split activities.

	v_4	v_5	v_{12}	e	V^s_1	V^s_4	V^s_6	$r^{XS}{}_1$	$r^{XS}{}_2$	$r^{XS}{}_3$
s					*					
v_4	NA	$>^3_c$					$>^4_c$		*	
v_5		NA	$>^6_c$				$>^5_c$			*
v_{12}			NA	*						
V^s_1	$>^2_c$				NA	$>^1_c$		*		
V^s_2				*		NA				
V^s_3				*			NA			
r^S_1	*					*				
r^S_2		*					*			
r^S_3			*				*			

[3] Note that this is a special case since there is no activity on the arc from the split node $r^{XS}{}_2$ to v_6.

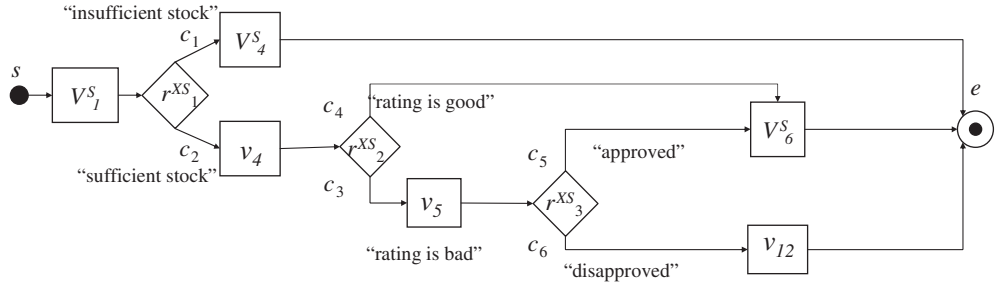


Fig. 5. A workflow model with sequential inline block activities.

5.5. Generality of workflow design analytics

We have shown above how to derive a workflow model from data and activity dependencies. The ingenuity of our approach lies in the idea of breaking down a difficult task through a refining process, which enables us to consider sequential structures, conditional routings, and parallel routings separately. In doing so, we are able to limit the focus first on deciding the application of XOR-Splits and creating a correct model. Then the correct model is refined into an efficient model by adding back the AND-parallel relations previously ignored, and finally Joins are added to finalize the model. Nevertheless, the use of design instruments, such as "activity relation matrices", "graph simplification with inline blocks", and "emphasizing correctness before efficiency", does not alter any critical design decisions when determining activity relations.

Nevertheless, the essence of our approach is to derive a control flow from a dataflow through a two-phase procedure by applying Propositions 1–6 as shown in Fig. 12. Note that Propositions 1 through 3 transform, in Phase 1, a dataflow Λ and the set of associated activities A into a set of activity relations Φ along with a set of intermediate arcs L' consistently, and then in Phase 2, Propositions 4 through 6 derive the set of routing activities R along with the final set of arcs L from the activity relations Φ and the intermediate arc L'. In such a way, the control flow derived is consistent with the dataflow as formally specified in Proposition 7.

Proposition 7. (*Verification of Flow-Consistency*) *Starting with a dataflow Λ and the related set of activities A, application of Propositions 1–6 leads to a control flow, defined with R and L, that satisfies the flow-consistency property defined in Definition 11A.*

Note that since Propositions 3 and 5 help identify all possible parallel branches, a control flow obtained through applying these two propositions are efficient, i.e., no more parallelism can be added without violating any dataflow dependencies.

Next, we discuss that all six propositions cover the necessary instruments needed for the transformation from dataflow to control flow.

1. *Theoretical generality of a naive approach*: Given any well-defined dataflow that is acyclic, well-connected, complete, and concise, applying Proposition 1 alone will derive all linear sequences of activities in the control flow. Of course, the resulting workflow model is not yet fully connected because execution dependencies have not been considered. Next, we assume that all execution activity dependencies are mandatory dependencies; the effect of this naive assumption is for a workflow to do more work than necessary by ignoring routing constraints but will definitely derive all necessary output data. Under this naive assumption, we can apply Proposition 1 further. This will lead to a control flow that satisfies the dataflow requirements although it does so improperly because of (1) wasteful execution of activities due to the ignorance of routing rules and (2) a potential lack of synchronization caused by not using parallelism correctly. This naive approach definitely satisfies any dataflow that is acyclic, well-connected, complete, and concise and is generic since it can derive a naive control flow that satisfies the given dataflow.
2. *Practical generality of our approach*: In our approach, Proposition 2 and Proposition 4 combined together help avoid wasteful execution of paths that need only be executed when a routing condition is satisfied. This makes our approach more efficient than the naive approach but does not reduce its generality because Proposition 2 and Proposition 4 simply help reduce the workload of the naive control flow by applying routing constraints to trigger certain activities only when necessary via XOR-Split. Further, Proposition 3 and Proposition 5 combined together make the workflow more efficient by explicitly applying parallelism via AND-Split without changing the overall workload of the workflow. Proposition 6 simply refines the look of control flow by adding appropriate join activities.

Therefore, our approach is general to help design a workflow model from a data flow that is acyclic, well-connected, complete, and concise.

5.6. How workflow design analytics prevent structural anomalies

We now discuss how the workflow design analytics can help prevent not only dataflow anomalies as discussed in Section 3.4, but also some structural anomalies, thus reducing the burden for verification. We argue about the validity of this claim informally.

1) The property of well-connectedness given in Section 3 ensures that some simple types of structural errors are avoided. For instance, dangling activities and disconnected workflows occur when some

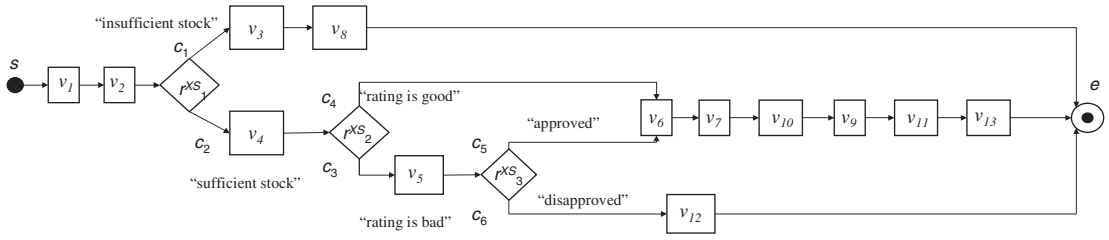


Fig. 6. A workflow model without parallelism.

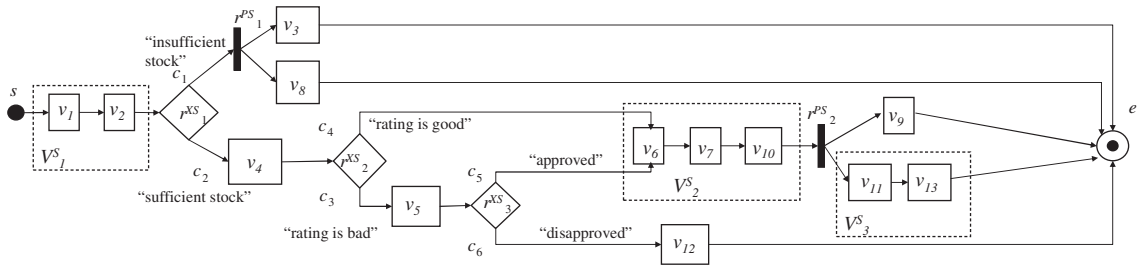


Fig. 7. A workflow model with parallelism.

links in a workflow are broken [7]. Given a well connected dataflow, all activities will be linked to the workflow designed with our approach. Thus, dangling activities and disconnected workflows are avoided.

2) Since a deadlock can occur when an AND-Join is paired with an XOR-Split and Lack-Of-Synchronization (LOS) errors can occur when an XOR-Join is paired with an AND-Split [16], Proposition 6, i.e., the Join rule, help avoid deadlocks and LOS through ensuring that each AND-Join is paired properly with an AND-Split and each XOR-Join is paired properly with an XOR-Split.
3) The application of inline blocks leads to simplification of workflow models, e.g., Fig. 5. This makes it quite easy to spot potential structural errors visually.

Given that dangling activities, deadlocks, and LOS are the most common workflow anomalies [17,20], the ability to avoid such workflow anomalies via our workflow design analytics is a significant feature.

6. Conclusions

Designing workflow models is a critical step in business process management. However, existing workflow design methods remain a largely manual and experiential effort, and the quality of workflow design depends mainly on the skills of the workflow developers, thus resulting in inefficiency in workflow design tasks and potential errors in workflow models. In this paper, we have demonstrated the feasibility of injecting analytics into workflow design by mathematically representing data and activity dependencies. We have shown that workflow design analytics is nontrivial because of the complex interplay of data, activity, and routing constraints. In our research, we succeeded in formulating a workflow model via an two-phase procedure by means of several novel techniques, including the concept of activity relations in workflow modeling, inline blocks analysis for design simplification, and decoupling model correctness from model efficiency when creating the control flow. Our research is significant because it is the first workflow design approach that incorporates a formal procedure starting with dataflow specification. Moreover, certain potential workflow errors can be avoided at the time of workflow design, thus alleviating the burden of workflow verification to a great extent.

Our approach is applicable when a dataflow specification is well-connected, complete, concise, and acyclic. We limit our focus on the five basic workflow constructs, i.e., sequence, XOR-Split, XOR-Join, AND-Split, and AND-Join [36]; other workflow patterns [28] may be added as extension to our research. Thus, we do not claim completeness in this paper. Since advanced workflow constructs such as the OR vertex (either Join or Split) can be simulated by a combination of AND and XOR [7], the design approach we propose is extendable to handle more advanced workflow constructs. In order to design workflows with loops, one approach is to first identify the main path and the feedback path [8]. We are currently in the process of investigating design mechanisms for cyclic workflows with OR-nodes.

It is worth noting that there are a few factors that can constrain the potential benefits of our workflow design approach. First, our approach relies on the quality of dataflow specification. Even though our approach can be used to determine if a set of activities along with their input and output data is sufficient for generating a workflow model, we do not address how to collect dataflow relevant information, how to determine if a dataflow specification truthfully represents the data transformation of a process, and how to deal with incomplete information of activities and dataflow. Therefore, when the dataflow is not well-documented, the usage of our approach may be restricted. Second, when other considerations, such as resource limitations and cost optimization, also play an significantly important role in determining the structure of a workflow model, those considerations may confine the use of the proposed method and require some modification with the workflow model resulting from our approach.

In future research, we intend to extend our work mainly in two directions. First, we will extend our initial research results to more complex situations and develop practical guidelines on workflow design. Second, in order to evaluate the effectiveness of the workflow design analytics, it is important to either conduct laboratory experiments or examine the proposed method though real-world applications.

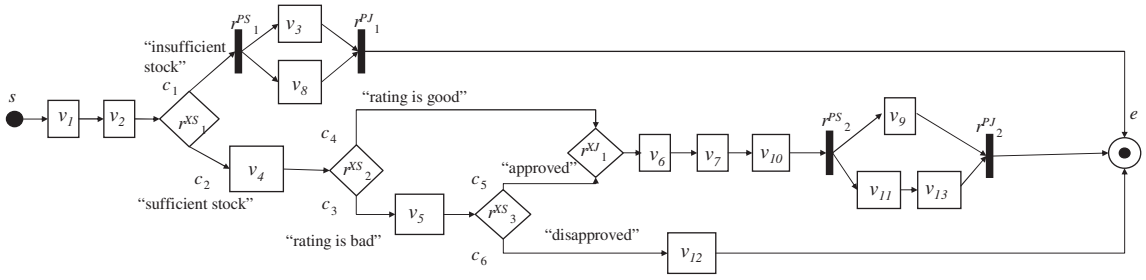


Fig. 8. A workflow model with AND-Joins.

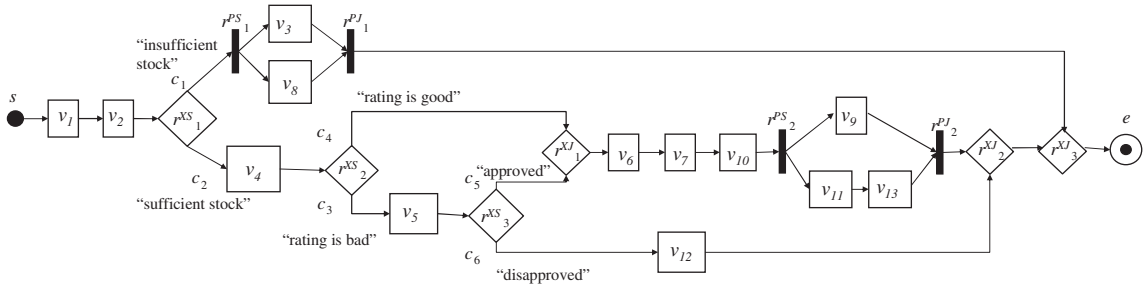


Fig. 9. The final workflow model.

Table 7
The partial relation matrix for a workflow with overlapping structures.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	e
s	*										
v_1	NA	$>^1_c$	$>^2_c$								
v_2		NA		*	*						
v_3			NA			*	*				
v_4				NA	∧			*			
v_5				∧	NA				*		
v_6						NA	∧	*			
v_7						∧	NA		*		
v_8								NA		*	
v_9									NA	*	
v_{10}										NA	*

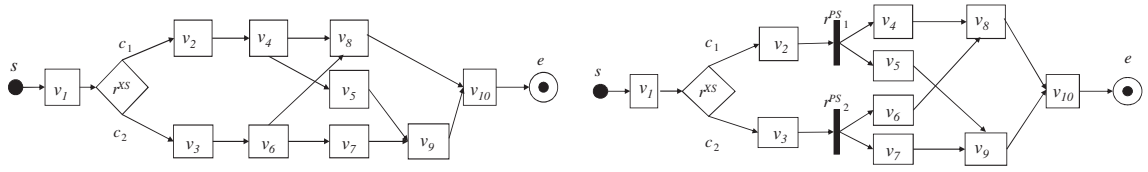
Acknowledgment

The work described in this paper was substantially supported by a strategic research grant (Project No. 7002628) from City University of Hong Kong and partially supported by a grant from the Research Grants Council of Hong Kong Special Administrative Region, China (Project No. CityU 149511).

Appendix A

- *Proof of Proposition 1.* Given $v_i \rightarrow_m v_j$, by Definition 4, there exists data $d \in O_{vi}$ such that $d \in I^u_{vj}$ or $d \in I^c_{vj}$. Thus, v_i needs be executed before v_j to make d available for v_j by Definition 11A. Since $v_i \Rightarrow v_j$ does not exist, no other activities need to be executed after the execution of v_i and before the execution of v_j. By Definition 4, $v_i \rightarrow_m v_j$ implies that v_j doesn't have execution dependency on v_i. Thus, v_j can be executed whenever v_i has been executed. As such, in the workflow model, $<v_i, v_j> \in L$. We have $v_i * v_j$ by Definition 14. ◆
- *Proof of Proposition 2.* Given $c_k = f(D)$:*Execute* (V) where $d \in O_{vi}$, $d \in D$ and $v_j \in V$, we have $v_i \rightarrow_e v_j$ by Definitions 3 and 4. Thus, v_j may or may not be executed after v_i. Furthermore, given either $V_p = \varnothing$ or $v_i \rightarrow_e v_p$, $v_p \rightarrow v_j$ or $v_p \Rightarrow v_j$, and $O_{vp} \cap I^u_{vj} = \varnothing$ hold for each $v_p \in V_p$, no activity must be executed after v_i and before v_j. By Definition 12, v_i and v_j are separated by an *XOR-Split*. By Definition 14, $v_i >^k_c v_j$ holds. ◆
- *Proof of Proposition 3.* Given $v_i \notin \Gamma_{vj}$ and $v_j \notin \Gamma_{vi}$, v_i and v_j have no dependency on each other. That is, the execution of v_i is independent of the execution of v_j. Under the condition (1), given $v_x * v_i$ and $v_x * v_j$, both v_i and v_j can be executed right after v_x. By Definition 12, there should be an AND-Split between v_x and $\{v_i, v_j\}$. By Definition 14, we have $v_i \wedge v_j$. Under the condition 2, given that $v_x >^k_c v_i$, $v_x >^h_c v_j$, $c_k = f_k(D_k)$: *Execute*(V_k), $c_h = f_h(D_h)$:*Execute*(V_h), $v_i \in V_k$, $v_j \in V_h$, and $f_k(D_k)$ is true if and only if $f_h(D_h)$ is true, whenever v_i is executed, v_j is executed as well. Given that the execution of v_i is independent of the execution of v_j, an *AND-Split* is needed between v_x and $\{v_i, v_j\}$ by Definition 12. By Definition 14, we have $v_i \wedge v_j$. ◆
- *Proof of Proposition 4.* Since $v_x >^k_c v_i$ always holds for every $v_i \in V$, there exists an XOR-Split between v_x and every v_i. Since the routing constraint c_i for every $v_i \in V$ is mutually exclusive, only one $v_i \in V$ can be executed. By Definition 12, there can be only one XOR-Split, namely r^{XS}, between v_x and V such that $v_x * r^{XS}$ and $r^{XS} * v_i$ for every $v_i \in V$. ◆
- *Proof of Proposition 5.* Since $v_i \wedge v_j$ holds for every v_i, $v_j \in V$, by Definition 12, only one AND-Split is needed, namely r^{PS}, such that $r^{PS} * v_i$ for every $v_i \in V$. ◆
- *Proof of Proposition 6.* Given $v_i \in R^{PS} \cup R^{XS}$ and $v_j \in A$ holds for a simple $SJG(v_i, v_j)$, W is not a standardized workflow according to Definition 11. A Join r is needed before v_j to standardize W. Given $r \in R^{PJ}$ if $v_i \in R^{PS}$ and $r \in R^{XJ}$ if $v_i \in R^{XS}$, $SJG(v_i, r)$ is a proper SJ-Graph by Definition 13. Thus, Proposition 6 holds. ◆
- *Proof of Proposition 7.* Referring to Fig. 12, we prove this proposition in two steps.

First, we prove via contradiction that the set of activity relations Φ and the associated intermediate links L', which are derived from Λ by applying Propositions 1–3, are consistent with dataflow Λ. Assuming that Φ is not consistent with Λ, then there must be at least one such activity relation $f \in \Phi$ that causes a dataflow breakdown so that a data is not available when it is needed. Thus, f cannot be derived from Propositions 1–3, which follow strictly dataflow Λ. However, Propositions 1 through 3 are the only set of design rules used to



a) The model without AND-Splits **b) The model with AND-Splits**

Fig. 10. Models for the workflow with overlapping structures.

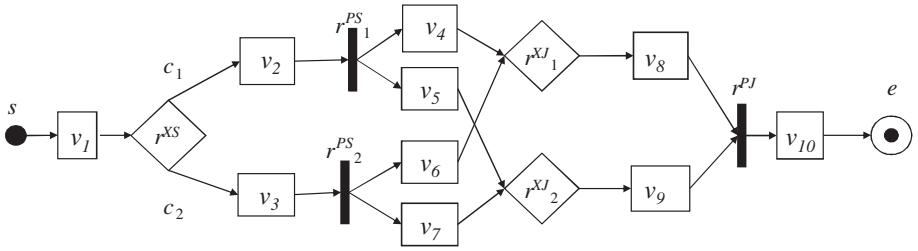


Fig. 11. The final model for the workflow with overlapping structures.

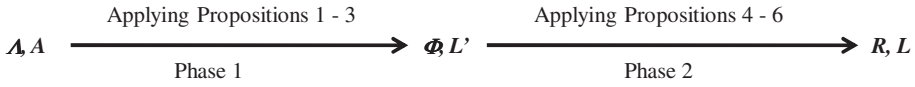


Fig. 12. Illustration of workflow design analytics.

derive Φ. Consequently, Φ and the associated *L'* have to be consistent with *Λ*.

Second, we prove via contradiction that *R* and *L*, which are derived from Φ and *L'* by means of Propositions 4–6, are consistent with *Λ*. Assuming that the control flow defined by *R* and *L* is not consistent with the activity relations Φ and therefore violates the dataflow *Λ*, there must be at least one routing activity $r \in R$ or one arc $l \in L$ that violates an activity relation. However, Propositions 4 through 6 strictly reply on activity relations and they are the only design rules used to derive *R* and *L*. Consequently, *R* and *L* must be consistent with Φ and in turn with *Λ*. As such, Proposition 7 holds. ♦

References

[1] J. Bang-Jensen, G. Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag, London, UK, 2001.
[2] A. Basu, R.W. Blanning, A formal approach to workflow analysis, Information Systems Research 11 (1) (2000) 17–36.
[3] H.H. Bi, J.L. Zhao, Applying propositional logic to workflow verification, Information Technology and Management 5 (3–4) (2004) 293–318.
[4] Y. Choi, J.L. Zhao, Handling cycles in workflow verification by feedback identification and partition, Proc. of 2003 Intl. Conf. on Info. & Know. Engg., June 23-26 2003.
[5] T. Curran, G. Keller, A. Ladd, SAP R/3 Business Blueprint: Understanding the Business Process Reference Model, Prentice Hall PTR, Upper Saddle River, 1998.
[6] E.J. Davidson, Joint application design (JAD) in practice, Journal of Systems and Software 45 (3) (1999) 215–223.
[7] R.M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology (2008), http://dx.doi.org/10.1016/j.infsof.2008.02.006.
[8] M. Dumas, A. ter Hofstede, UML activity diagrams as a workflow specification language, Proc. of the 4th Int. Conf. on the Unified Modeling Language, 2185, 2001, pp. 76–90.
[9] T. Herrmann, T. Walter, The relevance of showcases for the participative improvement of business processes and workflow management, Part. Design Conf, Nov. 12 – 14 1998, pp. 117–127.
[10] K.Y. Kwahk, Y.G. Kim, Supporting business process redesign using cognitive maps, Decision Support Systems 25 (2) (1999) 155–178.
[11] J. Li, H.J. Wang, Z. Zhang, J.L. Zhao, Relation-Centric Task Identification for Policy-Based Process Mining, Proc of the Intl Conf on Information Systems ICIS, Dec. 14-17 2008.
[12] R. Liu, A. Kumar, An Analysis and Taxonomy of Unstructured Workflows, Proceedings of the 3rd Intl Conference on Business Process Management, September 6-8 2005.
[13] S. Lu, A. Bernstein, P. Lewis, Automatic workflow verification and generation, Theoretical Computer Science 353 (1–3) (2006) 71–92.
[14] T.W. Malone, K. Crowston, J. Lee, B. Pentland, Tools for inventing organizations: toward a handbook or organizational processes, Management Science 45 (3) (1999) 425–443.
[15] W. McUmber, B. Cheng, A general framework for formalizing UML with formal languages, Proc. 23rd Internat. Conf. Software Engg, 2001, pp. 433–442.
[16] J. Mendling, Empirical studies in process model verification, Transactions on Petri Nets and Other Models of Concurrency II, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 208–224.
[17] J. Mendling, W.M.P. van der Aalst, Formalization and verification of EPCs with OR-joins based on state and context, Lecture Notes in Computer Science 4495 (2007) 439–453.
[18] J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, G. Neumann, Detection and prediction of errors in EPCs of the SAP reference model, Data & Knowledge Engineering 64 (2008) 312–329.
[19] D. Moody, Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions, Data & Knowledge Engineering 55 (2005) 243–276.
[20] R.M. O'Keefe, S. Lee, Developing a strategy for expert system verification and validation, IEEE Transactions on Systems, Man, and Cybernetics 24 (4) (1994) 643–655.
[21] S. Philippi, H. Hill, Communication support for systems engineering — process modeling and animation with april, Journal of Systems and Software 80 (2007) 1305–1316.
[22] A.D. Preece, S. Talbot, L. Vignollet, Evaluation of verification tools for knowledge-based systems, International Journal of Human Computer Studies 47 (5) (1997) 629–658.
[23] H.A. Reijers, S. Limam, W.M.P. van der Aalst, Product-based workflow design, Journal of Management Information Systems 20 (1) (2003) 229–262.
[24] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, N. Mulyar, Workflow Control-Flow Patterns: A Revised View, BPM Center Report BPM-06-22, 2006, (BPMcenter.org).
[25] S. Sadiq, M. Orlowska, W. Sadiq, C. Foulger, Data flow and validation in workflow modeling, Proc. 15th Australasian Database Conf., Jan. 18–22 2004, pp. 207–214.
[26] S. Sarkar, M. Ramaswamy, Knowledge base decomposition to facilitate verification, Information Systems Research 11 (3) (2000) 261–283.
[27] E.A. Stohr, J.L. Zhao, Workflow automation: overview and research issues, Information Systems Frontiers 3 (3) (2001) 281–296.
[28] S.X. Sun, J.L. Zhao, J. Nunamaker, O.R. Sheng, Formulating the dataflow perspective for business process management, Information Systems Research 17 (4) (2006) 374–391.
[29] N. Trčka, W.M.P. Aalst, van der, N. Sidorova, Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. *Proceedings of CAiSE 2009*, Lecture Notes in Computer Science 5565 (2009) 425–439.
[30] J.D. Ullman, J. Widom, A First Course in Database Systems, Prentice-Hall International, Inc., Upper Saddle River, New Jersey, 1997.
[31] W.M.P. van der Aalst, The application of Petri nets to workflow management, The Journal of Circuits Systems and Computers 8 (1) (1998) 21–66.
[32] W.M.P. van der Aalst, A.H.M. ter Hofstede, YAWL: yet another workflow language, Information Systems 30 (4) (2005) 245–275.
[33] W.M.P. van der Aalst, K. van Hee, Workflow Management: Models, Methods, and Systems, The MIT Press, Cambridge, MA, 2002.
[34] W.M.P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Transactions on Knowledge & Data Engineering 16 (9) (2004) 1128–1142.
[35] I.T.P. Vanderfeesten, H.A. Reijers, W.M.P. Aalst, Product Based Workflow Support: Dynamic Workflow Execution, Proc. 20th Intl. Conf. on Advanced Information Systems Engineering(CAiSE'08), Montpellier, France, June 18-20 2008, pp. 571–574.
[36] WfMC, Workflow Management Coalition Interface 1: Process Definition Interchange Process Model. Document Number WfMC TC-1016-P, 1999.

[37] E.S.K. Yu, J. Mylopoulos, Using Goals, Rules, and Methods to Support Reasoning in Business Process Re-engineering, Proc. of 27th Hawaii Intl. Conf. on Sys. Sci., 1994, pp. 234–243.
[38] E.S.K. Yu, J. Mylopoulos, From E-R to "A-R" — Modeling strategic actor relationships for business process reengineering, International Journal of Cooperative Information Systems 4 (2–3) (1995) 125–144.

Sherry X Sun is an assistant professor at the Department of Information Systems, City University of Hong Kong. Before joining City University of Hong Kong, she worked as a database developer in the Artificial Intelligence Lab at the University of Arizona and a database administrator in Arizona Cancer Center. Dr Sun received the M.S. and Ph.D. degrees in Management Information Systems from Eller College of Management, the University of Arizona, Tucson, Arizona, USA. Her research focuses on the development of technologies for workflow management, service centric computing, and business intelligence. Her research work has been published in renowned journals, including Information Systems Research, IEEE Transactions on Systems, Man and Cybernetics, Information Systems Frontiers, Journal of Systems and Software, and Knowledge Based systems.

J. Leon Zhao is Head and Chair Professor in Information Systems, City University of Hong Kong. He was an Interim Head and an Eller Professor in the Department of Management Information Systems, University of Arizona before moving to Hong Kong in January 2009. He also taught previously at HKUST and College of William and Mary, respectively. He holds Ph.D. and M.S. degrees from the Haas School of Business, UC Berkeley, M.S. degree from UC Davis, and B.S. degree from Beijing Institute of Agricultural Mechanization. His research is on workflow technology and management, with a particular focus on applications of computational intelligence in business process management, business information services, and financial risk management. Leon's research has been supported by NSF (USA), UGC (HK), SAP, IBM, and others. Leon has been an associate editor of ACM Transactions on MIS, Information Systems Research, IEEE Transactions on Services Computing, Decision Support Systems, Electronic Commerce Research and Applications, Information Systems Frontiers, International Journal of Business Process Integration and Management, International Journal of Web and Grid Services, and International Journal of Web Services Research and is on the editorial board of the Journal of Database Management.