The Journal of Systems and Software 82 (2009) 1600–1619

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Detecting artifact anomalies in business process specifications with a formal model

Ching-Huey Wang *, Feng-Jian Wang

Institute of Computer Science and Engineering, College of Computer Science, National Chiao Tung University, Room 510, EC Building, 1001 Ta-Hsueh Road, Hsinchu City, Taiwan, ROC

ARTICLE INFO

Article history:
Available online 5 April 2009

Keywords:
Workflow
Business process
Analysis
Control flow
Data flow
Artifact
Anomaly

ABSTRACT

Many business process analysis models have been proposed, however there are few discussions for artifact usages in workflow specifications. A well-structured business process with sufficient resources might fail or yield unexpected results dynamically due to inaccurate artifact specification, e.g. an inconsistency between artifact and control flow, or contradictions between artifact operations. This paper, based on our previous work, presents a model for describing the input/output of a workflow process and analyzes the artifact usages upon the model. This work identifies and formulates thirteen cases of artifact usage anomalies affecting process execution and categorizes the cases into three types. Moreover, the methods for detecting these anomalies with time complexities $O(n^2)$, less than $O(n^3)$ in previous methods, are presented. Besides, the paper uses an example to demonstrate the processing of them.

1. Introduction

A workflow is seemingly a set of interrelated tasks systematized to achieve certain business goals by accomplishing each task in a particular order under automatic control (The Workflow Management Coalition, 1995). Workflow implementation requires resources for supporting process execution. Resource allocation and resource constraint analysis (Senkul and Toroslu, 2005; Li et al., 2004; Liu et al., 2003; Du and Shan, 1999; Muehlen, 1999) are popular workflow research topics. However, data flow within workflow is seldom addressed (Sadiq et al., 2004; Sun and Zhao, 2004; Sun et al., 2004, 2006).

An artifact is a data instance within a workflow. Introducing artifact usage analysis into (control-oriented) workflow designs can help maintain consistency between execution order and data transition (Sadiq et al., 2004; Sun and Zhao, 2004; Sun et al., 2004, 2006), as well as prevents exceptions due to contradiction between data flow and control flow. In contrast to structural correctness, accuracy in artifact manipulation helps determine whether the execution result of a workflow is meaningful and desirable. Our earlier work (Hsu, 2005; Wang et al., 2006; Hsu and Wang, 2007) introduced the artifact usage analysis into workflow design phase and identified preliminary improper artifact usages affecting workflow execution.

This paper proposes a process model for describing the input/output of business processes and addresses three types of artifact usage anomalies. The model is based on component-based design technique (Zhuge, 2003; Hitomi and Le, 1998) and is compatible with existing control-oriented workflow design models. It provides an easier way to extract knowledge of artifact usages in a workflow. With the model, an analysis procedure of artifact usage is applied before deploying the workflow schema. On the other hand, the checking between data flow and control flow and the information of manipulating artifacts can be applied stepwise along with the specification process. An example demonstrating the contribution of our work and a comparison among related works and ours is also presented.

The remainder of this paper is organized as follows. Section 2 presents the research background and related works. Section 3 presents our process model, including the control flow and artifact flow. There are thirteen cases of artifact usage anomalies identified and then categorized into three types in Section 4. In Section 5, we present a method for detecting each type of anomalies. A comparison between our approach and some related works is given in Section 6. Finally, a conclusion and some recommendations of future work are given in Section 7.

2. Related work and background

2.1. Analysis in workflow specification

A workflow can be deemed as a collection of cooperating and coordinated activities designed to carry out a well-defined complex process, such as a trip planning, conference registration procedure, or business process in an enterprise. In addition, the technology is adopted further in developing service-oriented architecture in the last decade (Yau et al., 2008, 2007). A workflow model describes a workflow in terms of various elements, such as roles and resources, tools and applications, activities, and data, which

* Corresponding author.
E-mail address: chinghui@cs.nctu.edu.tw (C.-H. Wang).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.03.077

represent different perspectives of a workflow (Curtis et al., 1992; Jablonski and Bussler, 1996). Roles and resource elements represent organizational perspective that describes the performers of the tasks instantiated. Tools and application elements represent operational perspectives by specifying what tools and applications are used to execute a particular task. Activity elements are defined with two perspectives: (1) functional: what tasks a workflow performs; and (2) behavioral: when and how tasks are performed. Data elements represent the informational perspective, i.e., what information entities are produced or manipulated in corresponding workflow activities.

A well-defined workflow model leads to efficient development of an effective and reliable workflow application. Correctness issues in a workflow might be classified into three dimensions: control-flow, resource, and data-flow. Generally, the analyses in control-flow dimension are focused on correctness issues of control structure in a workflow. Common control-flow anomalies include deadlock, livelock (infinite loop), lack of synchronization, and dangling reference (Karamanolis et al., 2000a; van der Aalst, 1998a, 1997, 1998b; van der Aalst and ter Hofstede, 2000; van der Aalst et al., 2000a,b, 2003; van der Aalst and Basten, 2002; Verbeek and van der Aalst, 2000; Karamanolis et al., 2000b; Verbeek et al., 2001). For example, a deadlock anomaly occurs if it is no longer possible to make any progress for a workflow instance, e.g. synchronization on two mutually exclusive alternative paths.

Activities belonging to different workflows or parallel activities in the same workflow might access the same resources. A resource conflict occurs when these activities execute over the same time interval. Thus, the analyses in resource dimension include identification of resource conflicts under resource allocation constraints and/or under temporal and/or causality constraints (Senkul and Toroslu, 2005; Li et al., 2004; Liu et al., 2003; Du and Shan, 1999; Muehlen, 1999). On the other hand, missing, redundancy, and conflict use of data are common anomalies in data-flow dimension (Sadiq et al., 2004; Sun and Zhao, 2004; Sun et al., 2004, 2006). A missing data anomaly occurs when an artifact is accessed before it is initialized. A redundant data anomaly occurs when an activity produces an intermediate data output but this data is not required by any succeeding activity. A conflicting data anomaly represents different versions of the same artifact.

Current workflow modeling and analyzing paradigms mainly focus on soundness of control logic, i.e., in the control-flow dimension, including process model analysis (van der Aalst and ter Hofstede, 2000; van der Aalst et al., 2000a,b, 2003; van der Aalst, 1997, 1998b; van der Aalst and Basten, 2002; Verbeek and van der Aalst, 2000; Karamanolis et al., 2000b; Verbeek et al., 2001; Gong and Wang, 2004; Sadiq and Orlowska, 2000), workflow patterns (van der Aalst et al., 2000a,b, 2003; van der Aalst, 1997, 1998b; van der Aalst and Basten, 2002; Verbeek and van der Aalst, 2000; Karamanolis et al., 2000b; Verbeek et al., 2001; Gong and Wang, 2004; Sadiq and Orlowska, 2000, 1997, 1999; Russell et al., 2004) and automatic control of workflow process (Bae et al., 2004). Aalst and ter Hofstede van der Aalst and ter Hofstede (2000) proposed a WorkFlow net (WF-net), based on Petri nets, to model a workflow: transitions representing activities, places representing conditions, tokens representing cases, and directed arcs connecting transitions and places. Son and Kim (2005) defined a well-formed workflow based on closure and control block concepts. He claimed that a well-formed workflow is free from structural errors, and that complex control flows can be made with nested control blocks. Sadiq and Orlowska (2000) proposed a visual verification approach and algorithm with a set of graph reduction rules to discover structural conflicts in process models for given workflow modeling languages.

There are several research topics discussed in resource dimension, including resource allocation constraints (Senkul and Toroslu, 2005; Li et al., 2004), resource availability (Liu et al., 2003), resource management (Du and Shan, 1999) and resource modeling (Muehlen, 1999). Senkul Senkul and Toroslu (2005) developed an architecture to model and schedule workflow with resource allocation constraints and traditional temporal/causality constraints. Li et al. (2004) concluded that a correct workflow specification should have resource consistence and provided a series of detection algorithms. Both Pinar and Hongchen extended workflow specifications with constraint descriptions. Liu et al. (2003) proposed a three-level bottom-up workflow design method to effectively incorporate confirmation and compensation in case of failure.

There was little attention paid to the data-flow dimension, although related analysis in the data-flow dimension is very important since activities cannot be executed properly without sufficient data information. For example, Sadiq et al. (2004) identified and justified the importance of data modeling in the overall workflow design process. In addition, data-flow validation issues and essential requirements of data-flow modeling in workflow specifications are identified. They illustrated and defined seven basic data validation problems: redundant data, lost data, missing data, mismatched data, inconsistent data, misdirected data, and insufficient data. However, Sadiq worked only on the conceptual level and thus, neither concrete data-flow model nor detecting algorithms are proposed. Furthermore, operations on data are only classified into read and write type.

Sun and Zhao (2004), Sun et al. (2004, 2006) formulate the data-flow perspective by means of dependency analysis. The data-flow matrix and an extension of the unified modeling language (UML) activity diagram are proposed to specify the data flow in a business process. Then, three basic types of data-flow anomalies, missing data, redundant data, and conflicting data, were defined. Based on the dependency analysis, algorithms to data-flow analysis for discovering the data-flow anomalies are presented and execute in $O(n^3)$ time. However, as in Sun's work, there is no explicit model proposed to characterize data behaviors. Also, the operation types are considered only with read and initial write.

Our previous work (Hsu and Wang, 2007) was concerned with five different types of operations, read, write, specify, destroy and revise, but the behaviors of an artifact are not explicitly modeled by a finite state machine. The three-layer workflow model is proposed for constructing state transition diagrams of each flow structure to detect inaccurate artifact usage. Detection algorithms are designed based on critical paths for discover the inaccurate state transition of an artifact. However, the time complexity of the algorithms is $O(n^3)$. Table 2.1 summaries the comparisons.

Table 2.1
Summary of comparisons.

	Sadiq et al.	Sun et al.	Our previous approach
Process model	Conceptual level	Data-flow matrices process data diagram	Three-layer workflow model State transition diagram
Operations concerned	Read, write	Read, write	Read, write, specify, destroy, revise
Detecting method	N/A	Data dependency analysis	Artifact usage analysis
Concrete algorithm	N/A	Yes	Yes
Complexity	N/A	$O(n^3)$	$O(n^3)$

2.2. BPEL, BPMN and loop-simplification

BPEL is one of the most popular workflow languages. In BPEL (Alves et al., 2007), the control structures, proposed to indicate the order where the individual activities are executed, include "sequential processing" (Sequence), "repetitive execution" (While and RepeatUtil), "parallel processing" (Flow), "exclusive branches" (If and Pick) and "inclusive branches" (OR and Complex). All types of business processes can be defined by nesting and/or combining the five structures in arbitrary ways (White, 2008).

BPMN (White, 2008) is a solution to represent the business process in a graph. The elements defined in BPMN are classified into *sequence flow* and *object*. A sequence flow links two objects to show the execution order. An object can be an *event*, a *task* or a *gateway*. An event may signal the start (*start event*), the end (*end event*) of a process, a message that arrives or a specific time-date being reached during a process (*intermediate message*/*timer event*). A task is an atomic activity and stands for the work to be performed within a process. A gateway is a routing construct used to control the divergence and convergence of sequence flow. For a set of parallel sequences, a *parallel fork/join gateway* creates/synchronizes concurrent sequence flows. For a set of exclusive sequences, a *data/event-based XOR decision/merge gateway* selects one from/joins a set of mutually exclusive sequence flows. The selection is based on either the process data (data-based) or external event (event-based). Besides, there are two pairwise OR and complex gateways: *OR decision gateway*, *OR merge gateway*, *complex decision gateway* and *complex merge gateway*, their functions can be replaced by combining and/or nesting two or more gateways indicated above.

Ouyang et al. (2006) asserts that the control structures can be represented with BPEL or BPMN in patterns; moreover, the well-structured components representing these patterns with BPMN are proposed. Fig. 2.1 shows examples depicted with BPMN in a well form. In the exclusive branch structures (c) and (d), no matter which succeeding sequence flow is selected (due to the data propagated from its preceding activities in (c) or an event signaled for a notification in (d)), the data propagation mechanisms of both cases are the same in an implicit data flow model (Sadiq et al., 2004). A data exchange between activities is done through global variable(s) stored in a common database. However, the analysis in an iteration is not well concerned.

The set of gateways defined in BPMN is not intended to be minimal (White, 2008). The semantics of one structure might be implemented with other structure(s). To simplify the discussion, this paper minimizes the set of gateways by excluding the ones whose functions are replaceable. In addition, the elements defined in a general model, e.g. BPMN or BPEL, which do not support the analysis of artifact usage, are omitted in our model. For example, the event object of BPMN determining the succeeding flow does not affect the design of data propagation mechanism, it is not concerned here.

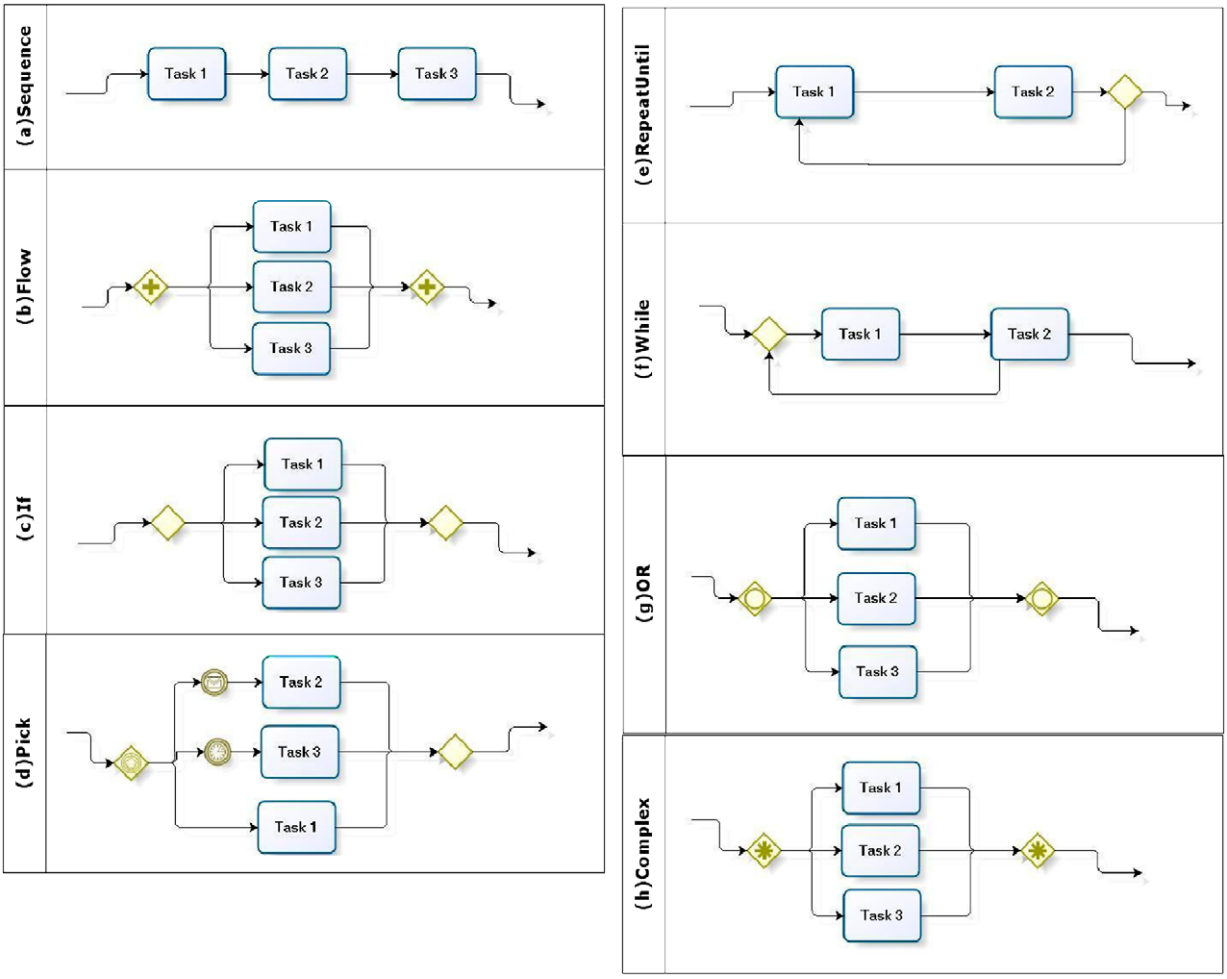

Fig. 2.1. Examples of well-structured control flows.

3. Process modeling

3.1. Basic concept of loop reduction

Our major concern is to find artifact usage anomalies and reduce the computation time. The basic concept of loop reduction is to transform a loop into an XOR structure. The condition(s) of a RepeatUntil structure R is evaluated after each iteration but that of a While structure W is done before each iteration. Obviously, the least time k of RepeatUntil(R)/While(W) execution is 1/0 if the evaluation result in R/W is not concerned. An artifact can be associated with a state representing the latest activity on the artifact (Hsu and Wang, 2007). For iteration times $k > 2$, all the possible state variations of the artifacts operated are the same as those for iterations $k = 2$. Therefore, the state variations in a loop can be grouped according to k. For R, there are two groups of state variations. The first group is for $k = 1$ and the second one is for $k \geqslant 2$. For W, besides above two groups, there is a group for $k = 0$.

Each above group of operation(s) can be translated into a sequence flow (of operations). The R and W can be represented with corresponding XOR gateways and the flows in order to analyze the abnormal behaviors due to these operations. The XOR structures for the loops shown in Fig. 2.1e and f are presented in Fig. 3.1a and b, correspondingly.

3.2. Process specifications

A process consists of a network of activities designed to produce a product or service for a particular customer or market. A process specification, a formalized view of a business process, defines a set of linked (parallel and/or sequential) activities associated with clear defined inputs and outputs respectively. Each activity takes a subset of the process input(s) or the output(s) of its previous activity(ies) and transforms them into data for later use or as process outputs. These data are called artifacts. Thus, a process specification contains not only the control flow but also the artifact flow inside the business process. Here, we give a formal definition of process with these two parts in Definition 3.1.

Definition 3.1. A process specification is a tuple $BP = (G, VT, D, I_W, O_W)$, where

- $G = (V, E)$, representing the control flow, is a directed and acyclic graph, where V is a set of vertices of which each represents an activity and $E \subset V \times V$ is a set of directed edges indicating the precedence relation between two activities.
- $VT : V \rightarrow T$ is a type function that maps each activity into one of the activity types in T, where

$$T = \{Task, SubProcess, ProcessStart, ProcessEnd, AndSplit, AndJoin, XorSplit, XorJoin\}.$$

  The activities whose types are *Task* are called task activities while the others are called control activities.
- D is a set of artifacts used in the process.
- $I_W \subset D$, a subset of D, denotes the set of process inputs.
- $O_W \subset D$, a subset of D, denotes the set of process outputs.

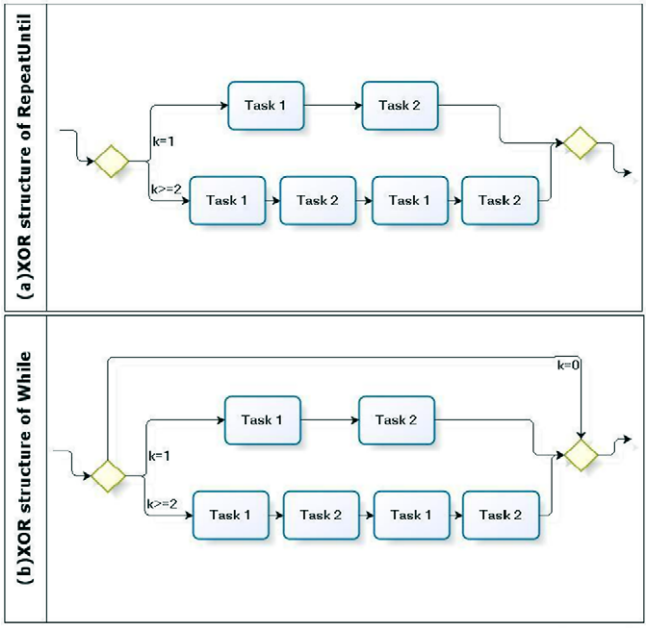

Fig. 3.1. The XOR structures of the loops shown in Fig. 2.1e and f.

3.3. Control flow specification

3.3.1. Activities and control blocks

An activity in a business process might be atomic or non-atomic (compound). An atomic activity is an indecomposable unit of work that is scheduled by a workflow engine. A *sub-process* activity within a process represents a compound activity. The function-based classification of atomic activities divides these activities into two groups: *Task* and *control*. A task activity is defined as making some function progress provided from its associated business process. Three pairs of control activities can be defined to bound a group of activities: (1) ProcessStart and ProcessEnd, (2) AndSplit and AndJoin and (3). XorSplit and XorJoin. Each pair and the activities bounded by them are named as a control block. Notations for activities in BPMN (White, 2008) are partially adopted and shown in Fig. 3.2.

Three control structures "sequential", "parallel branch" and "conditional branch" are constructed from typed activities and their precedence relations, as the figures shown in Fig. 3.3. These structures are used in analyzing the artifact usages. In this paper, the three structures are implemented with *Sequential*, *AND Control* and *XOR Control Block,* respectively. Statements of the four implementations are given below:

- *Sequential block:* Activities within this structure are executed sequentially. Serial activities are fired while completing their preceding activities. Succeeding activities are triggered after their execution.

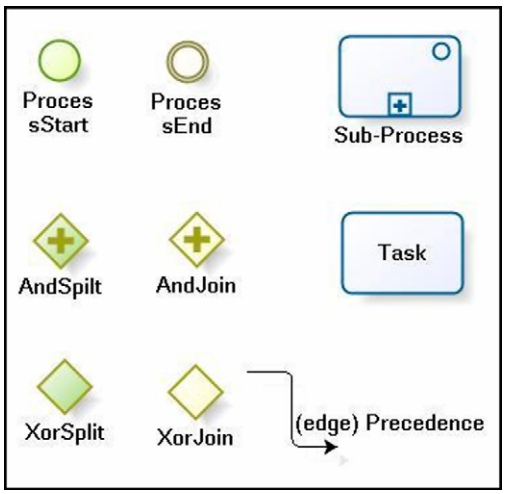

Fig. 3.2. The adopted notations.

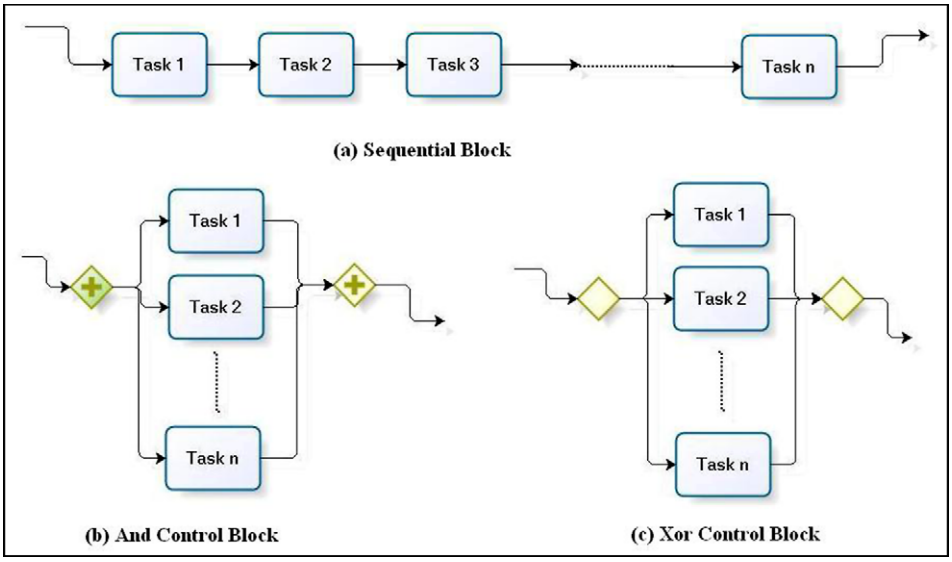

Fig. 3.3. Graphical presentation of the three control structure implementations.

- *AND control block*: An *AndSplit* activity connects with more than one outflow. All outflows of the *AndSplit* activity execute currently. These outflow executions merge synchronously into one on *AndJoin* activity.
- *XOR (eXclusive OR) control block*: An *XorSplit* activity connects with more than one outflow, called branches. The evaluation result of the *XorSplit* activity decides one of the outflows to continue. After execution, these branches converge on an *XorJoin* activity. No synchronization is required since there is only one thread chosen for execution.

The control flow $G = (V,E)$ of a process specification is *well-formed* if the following constraints hold:

- G has a unique vertex v of type *ProcessStart*, which has no incoming edge and one outgoing edge.
  - $\exists! v : VT(v) = ProcessStart \rightarrow InDegree(v) = 0 \wedge OutDegree(v) = 1$.
- G has a unique *ProcessEnd* vertex v of type *ProcessEnd*, which has one incoming edge and no outgoing edge.
  - $\exists! v : VT(v) = ProcessEnd \rightarrow InDegree(v) = 1 \wedge OutDegree(v) = 0$.
- Vertices of type *AndSplit* and *XorSplit* have one incoming edge and more than one outgoing edge.
  - $\forall v : (VT(v) = AndSplit \vee XorSplit) \rightarrow InDegree(v) = 1 \wedge OutDegree(v) > 1$.
- Vertices of type *AndJoin* and *XorJoin* have more than one incoming edge and one outgoing edge.
  - $\forall v \in (VT(v) = AndJoin \vee XorJoin) \rightarrow InDegree(v) > 1 \wedge OutDegree(v) = 1$.
- Any two control blocks can be nested but not overlapped.
  - A typed control block b is denoted with a startVertex and an endVertex. The two vertices corresponds to a pair of control activities defined in Section 3.3.1.

$$\forall b_1 = [v_i, v_j], \quad b_2 = [v_x, v_y],$$
$$b_1 \neq b_2 \rightarrow b_1 \subset b_2 \vee b_2 \subset b_1 \vee b_1 \cap b_2 = \emptyset.$$

3.3.2. Relations among activities and control blocks

In this session, relations among activities and control blocks are identified as follows.

Definition 3.2 (*Paths*). A *path* from v_1 to v_k is a sequence of vertices $\langle v_1, \ldots, v_k \rangle$ in a control graph $G = (V,E)$ such that each node is connected to the next vertex in the sequence i.e., the edges (v_i, v_{i+1}) for $i = 1,2,\ldots,k-1$ are in the edge set E. The *path* from v_1 to v_k is denoted by $Path(v1, vk)$.

Definition 3.3 (*Reachability*). Given two vertices u and v, $IsReachable(u,v)$ is a Boolean function that indicates whether there is a path from u to v.

$$\forall u, v \in V, \quad IsReachable(u, v) = true \leftrightarrow \exists Path(u, v) \vee u = v.$$

Definition 3.4 (*Predecessors and successors*).

$$V_v^{IsPredecessor} = \{u \in V | (u, v) \in E\},$$
$$V_v^{\overline{IsPredecessor}} = \left\{ t \in V | t \in V_v^{IsPredecessor} \vee \left(\exists u \in V_v^{IsPredecessor} : t \in V_u^{\overline{IsPredecessor}} \right) \right\},$$
$$V_v^{IsSuccessor} = \{u \in V | (v, u) \in E\},$$
$$V_v^{\overline{IsSuccessor}} = \left\{ t \in V | t \in V_v^{IsSuccessor} \vee \left(\exists u \in V_v^{IsSuccessor} : t \in V_u^{\overline{IsSuccessor}} \right) \right\},$$

$V_v^{IsPredecessor}$ comprises the set of vertices which are the source of an edge with destination vertex $v \in V$. Each element u in $V_v^{IsPredecessor}$ is called a *direct predecessor* of the vertex and is denoted by $u \rightarrow v$. $V_v^{\overline{IsPredecessor}}$ denotes the transitive closure of $V_v^{IsPredecessor}$. $\forall u \in V_v^{\overline{IsPredecessor}}$, v is reachable from u. Each element u in $V_v^{\overline{IsPredecessor}}$ is called a *predecessor* of v and is denoted by uv. $V_v^{IsSuccessor}$ and its transitive closure $V_v^{\overline{IsSuccessor}}$ are defined similarly.

Definition 3.5 (*Ancestor blocks and level of an activity*). $\forall v \in V$, let $v.PB$ denote the parent control block containing v. $\overline{AncestorBlock}$ comprises the set of all control blocks that contain v

$$\overline{AncestorBlock}(v) = \{b | b = v.PB \vee (b \in \overline{AncestorBlock}(v.PB.startVertex))\}.$$

In addition, the cardinality of $\overline{AncestorBlock}(v)$ identifies the nested level of v

$$Level(v) = \begin{cases} \left| \overline{AncestorBlock}(v) \right| & \text{if } v \in V, \\ \left| \overline{AncestorBlock}(v.StartVertex) \right| & \\ \quad \text{if } v \text{ represents a control block.} & \end{cases}$$

Definition 3.6 (*Common ancestor blocks and nearest common ancestor blocks*). Given a set of vertices, $v_1, \ldots, v_n$, B_i is a *common ancestor block* of $v_1, \ldots, v_n$ if and only if the following holds:

$$Bi \in \bigcap_{i=1}^{n} \overline{AncestorBlock}(v_i), \text{ denoted by } Bi \in CAB(v_1, \ldots, v_n).$$

B_i is the *Nearest common ancestor* of $v_1, \ldots, v_n$ if and only if the following holds:

$$\forall Bj \in CAB(v_1, \ldots, v_n) \wedge Bj \neq Bi : Level(Bj) < Level(Bi), \text{ denoted by } NCAB(v_1, \ldots, v_n) = Bi.$$

Definition 3.7 (*Parallel activities*). Given two vertices, u and v, $IsParallel(u, v)$ is a Boolean function to represent if u and v might be executed in parallel within a workflow instance.

$$IsParallel(u, v) = true \Longleftrightarrow NCAB(u, v).Type = \text{“}AND\text{”} \wedge \neg IsReachable(u, v) \wedge \neg IsReachable(v, u).$$

$IsParallel(u,v) = true$, denoted as $u \oplus v$, indicates that u and v might be executed in parallel and v is called a parallel activity of u.

Definition 3.8 (*Exclusive activities*). Given two vertices, u and v, $IsExclusive(u, v)$ is a Boolean function to represent some XOR characteristics of u and v. Within a workflow instance, if u will not be selected for execution, v is selected for execution and vice versa

$$IsExclusive(u, v) = true \Longleftrightarrow NCAB(u, v).Type = \text{“}XOR\text{”} \wedge \neg IsReachable(u, v) \wedge \neg IsReachable(v, u),$$

$IsExclusive(u, v) = true$, denoted as $u \otimes v$, indicates that at most one of u and v can be selected for execution and v is called an exclusive activity of u.

Definition 3.9 (*Companion activities*). Given two vertices, u and v, $IsCompanion(u, v)$ is a Boolean function which indicates whether both u and v are selected for computation

$$IsCompanion(u, v) = true \wedge Level(u) \neq Level(v) \Longleftrightarrow \forall b \in \overline{AncestorBlock}(u) \cup \overline{AncestorBlock}(v) \setminus CAB(u, v) : b.type = \text{“}AND\text{”},$$

$$IsCompanion(u, v) = true \wedge Level(u) = Level(v) \Longleftrightarrow \forall b \in \{NCAB(u, v)\} : b.type = \text{“}AND\text{”},$$

$IsCompanion(u, v) = true$, denoted as $u \odot v$, indicates that neither or both of them are selected for execution. v is called a *companion* activity of u.

3.4. Artifact flow specification

Currently, as identified in Sadiq et al. (2004), there are three major implementation models for artifact flow: explicit data flow, implicit data flow through control flow, and implicit data flow through a process data store. This paper adopts implicit data flow model through a common process data store. Artifact exchanges between tasks are done through global variables stored in a common database. In a workflow, some activities store their output artifacts in the database, and their following activities may access these artifacts later. The activities in our model are regarded as black boxes, i.e., neither the internal computations nor intermediate execution states are visible for each activity. Thus, the artifact usages of an activity are identified as the inputs/outputs of the activity.

3.4.1. Artifacts and artifact operations

Artifacts are the information entities involved in a process, including the input data to the process, the intermediate data produced within the process, and the (final) output data from the process. An artifact is an atomic data item (e.g. a number, a character string, or an image) or a collection of atomic data items (e.g. a document). Intuitively, all artifacts participating in a workflow execution must be pre-defined in a process specifications. Each artifact contains a set of legal operations for its internal data. An activity designed to manipulate a certain artifact can work only with the legal operation(s) for the artifact. From the data storage point of view, each artifact operation can be regarded as one of the following operations, regardless of its semantic meaning:

- *Initialize*: all definition operations, e.g. “fill in”, “create”, and “define” operations.
- *Read*: all reference operations, e.g. “use”, “fetch”, “select”, and “retrieve” operations.
- *Update*: all modification operations, e.g. “write”, “change”, and “update” operations.
- *Destroy:* all deletion operations, e.g. “remove”, “erase”, “cancel”, and “discard” operations.

In general, an *Initialize* operation is used to create an artifact instance in a process. *Read* and *Update* operations are then used to access the instance. Finally, a *Destroy* operation is used to delete the artifact instance. *Destroy* operations are applied for temporary artifacts created during the workflow execution, but may not be strict for all artifacts (see Fig. 3.4).

Fig. 3.4 shows the state diagram of an artifact with the above four kinds of operations, “Uninitialized”, “Initialized”, “Updated”, and “Read”. ‘Uninitialized’ represents the initial state of an artifact. “Initialized”, “Updated”, and “Read” represent states after an *Initialize*, *Update*, and *Read* operation is performed respectively. In addition, the artifact state is set to “Uninitialized” after a *Destroy* operation.

3.4.2. Artifact flow and artifact usages

To simplify the discussion of artifact usages, a formal and complete definition of a task/control activity is shown below:

Definition 3.10 (Task/control activities).
An task/control activity is a tuple $v = (AT_v, SC_v, EC_v, RC_v, I_v, O_v, AS_v)$, where

- AT_v represents the type of the activity.
- SC_v, EC_v, and RC_v are the sets of logical expressions which are evaluated by a workflow engine.

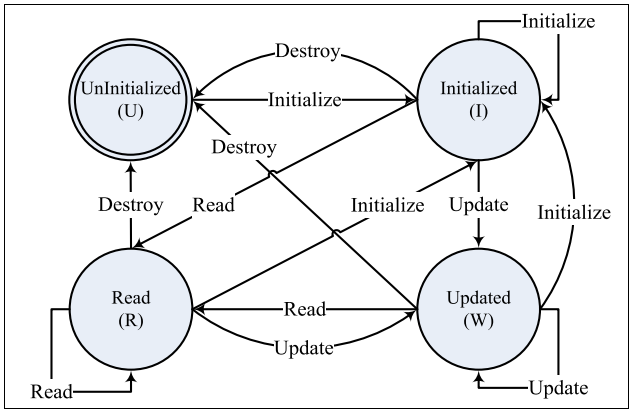

Fig. 3.4. The state diagram of an artifact.

  - SC_v is the set of pre-conditions of which each is evaluated to decide whether an activity within a process instance can be started (only used by task activities).
  - EC_v is the set of post-conditions of which each is evaluated to decide whether an activity within a process instance is completed (only used by task activities).
  - RC_v is the set of routing conditions of which each is evaluated to decide the sequence of activity execution within a process (only used by control activities).
- I_v, the input set, identifies all the artifacts required to be accessed by the activity.
  - For a task activity, I_v contains all the artifacts required for its computation.
  - For a control activity, I_v contains all the artifacts used to evaluate the routing conditions.
- O_v, the output set, identifies all the artifacts produced, updated, or destroyed after executing the activity v. O_v is divided into two disjoint subsets, O_v^+ and O_v^-, where O_v^+ represents the set of the artifacts initialized or updated by v and O_v^- represents the set of the artifacts destroyed by v.
- AS_v is the activity specification (only used by task activities).

Based on Definition 3.10, a usage relation between an activity and an artifact can be defined as follows:

Definition 3.11 (*Consumer, producer, updator, and destroyer activities of an artifact*).
For a given artifact d, the memberships between artifact d and I_v, O_v^+, and O_v^- can be applied for identifying the usage of artifact d at activity v. All the possible usages are categorized as follows:

- if $d \in I_v$ and $\begin{cases} d \notin O_v^+ \\ d \notin O_v^- \end{cases}$, v is called a *Reader* (*Activity*) of artifact d.
- if $d \in I_v$ and $d \in O_v^+$, v is called an *Updator (Activity)* of artifact d.
- if $d \in I_v$ and $d \in O_v^-$, v is called a *Destroyer (Activity)* of artifact d.
- if $d \notin I_v$ and $d \in O_v^-$, v is called a *Illegal Destroyer*[1] (*Activity*) of artifact d.
- if $d \notin I_v$ and $d \in O_v^+$, v is called a *Producer (Activity)* of artifact d.
- if $d \notin I_v$ and $\begin{cases} d \notin O_v^+ \\ d \notin O_v^- \end{cases}$, v is called an *Irrelevantor (Activity)* of artifact d.

In addition, if $d \in Iv$, v is generally called a *Consumer (Activity)* of artifact d and if $d \in O_v^+$,v is generally called a *Writer (Activity)* of artifact d.

Definition 3.12 (*Consumer, updator, destroyer and producer activity sets of an artifact*).

- $V_d^{IsConsumer} = \{v \in V | d \in I_v\}$ is called the *Consumer Activity Set* of artifact d.
- $V_d^{IsUpdator} = \{v \in V | d \in I_v \text{ and } d \in O_v^+\}$ is called the *Updator Activity Set* of artifact d.
- $V_d^{IsDestroyer} = \{v \in V | d \in I_v \text{ and } d \in O_v^-\}$ is called the *Destroyer Activity Set* of artifact d.
- $V_d^{IsProducer} = \{v \in V | d \notin I_v \text{ and } d \in O_v^+\}$ is called the *Producer Activity Set* of artifact d.
- $V_d^{IsReader} = \{v \in V | d \in I_v, d \notin O_v^+ \text{ and } d \notin O_v^-\}$ is called the *Reader Activity Set* of artifact d.

[1] The illegal destroyer is not concerned in our model because the activity destroy artifact arbitrarily. Any useful artifact could be destroyed by the activity during the workflow execution.

4. Artifact usage anomalies

4.1. Artifact usage anomalies

In process specification, the following three types of anomalies might occur: (1) missing production, (2) redundant write, and (3) conflict write. In the subsections, these anomalies are defined and the corresponding usage patterns that cause the anomalies are identified. Every usage pattern is given a name, description, and formulated detection conditions. Table 4.1 shows the symbols used in the usage patterns.

4.2. Missing production anomalies

A *missing* production anomaly occurs when an artifact is consumed before it is produced or after it is destroyed. In order to formulate this type of anomaly, the propagation of an artifact is introduced in Definition 4.1.

Definition 4.1 (*Propagation of artifacts to an activity*). Given an activity v, let a preceding execution order to v denote an execution order leading to v without any parallel activities of v, i.e., only consisting of the predecessors of v. Given an artifact d, if there is at least one preceding execution order to v such that d is produced but not destroyed yet (i.e., d is not in *Uninitialized* state), d can be *propagated* to v. The propagations of artifact d regarding only the preceding execution orders to v are called *preceding propagations* of d to v and can be classified into three cases: (1) *no preceding propagation*, (2) *conditional preceding propagation*, and (3) *unconditional preceding propagation*:

Case (1) indicates that d is always *Uninitialized* for all preceding execution orders to v.
Case (2) indicates whether d is *Uninitialized* depends on the preceding execution orders to v taken.
Case (3) indicates that d is *Initialized* for all preceding execution orders to v.

Let AA_v be the set composed of all artifacts which can be propagated from the predecessors of v. AA_v can be divided into two disjoint subsets, AA_v^u and AA_v^c, where AA_v^u contains artifacts propagated from the predecessors of v unconditionally and AA_v^c contains those propagated from the predecessors of v conditionally.

The causes of missing production anomalies can be discussed as follows. Intuitively, if $v \in V_d^{IsConsumer}$ and $d \notin AA_v$ hold, a missing production anomaly might occur due to *No Preceding Propagation* of d to v. Similarly, if $v \in V_d^{IsConsumer}$ and $d \in AA_v^c$ hold, a missing production anomaly might occur by *Conditional Preceding Propagation* of d to v. Furthermore, considering the parallel activities of v, even

Table 4.1
Symbols used in usage patterns.

W_d: a writer ($d \in O_v^+$)	$\not C_d$: no consumer of d exists
C_d: a consumer ($d \in I_v$)	$\not P_d$: no producer of d exists
U_d: a updator ($d \in I_v$ and $d \in O_v^+$)	$\not R_d$: no reader of d exists
P_d: a producer ($d \notin I_v$ and $d \in O_v^+$)	$\not D_d$: no destroyer of d exists
R_d: a reader $\left(d \in I_v \text{ and } \begin{cases} d \notin O_v^+ \\ d \notin O_v^- \end{cases}\right)$	(): a control block
D_d: a destroyer ($d \in I_v$ and $d \in O_v^-$)	($\otimes$): XOR control block
$\twoheadrightarrow$: reachable link	($\oplus$): AND Control block

though $v \in V_d^{IsConsumer}$ and $d \in AA_v^u$ hold, a missing production anomaly might occur when there is a destroy activity of d in the parallel activities. The execution order of the parallel activities decides the production of d. This kind of anomaly is named *Uncertain Preceding Propagation*.

For each cause of the missing production anomaly, possible usage patterns are characterized by its name, description, and required condition as following:

(1) **No preceding propagation:** $v \in V_d^{IsConsumer} \wedge d \notin AA_v$.

Usage Pattern 1: $\not{P}_d \rightarrow\!\!\!\!\rightarrow C_d \rightarrow\!\!\!\!\rightarrow \not{P}_d$

- **Name**: *No Production*.
- **Description**: Artifact d has at least one consumer activity v; however, no producer activity of d exists in the process.
- **Conditions:** $\exists v \in V_d^{IsConsumer} \wedge V_d^{IsProducer} = \emptyset$.

Usage Pattern 2: $\not{P}_d \rightarrow\!\!\!\!\rightarrow C_d \rightarrow\!\!\!\!\rightarrow P_d \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Delayed Production*.
- **Description**: Artifact d has a consumer activity v which precedes every producer activity of d.
- **Conditions**: $\exists v \in V_d^{IsConsumer} \wedge (V_v^{\overline{IsPredecessor}} \cap V_d^{IsProducer}) = \emptyset \wedge (V_v^{\overline{IsSuccessor}} \cap V_d^{IsProducer}) \neq \emptyset$.

Usage Pattern 3: $\rightarrow\!\!\!\!\rightarrow P_d \rightarrow\!\!\!\!\rightarrow D_d \rightarrow\!\!\!\!\rightarrow C_d \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Early Destruction*.
- **Description**: Artifact d is produced and then destroyed before it is consumed.
- **Conditions**:

$$\exists v \in V_d^{IsConsumer} \wedge d \notin AA_v \wedge \left(V_v^{\overline{IsPredecessor}} \cap V_d^{IsProducer}\right) \neq \emptyset \wedge \left(V_v^{\overline{IsPredecessor}} \cap V_d^{IsDestroyer}\right) \neq \emptyset$$

Usage Pattern 4: $\not{P}_d \rightarrow\!\!\!\!\rightarrow (C_d \otimes P_d) \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Exclusive Production*.
- **Description**: Given two exclusive activities v and u such that v is a consumer of artifact d and u is a producer of d. Due to the characteristic of exclusive activities, only one of v and u might be selected for execution. Although u is a producer of d, it makes no contribution to the propagation of d to v and thus a missing production anomaly occurs.
- **Conditions**: $\exists v \in V_d^{IsConsumer} \wedge d \notin AA_v \wedge (V_v^{IsExclusive} \cap V_d^{IsProducer}) \neq \emptyset$.

Usage Pattern 5: $\not{P}_d \rightarrow\!\!\!\!\rightarrow (C_d \oplus P_d) \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Uncertain Production*.
- **Description**: Given two parallel activities v and u such that v is a consumer of artifact d and u is a producer of d. Due to the race hazard of parallel activities, v might be executed before u. Therefore, u may not make contribution to the propagation of d for v. Consequently, a missing production anomaly occurs if artifact d will not be propagated from the predecessors of v.
- **Conditions**: $\exists v \in V_d^{IsConsumer} \wedge d \notin AA_v \wedge (V_v^{IsParallel} \cap V_d^{IsProducer}) \neq \emptyset$.

(2) **Conditional Preceding Propagation:** $v \in V_d^{IsConsumer} \wedge d \in AA_v^c$. Whether d is propagated depends on what preceding path of v is taken. Consequently, a missing production anomaly occurs when those preceding paths of v in which d is not propagated are taken.

Usage Pattern 6: $\not{P}_d \rightarrow\!\!\!\!\rightarrow (P_d \otimes \not{P}_d) \rightarrow\!\!\!\!\rightarrow C_d \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Conditional Production*.
- **Description**: Artifact d is produced conditionally before a consumer activity of d.
- **Conditions**: $\exists v \in V_d^{IsConsumer} \wedge d \in AA_v^c$.

Usage Pattern 7: $\rightarrow\!\!\!\!\rightarrow P_d \rightarrow\!\!\!\!\rightarrow (D_d \otimes \not{D}_d) \rightarrow\!\!\!\!\rightarrow C_d \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Conditional Destruction*.
- **Description**: Artifact d is destroyed conditionally before a consumer activity of d.
- **Conditions**: $\exists v \in V_d^{IsConsumer} \wedge d \in AA_v^c$.

(3) **Uncertain Preceding Propagation**: $v \in V_d^{IsConsumer} \wedge d \in AA_v^u$.

Usage Pattern 8: $\rightarrow\!\!\!\!\rightarrow P_d \rightarrow\!\!\!\!\rightarrow (D_d \oplus C_d) \rightarrow\!\!\!\!\rightarrow$

- **Name**: *Uncertain Destruction*.
- **Description**: Given two parallel activities v and u such that v is a consumer of artifact d and u is a destroyer of d. Due to the race hazard of parallel activities, v might be executed before u. d is unconditionally propagated from the predecessors of v, but d might be destroyed by u before v is executed. A missing production anomaly occurs.
- **Conditions**: $\exists v \in V_d^{IsConsumer} \wedge d \in AA_v^u \wedge (V_v^{IsParallel} \cap V_d^{IsDestroyer}) \neq \emptyset$.

Theorem 1 (Missing production verification). *A process BP is free from missing production anomalies if the following condition holds:* $\forall v \in V, \forall d \in I_v$: $d \in AA_v^u$ *and* $(V_v^{IsParallel} \cap V_d^{IsDestroyer}) = \emptyset$.

Proof. This theorem is proven by contradiction. Firstly, we assume a missing production anomaly in *BP*. The assumption indicates an activity $v \in V$, an artifact $d \in I_v$, and an execution order Γ such that $v \in \Gamma$ and d is *Uninitialized* when v is selected for execution. However, $d \in AA_v^u$ implies that d is always propagated from the predecessors of v. Furthermore, $(V_v^{IsParallel} \cap V_d^{IsDestroyer}) = \emptyset$ implies that none of parallel activities of v affects the propagation of d to v. Thus, no matter what preceding execution order of v is taken, d is always propagated to v. It is obvious that Γ does not exist. The fact contradicts the assumption given in the beginning and thus Theorem 1 holds. □

4.3. Redundant write anomalies

A redundant write anomaly occurs when an artifact is written (produced or updated) by an activity but the artifact is neither required by the succeeding activities nor a member of the process outputs. Redundancy is not an error; nevertheless, it causes inefficiency. In order to formulate this type of anomalies, the set of artifacts *unused* in the following activities is introduced in Definition 4.2.

Definition 4.2. (*The set of artifacts unused before an activity*). Given an activity v and an artifact d, if there is one preceding execution path to v, where d is written but not consumed, d is *unused* before v. If artifact d is unused for the predecessors of the *Process End* vertex and is not a member of the set of process outputs, a redundant write anomaly occurs. The anomalies can be divided into two classes: *complete* and *conditional*. Artifact d is completely unused indicates that d is unused for all preceding paths of v. Artifact d is conditionally unused indicates that d is unused in certain preceding path(s) of v, i.e., that an anomaly occurs depending on which preceding path of v is taken.

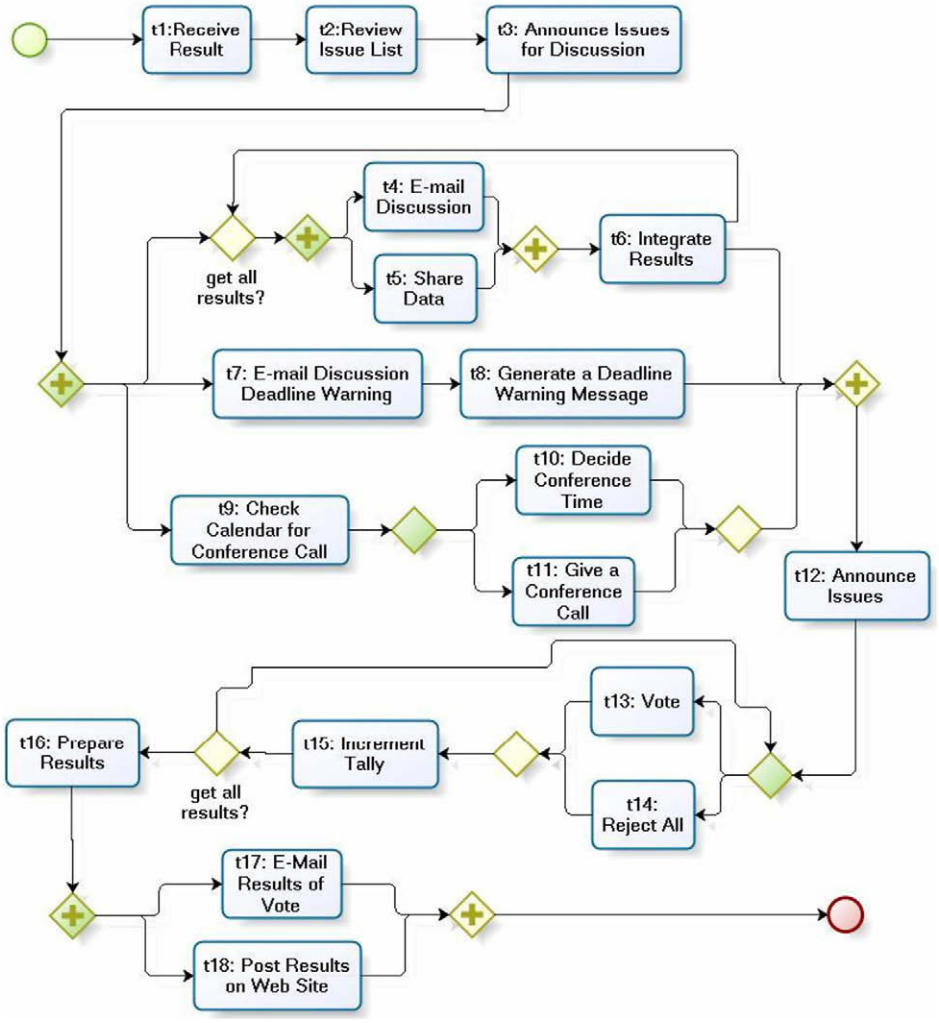

Fig. 5.1. E-mail voting process.

Let NC_v be the set composed of all unused artifacts for the predecessors of v. NC_v can be divided into two disjoint subsets NC_v^u and NC_v^c, where NC_v^u contains completely unused artifacts and NC_v^c contains conditionally unused artifacts.

The causes of redundant write anomalies can be discussed as follows. Intuitively, for every artifact $d \in NC^u_{ProcessEnd}$ and $d \notin O_w$, an *Explicit Redundant Write* anomaly always occurs for artifact d of the process. For every artifact $d \in NC^c_{ProcessEnd}$ and $d \notin O_w$, a *potential redundant write* anomaly might occur for artifact d.

For each category of the redundant write anomaly, the possible usage patterns are characterized by its name, description, and required condition as following:

(1) **Explicit redundant write Usage Pattern 9**:

$$\rightarrow\!\!\!\!\rightarrow W_d \rightarrow\!\!\!\!\rightarrow \cancel{R_d}$$
$$\rightarrow\!\!\!\!\rightarrow W_d \rightarrow\!\!\!\!\rightarrow \cancel{D_d}$$
$$\rightarrow\!\!\!\!\rightarrow (\cancel{W_d} \otimes W_d) \rightarrow\!\!\!\!\rightarrow \cancel{R_d}$$
$$\rightarrow\!\!\!\!\rightarrow (\cancel{W_d} \otimes W_d) \rightarrow\!\!\!\!\rightarrow \cancel{D_d}$$

- **Name**: *No Consumption After Last Write*.
- **Description**: For an artifact d, d does not belong to the process outputs, when d is written by an activity v and when the artifact is unused for all succeeding activities of v, a redundant write occurs for the artifact.
- **Conditions**: $\exists d \in NC^u_{ProcessEnd} : d \notin O_w$.

(2) **Potential redundant write Usage Pattern 10**:

Table 5.1
Artifacts in the e-mail voting process.

Artifacts	
d_1 Issue list	d_9 Integrated results
d_2 Applicant data	d_{10} Data shared
d_3 Discussion participant data	d_{11} Deadline warning message
d_4 Calendar	d_{12} Conference time
d_5 Signature of manager	d_{13} Vote participant data
d_6 Announce issues	d_{14} Vote result
d_7 Comment on announce issues	d_{15} Increment tally
d_8 Discussion result	d_{16} Vote final result
Artifact usages	
R Reader	P Producer
U Updator	D Destroyer

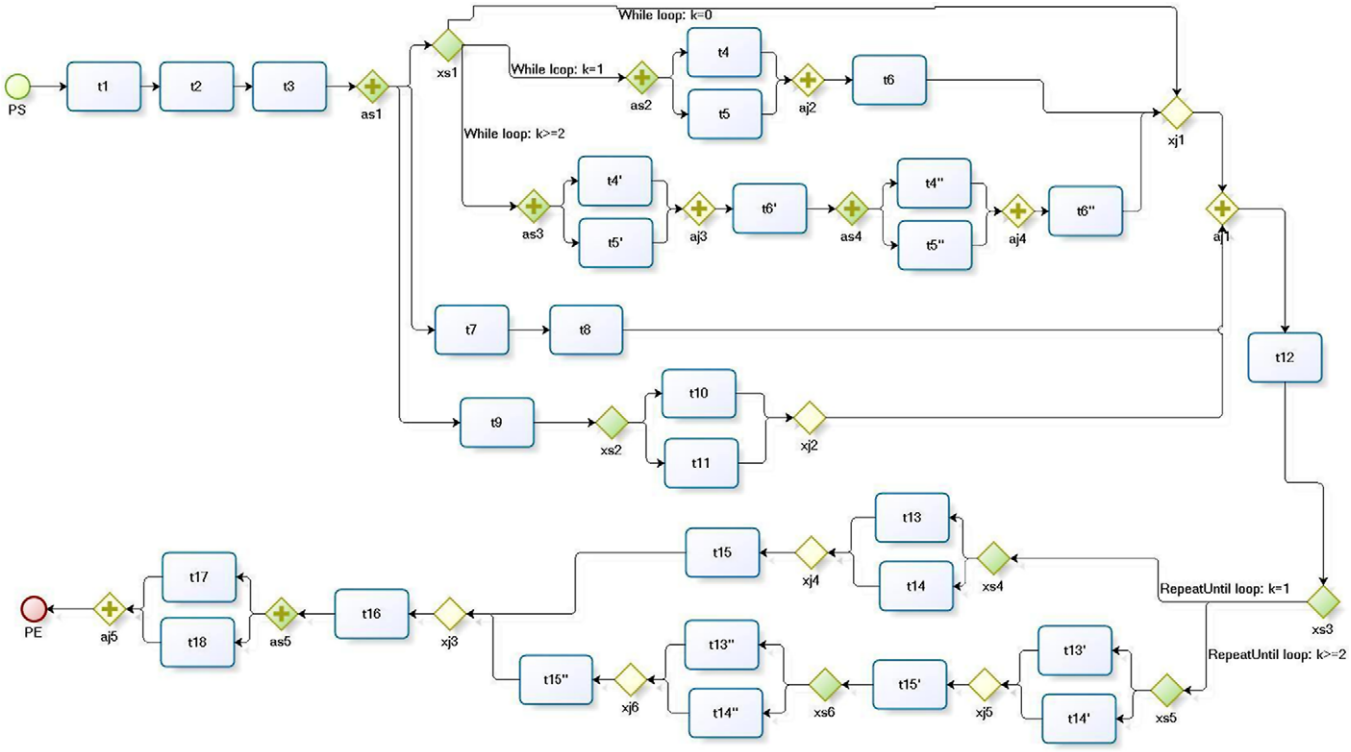

Fig. 5.2. E-mail voting process presented with our process model.

$$\rightarrow\!\!\!\!\!\rightarrow W_d \rightarrow\!\!\!\!\!\rightarrow (R_d \otimes (\cancel{R_\alpha} \wedge \cancel{D_\alpha}))$$
$$\rightarrow\!\!\!\!\!\rightarrow W_d \rightarrow\!\!\!\!\!\rightarrow (D_d \otimes (\cancel{R_\alpha} \wedge \cancel{D_\alpha}))$$
$$\rightarrow\!\!\!\!\!\rightarrow (R_d \oplus W_d) \rightarrow\!\!\!\!\!\rightarrow (\cancel{R_\alpha} \wedge \cancel{D_\alpha})$$
$$\rightarrow\!\!\!\!\!\rightarrow (D_d \oplus W_d) \rightarrow\!\!\!\!\!\rightarrow (\cancel{R_\alpha} \wedge \cancel{D_\alpha})$$

- **Name**: *Conditional Consumption After Last Write*.
- **Description:** For an artifact d, d does not belong to the process outputs. When d is written by an activity v and the artifact is unused for some succeeding activities of v conditionally, a redundant write might occur for the artifact.
- **Conditions:** $\exists d \in NC^{c}_{ProcessEnd} : d \notin O_w$.

Theorem 2 (Redundant write verification). *A process BP is free from the redundant write anomalies if* $NC_{ProcessEnd} \backslash O_w = \emptyset$ *holds.*

Proof. $NC_{ProcessEnd} \backslash O_w = \emptyset$ indicates that artifacts produced in the process either contribute to the process output directly after its last write ($d \in O_w$) or is read/destroyed after its last write on all possible (preceding) execution orders leading to *Process End* ($d \notin NC_{ProcessEnd}$). Therefore, no redundant write anomaly exists if $NC_{ProcessEnd} \backslash O_w = \emptyset$ holds. □

4.4. Conflict writes anomalies

The conflict writes anomalies can be divided into three classes: (1) multiple parallel productions, (2) multiple parallel updates and (3) parallel read and update. An anomaly of multiple parallel productions occurs when more than one activity tries to initialize the same artifact in parallel. An anomaly of multiple parallel updates occurs when more than one activity updates the same artifact in parallel. An anomaly of parallel read and update anomaly occurs when two activities perform read and update on the same artifact concurrently. Each of these anomalies corresponds to the execution order. The anomalies make the artifact version hard to control.

Usage Pattern 11: $\rightarrow\!\!\!\!\!\rightarrow (P_d \oplus P_d) \rightarrow\!\!\!\!\!\rightarrow$
- **Name**: *Multiple Parallel Productions*.
- **Description**: More than one activity initializes the same artifact in parallel.
- **Conditions**: $\exists v \in V_d^{IsProducer} \wedge (V_d^{IsProducer} \cap V_v^{IsParallel}) \neq \emptyset$.

Usage Pattern 12: $\rightarrow\!\!\!\!\!\rightarrow P_d \rightarrow\!\!\!\!\!\rightarrow (U_d \oplus U_d) \rightarrow\!\!\!\!\!\rightarrow$
- **Name**: *Multiple Parallel Updates*.
- **Description**: More than one activity updates the same artifact in parallel.
- **Conditions**:
  $\exists v \in V_d^{IsUpdator} \wedge d \in AA_v \wedge (V_d^{IsUpdator} \cap V_v^{IsParallel}) \neq \emptyset$.

Usage Pattern 13: $\rightarrow\!\!\!\!\!\rightarrow P_d \rightarrow\!\!\!\!\!\rightarrow (R_d \oplus U_d) \rightarrow\!\!\!\!\!\rightarrow$
- **Name**: *Parallel Read and Update*.
- **Description**: Two activities perform read and update respectively on the same artifact concurrently.
- **Conditions**:
  $\exists v \in V_d^{IsReader} \wedge d \in AA_v \wedge (V_d^{IsUpdator} \cap V_v^{IsParallel}) \neq \emptyset$.

Theorem 3 (Conflict writes verification). *A process BP is free from the anomalies of conflict writes if for any two parallel activities v and u,* $(O_v^+ \setminus I_v) \cap (O_u^+ \setminus I_u) = \emptyset$, $(O_v^+ \cap I_v) \cap (O_u^+ \cap I_u) = \emptyset$, $I_v \cap (O_u^+ \cap I_u) = \emptyset$, *and* $I_u \cap (O_v^+ \cap I_v) = \emptyset$ *hold.*

Table 5.2
Artifacts usages in the e-mail voting process.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}		d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}
ps	P	P	P										P				t_{12}						R										
t_1	R	R	R										R				xs_3												R		P	
t_2	R	R	U		P	P	P						U				xs_4															
t_3						R											t_{13}						R			R					P		
as_1																	t_{14}						R			R					P		
xs_1			R														xj_4															
as_2									P								t_{15}														U	
t_4								P									xs_5															
t_5										P							$t_{13'}$						R			R					P		
aj_2																	$t_{14'}$						R			R					P		
t_6								R	U								xj_5															
as_3									P								$t_{15'}$														U	
$t_{4'}$								P									xs_6															
$t_{5'}$										P							$t_{13''}$						R			R					P		
aj_3																	$t_{14''}$						R			R					P		
$t_{6'}$								R	U								xj_6															
as_4									P								$t_{15''}$														U	
$t_{4''}$								P									xj_3												D			
$t_{5''}$										P							t_{16}	R	R	R			R			R				R		R	P
aj_4																	as_5															
$t_{6''}$								R	U								t_{17}													R			R
xj_1			D														t_{18}																R
t_7			R						R		R						aj_5																
t_8											P						pe																
t_9				R																													
xs_2			R																														
t_{10}												P																					
t_{11}			R									R																					
xj_2																																	
aj_1																																	

Table 5.3
Steps to detect missing production anomalies.

ps	$AA^u = \{d_1, d_2, d_3, d_{13}\}$, $AA^c = \emptyset$
t_1	$AA^u = \{d_1, d_2, d_3, d_{13}\}, AA^c = \emptyset, I_{t_1} = \{d_1, d_2, d_3, d_{13}\}, I_{t_1} \setminus AA = \emptyset$
t_2	$AA^u = \{d_1, d_2, d_3, d_{13}, (d_5), (d_6), (d_7)\}, AA^c = \emptyset, I_{t_2} = \{d_1, d_2, d_3, d_{13}\}, I_{t_2} \setminus AA = \emptyset$
t_3	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{t_3} = \{d_6\}, I_{t_3} \setminus AA = \emptyset$
as_1	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{as_1} = \emptyset, I_{as_1} \setminus AA = \emptyset$
xs_1	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{xs_1} = \{d_3\}, I_{xs_1} \setminus AA = \emptyset$
as_2	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_9)\}, AA^c = \emptyset, I_{as_2} = \emptyset, I_{as_2} \setminus AA = \emptyset$
t_4	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_8)\}, AA^c = \emptyset, I_{t_4} = \emptyset, I_{t_4} \setminus AA = \emptyset$
t_5	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_{10})\}, AA^c = \emptyset, I_{t_5} = \emptyset, I_{t_5} \setminus AA = \emptyset$
aj_2	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}, AA^c = \emptyset, I_{aj_2} = \emptyset, I_{aj_2} \setminus AA = \emptyset$
t_6	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}, AA^c = \emptyset, I_{t_6} = \{d_8, d_9\}, I_{t_6} \setminus AA = \emptyset$
as_3	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_9)\}, AA^c = \emptyset, I_{as_3} = \emptyset, I_{as_3} \setminus AA = \emptyset$
$t_{4'}$	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_8)\}, AA^c = \emptyset, I_{t_{4'}} = \emptyset, I_{t_{4'}} \setminus AA = \emptyset$
$t_{5'}$	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_{10})\}, AA^c = \emptyset, I_{t_{5'}} = \emptyset, I_{t_{5'}} \setminus AA = \emptyset$
aj_3	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}, AA^c = \emptyset, I_{aj_3} = \emptyset, I_{aj_3} \setminus AA = \emptyset$
$t_{6'}$	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}, AA^c = \emptyset, I_{t_{6'}} = \{d_8, d_9\}, I_{t_{5'}} \setminus AA = \emptyset$
as_4	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_8, d_{10}, (d_9)\}, AA^c = \emptyset, I_{as_4} = \emptyset, I_{as_4} \setminus AA = \emptyset$
$t_{4''}$	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_{10}, d_9, (d_8)\}, AA^c = \emptyset, I_{t_{4''}} = \emptyset, I_{t_{4''}} \setminus AA = \emptyset$
$t_{5''}$	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_8, d_9, (d_{10})\}, AA^c = \emptyset, I_{t_{5''}} = \emptyset, I_{t_{5''}} \setminus AA = \emptyset$
aj_4	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}, AA^c = \emptyset, I_{aj_4} = \emptyset, I_{aj_4} \setminus AA = \emptyset$
$t_{6''}$	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}, AA^c = \emptyset, I_{t_{6''}} = \{d_8, d_9\}, I_{t_{6''}} \setminus AA = \emptyset$
xj_1	$AA^u = \{d_1, d_2, \cancel{d_3}, d_{13}, d_5, d_6, d_7\}, AA^c = \{d_9, d_8, d_{10}\}, I_{Xj_1} = \{d_3\}, I_{Xj_1} \setminus AA = \emptyset$
t_7	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{t_7} = \{d_3, d_9, d_{11}\}$ $I_{t_7} \setminus AA = \{d_9, d_{11}\} \Rightarrow$ *no* preceding propagation
t_8	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_{11})\}, AA^c = \emptyset, I_{t_8} = \emptyset, I_{t_8} \setminus AA = \emptyset$
t_9	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{t_9} = \{d_4\}$ $I_{t_9} \setminus AA = \{d_4\} \Rightarrow$ *no* preceding propagation
xs_2	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{xs_2} = \{d_3\}, I_{xs_2} \setminus AA = \emptyset$
t_{10}	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_{12})\}, AA^c = \emptyset, I_{t_{10}} = \emptyset, I_{t_{10}} \setminus AA = \emptyset$
t_{11}	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \emptyset, I_{t_{11}} = \{d_3, d_{12}\}$ $I_{t_{11}} \setminus AA = \{d_{12}\} \Rightarrow$ *no* preceding propagation
xj_2	$AA^u = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}, AA^c = \{d_{12}\}, I_{xj_2} = \emptyset, I_{xj_2} \setminus AA = \emptyset$
aj_1	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{aj_1} = \emptyset$ $I_{t_7} \cap O_{xj_1} = \{d_3\} \Rightarrow$ uncertain preceding propagation $I_{xs_2} \cap O_{xj_1} = \{d_3\} \Rightarrow$ uncertain preceding propagation $I_{t_{11}} \cap O_{xj_1} = \{d_3\} \Rightarrow$ uncertain preceding propagation
t_{12}	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{12}} = \{d_6\}, I_{t_{12}} \setminus AA = \emptyset$
xs_3	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, (d_{15})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xs_3} = \{d_{13}\}, I_{xs_3} \setminus AA = \emptyset$
xs_4	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xs_4} = \emptyset, I_{xs_4} \setminus AA = \emptyset$
t_{13}	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, (d_{14})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{13}} = \{d_6, d_9\}$ $I_{t_{13}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
t_{14}	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, (d_{14})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{14}} = \{d_6, d_9\}$ $I_{t_{14}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
xj_4	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xj_4} = \emptyset, I_{xj_4} \setminus AA = \emptyset$
t_{15}	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{15}} = \{d_{15}\}, I_{t_{15}} \setminus AA = \emptyset$
xs_5	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xs_5} = \emptyset, I_{xs_5} \setminus AA = \emptyset$
$t_{13'}$	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, (d_{14})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{13'}} = \{d_6, d_9\}$ $I_{t_{13'}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
$t_{14'}$	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, (d_{14})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{14'}} = \{d_6, d_9\}$ $I_{t_{14'}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
xj_5	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xj_5} = \emptyset, I_{xj_5} \setminus AA = \emptyset$
$t_{15'}$	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{15'}} = \{d_{15}\}, I_{t_{15'}} \setminus AA = \emptyset$
xs_6	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xs_6} = \emptyset, I_{xs_6} \setminus AA = \emptyset$
$t_{13''}$	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, (d_{14})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{13''}} = \{d_6, d_9\}$ $I_{t_{13''}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
$t_{14''}$	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, (d_{14})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{14''}} = \{d_6, d_9\}$ $I_{t_{14''}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
xj_6	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{xj_6} = \emptyset, I_{xj_6} \setminus AA = \emptyset$
$t_{15''}$	$AA^u = \{d_1, d_2, d_{13}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{15''}} = \{d_{15}\}, I_{t_{15''}} \setminus AA = \emptyset$
xj_3	$AA^u = \{d_1, d_2, \cancel{d_{13}}, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{Xj_3} = \{d_{13}\}, I_{Xj_3} \setminus AA = \emptyset$
t_{16}	$AA^u = \{d_1, d_2, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}, (d_{16})\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{16}} = \{d_1, d_2, d_3, d_6, d_9, d_{13}, d_{15}\}$ $I_{t_{16}} \setminus AA = \{d_3, d_{13}\} \Rightarrow$ *no* preceding propagation $I_{t_{16}} \cap AA^c = \{d_9\} \Rightarrow$ conditional preceding propagation
as_5	$AA^u = \{d_1, d_2, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}, d_{16}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{as_5} = \emptyset, I_{as_5} \setminus AA = \emptyset$
t_{17}	$AA^u = \{d_1, d_2, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}, d_{16}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{17}} = \{d_{13}, d_{16}\}$ $I_{t_{17}} \setminus AA = \{d_{13}\} \Rightarrow$ *no* preceding propagation
t_{18}	$AA^u = \{d_1, d_2, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}, d_{16}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{t_{18}} = \{d_{16}\}, I_{t_{18}} \setminus AA = \emptyset$
aj_5	$AA^u = \{d_1, d_2, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}, d_{16}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{aj_5} = \emptyset, I_{aj_5} \setminus AA = \emptyset$
pe	$AA^u = \{d_1, d_2, d_5, d_6, d_7, d_{11}, d_{15}, d_{14}, d_{16}\}, AA^c = \{d_9, d_8, d_{10}, d_{12}\}, I_{pe} = \emptyset, I_{pe} \setminus AA = \emptyset$

Table 5.4
Steps to calculate the unused artifacts.

ps	$NC^u = \{d_1, d_2, d_3, d_{13}\}$, $NC^c = \emptyset$
t_1	$NC^u = \{ \cancel{d_1}, \cancel{d_2}, \cancel{d_3}, \cancel{d_{13}} \}$, $NC^c = \emptyset$
t_2	$NC^u = \{(d_5), (d_6), (d_7), (d_{13})\}$, $NC^c = \emptyset$
t_3	$NC^u = \{d_5, \cancel{d_6}, d_7, d_{13}\}$, $NC^c = \emptyset$
as_1	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
xs_1	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
as_2	$NC^u = \{d_5, d_7, d_{13}, (d_9)\}$, $NC^c = \emptyset$
t_4	$NC^u = \{d_5, d_7, d_{13}, d_9, (d_8)\}$, $NC^c = \emptyset$
t_5	$NC^u = \{d_5, d_7, d_{13}, d_9, (d_{10})\}$, $NC^c = \emptyset$
aj_2	$NC^u = \{d_5, d_7, d_{13}, d_9, d_8, d_{10}\}$, $NC^c = \emptyset$
t_6	$NC^u = \{d_5, d_7, d_{13}, \cancel{d_8}, d_{10}, (d_9)\}$, $NC^c = \emptyset$
as_3	$NC^u = \{d_5, d_7, d_{13}, (d_9)\}$, $NC^c = \emptyset$
$t_{4'}$	$NC^u = \{d_5, d_7, d_{13}, d_9, (d_8)\}$, $NC^c = \emptyset$
$t_{5'}$	$NC^u = \{d_5, d_7, d_{13}, d_9, (d_{10})\}$, $NC^c = \emptyset$
aj_3	$NC^u = \{d_5, d_7, d_{13}, d_9, d_8, d_{10}\}$, $NC^c = \emptyset$
$t_{6'}$	$NC^u = \{d_5, d_7, d_{13}, \cancel{d_8}, d_{10}, (d_9)\}$, $NC^c = \emptyset$
as_4	$NC^u = \{d_5, d_7, d_{13}, d_{10}, (d_9)\}$, $NC^c = \emptyset$
$t_{4''}$	$NC^u = \{d_5, d_7, d_{13}, d_{10}, d_9, (d_8)\}$, $NC^c = \emptyset$
$t_{5''}$	$NC^u = \{d_5, d_7, d_{13}, d_9, (d_{10})\}$, $NC^c = \emptyset$
aj_4	$NC^u = \{d_5, d_7, d_{13}, d_9, d_8, d_{10}\}$, $NC^c = \emptyset$
$t_{6''}$	$NC^u = \{d_5, d_7, d_{13}, \cancel{d_8}, d_{10}, (d_9)\}$, $NC^c = \emptyset$
xj_1	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
t_7	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
t_8	$NC^u = \{d_5, d_7, d_{13}, (d_{11})\}$, $NC^c = \emptyset$
t_9	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
xs_2	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
t_{10}	$NC^u = \{d_5, d_7, d_{13}, (d_{12})\}$, $NC^c = \emptyset$
t_{11}	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \emptyset$
xj_2	$NC^u = \{d_5, d_7, d_{13}\}$, $NC^c = \{d_{12}\}$
aj_1	$NC^u = \{d_5, d_7, d_{13}, d_{11}\}$, $NC^c = \{d_{12}\}$
t_{12}	$NC^u = \{d_5, d_7, d_{13}, d_{11}\}$, $NC^c = \{d_{12}\}$
xs_3	$NC^u = \{d_5, d_7, \cancel{d_{13}}, d_{11}, (d_{15})\}$, $NC^c = \{d_{12}\}$
xs_4	$NC^u = \{d_5, d_7, d_{11}, d_{15}\}$, $NC^c = \{d_{12}\}$
t_{13}	$NC^u = \{d_5, d_7, d_{11}, d_{15}, (d_{14})\}$, $NC^c = \{d_{12}\}$
t_{14}	$NC^u = \{d_5, d_7, d_{11}, d_{15}, (d_{14})\}$, $NC^c = \{d_{12}\}$
xj_4	$NC^u = \{d_5, d_7, d_{11}, d_{15}, d_{14}\}$, $NC^c = \{d_{12}\}$
t_{15}	$NC^u = \{d_5, d_7, d_{11}, d_{14}, (d_{15})\}$, $NC^c = \{d_{12}\}$
xs_5	$NC^u = \{d_5, d_7, d_{11}, d_{15}\}$, $NC^c = \{d_{12}\}$
$t_{13'}$	$NC^u = \{d_5, d_7, d_{11}, d_{15}, (d_{14})\}$, $NC^c = \{d_{12}\}$
$t_{14'}$	$NC^u = \{d_5, d_7, d_{11}, d_{15}, (d_{14})\}$, $NC^c = \{d_{12}\}$
xj_5	$NC^u = \{d_5, d_7, d_{11}, d_{15}, d_{14}\}$, $NC^c = \{d_{12}\}$
$t_{15'}$	$NC^u = \{d_5, d_7, d_{11}, d_{14}, (d_{15})\}$, $NC^c = \{d_{12}\}$
xs_6	$NC^u = \{d_5, d_7, d_{11}, d_{15}, d_{14}\}$, $NC^c = \{d_{12}\}$
$t_{13''}$	$NC^u = \{d_5, d_7, d_{11}, d_{15}, (d_{14})\}$, $NC^c = \{d_{12}\}$
$t_{14''}$	$NC^u = \{d_5, d_7, d_{11}, d_{15}, (d_{14})\}$, $NC^c = \{d_{12}\}$
xj_6	$NC^u = \{d_5, d_7, d_{11}, d_{15}, d_{14}\}$, $NC^c = \{d_{12}\}$
$t_{15''}$	$NC^u = \{d_5, d_7, d_{11}, d_{14}, (d_{15})\}$, $NC^c = \{d_{12}\}$
xj_3	$NC^u = \{d_5, d_7, d_{11}, d_{14}, d_{15}\}$, $NC^c = \{d_{12}\}$
t_{16}	$NC^u = \{d_5, d_7, d_{11}, d_{14}, \cancel{d_{15}}, (d_{16})\}$, $NC^c = \{d_{12}\}$
as_5	$NC^u = \{d_5, d_7, d_{11}, d_{14}, d_{16}\}$, $NC^c = \{d_{12}\}$
t_{17}	$NC^u = \{d_5, d_7, d_{11}, d_{14}, \cancel{d_{16}}\}$, $NC^c = \{d_{12}\}$
t_{18}	$NC^u = \{d_5, d_7, d_{11}, d_{14}, \cancel{d_{16}}\}$, $NC^c = \{d_{12}\}$
aj_5	$NC^u = \{d_5, d_7, d_{11}, d_{14}\}$, $NC^c = \{d_{12}\}$
pe	$NC^u = \{d_5, d_7, d_{11}, d_{14}\}$, $NC^c = \{d_{12}\}$

Proof. If any pair of parallel activities v and u such that $(O_v^+ \setminus I_v) \cap (O_u^+ \setminus I_u) = \emptyset$, no two activities initializes the same artifact in parallel. If $(O_v^+ \cap I_v) \cap (O_u^+ \cap I_u) = \emptyset$, then no two activities updates the same artifact in parallel. Furthermore, $I_v \cap (O_u^+ \cap I_u) = \emptyset$ and $I_u \cap (O_v^+ \cap I_v) = \emptyset$ indicate that no two activities perform read and update respectively on the same artifact. Thus, *BP* is free from conflict writes anomalies. □

5. The methods to detect artifact usage anomalies

The methods for detecting the artifact usage anomalies in a process specification are presented in this section. The goal in our study is to search the artifact usage anomaly only, and it is not necessary to construct the possible artifact activities in each process. Instead, loop can be replaced with a corresponding *XOR* structure. From the top-level view, a well-formed control flow can be deemed as one or a sequence of task(s) and/or top-level control block(s). Thus, an entire flow can be deemed as a sequence of nodes in which each node represents a task or a control block which represents a group of sequential flows. The same perspective can also be applied to the branches of the control block(s). In our approach, a business process is transformed into a sequence of nodes before applying the detection methods. Each method in Section 5.2 is aimed at detecting a type of artifact usage anomalies identified in Section 4.

5.1. Process transformation

Let control flow $G = (V, E)$ of a business process be transformed into a sequence of nodes S. The data structure of S and the nodes within S are defined in Definition 5.1.

Definition 5.1 (*The data structure of a sequence and a node*). S: a structure containing a sequence of nodes (a node could represent a task or a control block)

S.startVertex:	the vertex is the first node of the sequence
S.endVertex:	the vertex is the termination of the sequence
S.nodes:	a set of ordered nodes

node: a node is a structure denoted with type, startVertex, endVertex and subSequences

node.type:	the type of a task or control block
node.startVertex:	the start vertex of a control block
node.endVertex:	the end vertex of a control block
node.subSequences:	a set of sequences attached to the node

The transformation is designed to convert a control flow enclosed by ProcessStart and ProcessEnd vertices. The initial value of the level attribute for each vertex in V is 0. An empty sequence is declared for the control flow in the beginning. During the transformation, a node is created for each task and control block visited. The task node is appended to the sequence directly. Once a split activity s reached, its corresponding control block is identified by function SetLevel which traverses a path with the vertexes enclosed by s and its corresponding join activity j in G. During the traverse, the level attributes of the vertexes traversed are updated. The transformation is applied to the block identified. Such a recursive operation continues until all activities in the block are processed. The corresponding sequence of node(s) generated for each branch of a control block is attached to the node created for their own control blocks when the transformation completes at the end of the branch. The details of transforming a task and a control block are shown in PseudoCode1.

```
PseudoCode 1 TransformControlBlock(G, v, level,
  controlNodes){
// Input: G=(V,E): a directed connected graph
//        v: the traverse is started from vertex v.
//        level: the level of start vertex v
//        controlNodes: a stack containing a set of pairwise control
  activities bounded control blocks
// Output: S: a structure containing a sequence of nodes
    Stack currentControlNodes = new Stack();
    S.startVertex=currentVertex=v;
    while (currentVertex != null) {
        switch (currentVertex.type) {
            case "ProcessStart":
                nextVertex = currentVertex.next;
                break;
            case "Task":
                newNode.type=currentVertex.type;
                newNode.startVertex=currentVertex;
                newNode.endVertex=currentVertex;
                newNode. subSequences.append(null);
                S.nodes.append(newNode);
                nextVertex = currentVertex.next;
                break;
            case "AndSplit" or "XorSplit":
                newNode.type=currentVertex.type;
                newNode.startVertex=currentVertex;
                if (controlNodes.get(currentVertex) == null)
                  currentControlNodes.push(currentVertex.level,
                  currentVertex);
                for each edge (currentVertex, w) ∈ E {
                //recursively transform each branch within a
                control block
                    if (w.level <= currentVertex.level) {
                      controlNodes = SetLevel(w, current
                      Vertex.level++, currentControlNodes,
                      controlNodes);
                      //record pairwise control activities in the
                      branch
                    }
                    endVertex = contrlNodes.get
                    (currentVertex);
                    subsequence = TransformControlBlock(G',
                    currentVertex, currentVertex.level,
                    controlNodes);
                     // the directed connected graph G' of the
                     branch bounded by currentVertex and
                     endVertex
                     subsequence.parentBlock = newNode;
                    //collect every subSequence (corresponding
                    to each branch)
                   newNode.subSequences.append(subSequence);
                }
                // assign "AndJoin" or "XorJoin"to be the end
                vertex of the node
                newNode.endVertex= endVertex;
                S.nodes.append(newNode);
                nextVertex = newNode.endVertex.next;
                break;
             case "EndProcess":
                exit while;
        }
        previousVertex = currentVertex;    //remember last
        traversed vertex
        currentVertex = currentVertex.next;    //continue to
        traverse next node
    }
}
PseudoCode SetLevel(startVertex, level, currentControlNodes,
  controlNodes) {
    n = level;
    currentVertex= startVertex;
    while ((currentVertex.type != "AndJoin" & &
    currentVertex.type != "XorJoin")|| n>=level) {
        if (currentVertex.type== "AndSplit" ||
        currentVertex.type== "XorSplit") {
            currentVertex.level=n;
            currentControlNodes.push(n, currentVertex);
            n ++;
            currentVertex=currentVertex.next;
            // currentVertex.next returns the first node in one
            branch after currentVertex.
        } else if (currentVertex.type== "AndJoin" ||
        currentVertex.type== "XorJoin") {
            n –;
            currentVertex.level=n;
 controlNodes.push(currentControlNodes.get
 (currentVertex.level),currentVertex);
            // get the spilt activity s associated with level n
            from currentControlNodes stack and
            // then push the pair (s, currentVertex) into
            controlNodes stack
            currentControlNodes.pop(currentVertex.level);
            if (n<level) exit while;
            else currentVertex= currentVertex.next;
        } else {
            currentVertex.level=n;
            currentVertex= currentVertex.next;
        }
    }
    return controlNodes;
}
```

5.2. Anomaly detection methods

The methods for detecting the artifact usage anomalies in a process specification are presented in this section, named DetectMissingProduction, DetectRedundantWrite and DetectConflictWrites, respctively. Each is aimed at detecting a type of artifact usage anomalies identified in Section 4. The details of these methods are shown in the following subsections. An example described in Section 5.2.1.3 is adopted to demonstrate their how they work correspondingly.

5.2.1. Method for detecting missing production anomalies

5.2.1.1. Calculation of propagated artifacts from predecessors. Given a sequence S of the process derived from transformation, let $S.AA_v$ denote the set of artifacts propagated from the predecessors of activity v and $S.AA'_v$ be the set of artifacts of which each can be propagated to the direct successors of v after the execution of v. At the top level, $S.AA_{S.startVertex} = I_w$ for the starting node of S. During the traverse of sequence S, when a node n is reached, $S.AA$ is calculated as follows:

- If n represents a task activity v,v has only one direct successor x. $S.AA'_v$ and $S.AA_x$ are calculated as follows:
  - For each destroyed artifact $d \in I_v \cap O_v^-$, remove d from $S.AA_v^u$ or $S.AA_v^c$ where d is included.
  - For each produced artifact $d \in (O_v^+ \setminus I_v)$, add d to $S.AA_v^u$ and remove d from $S.AA_v^c$ if d is included.

Table 5.5
Steps to calculate the conflict write.

ps	$PA = \{d_1, d_2, d_3, d_{13}\}$, UA = ∅, RA = ∅
t_1	$PA = \{d_1, d_2, d_3, d_{13}\}$, UA = ∅, RA = $\{d_1, d_2, d_3, d_{13}\}$
t_2	$PA = \{d_1, d_2, d_3, d_{13}, (d_5), (d_6), (d_7)\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_3, d_{13}\}$
t_3	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_3, d_{13}, (d_6)\}$
as_1	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_3, d_{13}, d_6\}$
xs_1	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, (d_3)\}$
as_2	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_9)\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
t_4	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_8)\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
t_5	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_{10})\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
aj_2	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
t_6	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}, (d_9)\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, (d_8)\}$
as_3	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_9)\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
$t_{4'}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_8)\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
$t_{5'}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, (d_{10})\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
aj_3	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3\}$
$t_{6'}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}, (d_9)\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, (d_8)\}$
as_4	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_8, d_{10}, (d_9)\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, d_8\}$
$t_{4''}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_{10}, d_9, (d_8)\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, d_8\}$
$t_{5''}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_8, d_9, (d_{10})\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, d_8\}$
aj_4	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, d_8\}$
$t_{6''}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}, (d_9)\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, (d_8)\}$
xj_1	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, d_8\}$
t_7	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, (d_3), (d_9), (d_{11})\}$
t_8	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_{11})\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_3, d_9, d_{11}\}$
t_9	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_3, d_{13}, d_6, (d_4)\}$
xs_2	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, (d_3)\}$
t_{10}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, (d_{12})\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3\}$
t_{11}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, (d_3), (d_{12})\}$
xj_2	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_{12}\}$, $UA = \{d_3, d_{13}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}\}$
aj_1	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
t_{12}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_9, d_{11}, (d_6)\}$
xs_3	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, (d_{15})\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}, (d_{13})\}$
xs_4	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
t_{13}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, (d_{14})\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_{11}, (d_6), (d_9)\}$
t_{14}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, (d_{14})\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_{11}, (d_6), (d_9)\}$
xj_4	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
t_{15}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9, (d_{15})\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
xs_5	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
$t_{13'}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, (d_{14})\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_{11}, (d_6), (d_9)\}$
$t_{14'}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, (d_{14})\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_{11}, (d_6), (d_9)\}$
xj_5	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
$t_{15'}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9, (d_{15})\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
xs_6	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
$t_{13''}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, (d_{14})\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_{11}, (d_6), (d_9)\}$
$t_{14''}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, (d_{14})\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_1, d_2, d_{13}, d_4, d_3, d_{12}, d_8, d_{11}, (d_6), (d_9)\}$
xj_6	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
$t_{15''}$	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9, (d_{15})\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
xj_3	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_1, d_2, d_{13}, d_6, d_4, d_3, d_{12}, d_8, d_9, d_{11}\}$
t_{16}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}, (d_{16})\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_4, d_{12}, d_8, d_{11}, (d_1), (d_2), (d_3), (d_6), (d_9), (d_{13}), (d_{15})\}$
as_5	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}, d_{16}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_4, d_{12}, d_8, d_{11}, d_1, d_2, d_3, d_6, d_9, d_{13}, d_{15}\}$
t_{17}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}, d_{16}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_4, d_{12}, d_8, d_{11}, d_1, d_2, d_3, d_6, d_9, d_{15}, (d_{13}), (d_{16})\}$
t_{18}	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}, d_{16}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_4, d_{12}, d_8, d_{11}, d_1, d_2, d_3, d_6, d_9, d_{13}, d_{15}, (d_{16})\}$
aj_5	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}, d_{16}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_4, d_{12}, d_8, d_{11}, d_1, d_2, d_3, d_6, d_9, d_{13}, d_{15}, d_{16}\}$
pe	$PA = \{d_1, d_2, d_3, d_{13}, d_5, d_6, d_7, d_9, d_8, d_{10}, d_{11}, d_{12}, d_{15}, d_{14}, d_{16}\}$, $UA = \{d_3, d_{13}, d_9, d_{15}\}$, $RA = \{d_4, d_{12}, d_8, d_{11}, d_1, d_2, d_3, d_6, d_9, d_{13}, d_{15}, d_{16}\}$

$$S.AA_x = S.AA'_v = \begin{cases} S.AA'^u_v = (S.AA^u_v \setminus (I_v \cap O^-_v)) \cup (O^+_v \setminus I_v) \\ S.AA'^c_v = (S.AA^c_v \setminus (I_v \cap O^-_v)) \setminus (O^+_v \setminus I_v) \end{cases}$$

- If n represents a control block with subsequences $SS = (SS_1, SS_2, \ldots, SS_k)$ and $k \geqslant 2$, each vertex within the block will be recursively traversed as follows:
  - The traverse starts from *n.startVertex* which is the start vertex of the control block.
    - $S.AA'_{n.startVertex} = S.AA_{n.startVertex}$, since *n.startVertex* the start vertex of the control block.
  - For each subsequence SS_i, apply method DetectMissingProduction recursively to calculate each $SS_i.AA$.
  - The traverse terminates on *n.endVertex* which is the end vertex of the control block. All $SS_i.AA$ are merged according to the type of the control block.
    - If n is an *XOR* control block,

$$S.AA'_{n.endVertex} = S.AA_{n.endVertex} = \begin{cases} S.AA^u_{n.endVertex} = \bigcap_{i=1}^{k} SS_i.AA^u_{SS_i.endVertex} \\ S.AA^c_{n.endVertex} = \bigcup_{i=1}^{k} SS_i.AA_{SS_i.endVertex} \setminus S.AA^u_{n.endVertex} \end{cases}$$

    - If n is an *And* control block,

$$S.AA'_{n.endVertex} = S.AA_{n.endVertex} = \begin{cases} S.AA^u_{n.endVertex} = \bigcup_{i=1}^{k} SS_i.AA^u_{SS_i.endVertex} \\ \qquad \setminus \bigcup_{i=1}^{k} (SS_i.O^- \setminus SS_i.AA^u_{SS_i.endVertex}) \\ S.AA^c_{n.endVertex} = \bigcup_{i=1}^{k} SS_i.AA_{SS_i.endVertex} \setminus S.AA^u_{n.endVertex} \\ \qquad \setminus \bigcup_{i=1}^{k} \left(SS_i.O^- \setminus SS_i.AA^u_{SS_i.endVertex} \right) \end{cases}$$

5.2.1.2. Rules for detecting missing production anomalies. During the traverse of sequence S, the missing production anomalies defined in Section 4.2 can be detected with the following rules:

- **No Propagation**
  - When an activity v is reached and $MA^u_v = I_v \setminus AA_v \neq \emptyset$, a missing production anomaly occurs for each artifact $d \in MA^u_v$ due to *No Propagation*.
- **Conditional Propagation**
  - When an activity v is reached and $MA^c_v = I_v \cap AA^c_v \neq \emptyset$, a missing production anomaly occurs for each artifact $d \in MA^c_v$ due to *Conditional Propagation*.
- **Uncertain Propagation**

For an *And* control block with subsequences $SS = (SS_1, SS_2, \ldots, SS_k)$ and $k \geqslant 2$, before merging $SS_i.AA_{SS_i.endVertex}$ of subsequences, if $\exists i, j \wedge 1 \leqslant i, j \leqslant k \wedge i \neq j \wedge (UP_{SS_i,SS_j} = SS_i.I \cap (SS_j.I \cap SS_j.O^-) \neq \emptyset)$, a missing production anomaly may occur for each artifact $d \in UP_{SS_i.SS_j}$ due to *Uncertain Propagation*.

5.2.1.3. An example: a process of resolving issues through e-mail votes. To demonstrate the three analysis methods proposed in Section 5, this subsection introduces a process for resolving issues through e-mail votes (White, 2008) as an example. The methods presented are applied on this example to illustrate the steps to detect the artifact usage anomalies. Fig. 5.1 shows the control flow graph of the e-mail voting process where the artifacts are stated with details in Table 5.1. The result of representing the example with our process model is shown in Fig. 5.2. The artifact usages of activities are listed in Table 5.2.

5.2.1.4. Detection of missing production anomalies. Table 5.3 shows the calculation of propagated artifacts according to the activity taken order of the e-mail voting process introduced in Section 5.2.1.3. During detection, the method processes each activity once only and generates two sets of artifacts on each activity. The rules defined are applied on the calculated results of the operated activity recursively. The activity anomalies detected are listed in the table.

5.2.2. Method for detecting redundant write anomalies

5.2.2.1. Calculation of redundant write. Given a sequence S of the process derived from transformation, let $S.NC_v$ denote the set of artifacts unused before activity v and $S.NC'_v$ denote the set of artifacts unused after executing v. During the traverse of sequence S, when a node n is reached, $S.NC$ is calculated as follows:

- If n represents a task activity v, $S.NC'_v = \begin{cases} S.NC'^u_v = (S.NC^u_v \setminus I_v) \cup O^+_v, \\ S.NC'^c_v = S.NC^c_v \setminus I_v \setminus O^+_v. \end{cases}$
  - For each read or destroyed artifact $d \in I_v$ or $d \in I_v \cap O^-_v$, remove d from NC^u_v and NC^c_v.
  - For each produced or updated artifact $d \in O^+_v$, add d to NC^u_v and remove d from NC^c_v.
- If n represents a control block with subsequences $SS = (SS_1, SS_2, \ldots, SS_k)$ and $k \geqslant 2$, method DetectRedundantWrite is recursively applied to calculate $SS_i.NC$ of branches and then merge $SS_1.NC$ to $SS_k.NC$ according to the type of the control block.
  - If n is an *XOR* control block,

$$S.NC'_{n.endVertex} = S.NC_{n.endVertex} = \begin{cases} S.NC^u_{n.endVertex} \\ \quad = \bigcap_{i=1}^{k} SS_i.NC^u_{SS_i.endVertex} \\ S.NC^c_{n.endVertex} \\ \quad = \bigcup_{i=1}^{k} SS_i.NC_{SS_i.endVertex} \setminus S.NC^u_{n.endVertex} \end{cases}$$

  - If n is an *And* control block,
    - the unconditional set $S.NC^u_{n.endVertex}$ is constituted with two parts:
      - the intersection $\bigcap_{i=1}^{k} SS_i.NC^u_{SS_i.endVertex}$ of the sets of artifacts unused before the end vertexes of subsequence $SS_i, 1 < i \leqslant k$, and
      - the artifacts in $(\bigcup_{i=1}^{k} SS_i.NC^u_{SS_i.endVertex} \setminus S.NC^u_{n.startVertex}) \setminus \bigcup_{i=1}^{k} (SS_i.I \setminus SS_i.NC^u_{SS_i.endVertex})$ which contains the ones produced in subsequences without using by the parallel activities before the end vertex of n. The case of an artifact reproduced after the last use in branch SS_i is concerned.

$$S.NC'_{n.endVertex} = S.NC_{n.endVertex} = \begin{cases} S.NC^u_{n.endVertex} \\ \quad = \bigcap_{i=1}^{k} SS_i.NC^u_{SS_i.endVertex} \\ \quad \cup \left(\bigcup_{i=1}^{k} SS_i.NC^u_{SS_i.endVertex} \setminus S.NC^u_{n.startVertex} \right) \\ \quad \setminus \bigcup_{i=1}^{k} (SS_i.I \setminus SS_i.NC^u_{SS_i.endVertex}) \\ S.NC^c_{n.endVertex} \\ \quad = \bigcup_{i=1}^{k} SS_i.NC_{SS_i.endVertex} \setminus S.NC^u_{n.endVertex} \end{cases}$$

5.2.2.2. Rules for detecting redundant write anomalies. After visiting the *endVertex* of the top level sequence *S*, i.e. the end vertex of the process, the redundant write anomalies defined in Section 4.3 can be detected with the following rules:

- **Explicit redundant write**
  - If $EC = NC^{u}_{S.endVertex} \setminus O_w \neq \emptyset$, a redundant write anomaly occurs for every artifact $d \in EC$ due to *No Consumption After Last Write*.
- **Potential redundant write**
  - If $CC = NC^{c}_{S.endVertex} \setminus O_w \neq \emptyset$, a redundant write anomaly occurs for every artifact $d \in CC$ due to *Conditional Consumption After Last Write*.

5.2.2.3. Detection of redundant write anomalies. Table 5.4 shows the steps to calculate the set of unused artifacts for each activity in the e-mail voting process introduced in Section 5.2.1.3. During calculation, the method processes each activity once only and generates two sets of artifacts on each activity.

In this case, the redundant write anomalies detected belong to one of the following classes:

- **Explicit redundant write** $EC = NC^{u} \setminus O_w = \{d_5, d_7, d_{11}, d_{14}\} \setminus \{d_{16}\} = \{d_5, d_7, d_{11}, d_{14}\}$ is not empty and thus, a redundant write anomaly occurs for every artifact $d \in EC$ due to *Completely Unused for the Process*.
- **Potential redundant write** $CC = NC^{c} \setminus O_w = \{d_{12}\} \setminus \{d_{16}\} = \{d_{12}\}$ is not empty and thus, a redundant write anomaly occurs for d_{12} due to *Conditional unused for the process*.

5.2.3. Method for detecting conflict write anomalies

5.2.3.1. Calculation of conflict writes. Given a sequence *S* of the process derived from transformation, let *S.PA*, *S.UA* and *S.RA* be the sets of artifacts produced, updated and read respectively within sequence *S*. Initially, the three sets are empty. During the traverse of sequence *S*, when a node *n* is reached, *S.PA*, *S.UA* and *S.RA* are calculated as follows:

- If *n* represents a task activity *v*, $\begin{cases} S.PA = S.PA \cup (O^{+}_{v} \setminus I_v), \\ S.UA = S.UA \cup (O^{+}_{v} \cap I_v), \\ S.RA = S.RA \cup (I_v \setminus O^{+}_{v} \setminus O^{-}_{v}). \end{cases}$
  - For every artifact *d* produced by *v*, i.e. $d \in (O^{+}_{v} \setminus I_v)$, add *d* to *S.PA*.
  - For every artifact *d* updated by *v*, i.e. $d \in (O^{+}_{v} \cap I_v)$, add *d* to *S.UA*.
  - For every artifact *d* read by *v*, i.e. $d \in (I_v \setminus O^{+}_{v} \setminus O^{-}_{v})$, add *d* to *S.RA*.
- If *n* represents a control block with subsequences $SS = (SS_1, SS_2, \ldots, SS_k)$ and $k \geqslant 2$, the method is recursively applied to each subsequence. Then, $SS_i.PA$, $SS_i.UA$ and $SS_i.RA$ belong to the subsequence and are merged according to the following rules:

$$S.PA = S.PA \cup \bigcup_{i=1}^{k} SS_i.PA,$$

$$S.UA = S.UA \cup \bigcup_{i=1}^{k} SS_i.UA,$$

$$S.RA = S.RA \cup \bigcup_{i=1}^{k} SS_i.RA.$$

5.2.3.2. Rules for detecting conflict write anomalies. For an *And* control block with subsequences $SS = (SS_1, SS_2, \ldots, SS_k)$ and $k \geqslant 2$, the conflict write anomalies defined in Section 4.4 can be detected with the following rules:

- Multiple parallel productions
  - Before merging $SS_i.PA$, if $\exists i,j \wedge 1 \leqslant i,j \leqslant k \wedge i \neq j \wedge (MPA_{ss_i,ss_j} = SS_i.PA \cap SS_j.PA \neq \emptyset)$ a conflict writes anomaly may occur for every artifact $d \in MPA_{ss_i,ss_j}$ due to multiple parallel productions.
- Multiple parallel updates
  - Before merging $SS_i.UA$, if $\exists i,j \wedge 1 \leqslant i,j \leqslant k \wedge i \neq j \wedge (MUA_{ss_i,ss_j} = SS_i.UA \cap SS_j.UA \neq \emptyset)$ a conflict writes anomaly may occur for every artifact $d \in MUA_{ss_i,ss_j}$ due to multiple parallel updates.
- Parallel read and update
  - Before merging $SS_i.UA$ and $SS_i.RA$, if $\exists i,j \wedge 1 \leqslant i,j \leqslant k \wedge i \neq j \wedge (SS_i.RA \cap SS_j.UA \neq \emptyset)$, then a conflict writes anomaly may occur for every artifact $d \in SS_i.RA \cap SS_j.UA$ due to parallel read and update.

5.2.3.3. Detection of conflict write anomalies. Table 5.5 shows the steps to calculate the sets of artifacts produced, updated and read by activities in the e-mail voting process introduced in Section 5.2.1.3. During calculation, the method processes each activity once only and generates three sets of artifacts on each activity.

In this case, the conflict write anomalies detected belong to one of the following classes:

- **Multiple parallel productions**
  $MPA_{xj_1,t_{11},xj_2} = (PA_{xj_1} \setminus PA_{as_1}) \cap (PA_{t_8} \setminus PA_{as_1}) \cap (PA_{xj_2} \setminus PA_{as_1}) = \emptyset$ and $MPA_{t_{17},t_{18}} = \emptyset$ are empty and thus, no conflict write anomaly occurs because of *multiple parallel productions*.
- **Multiple parallel updates**
  $MUA_{xj_1,t_{11},xj_2} = (UA_{xj_1} \setminus UA_{as_1}) \cap (UA_{t_8} \setminus UA_{as_1}) \cap (UA_{xj_2} \setminus UA_{as_1}) = \emptyset$ and $MUA_{t_{17},t_{18}} = \emptyset$ are empty and thus, no conflict write anomaly occurs due to *multiple parallel updates*.
- **Parallel read and update**
  $SS_{xj_1}.UA \cap SS_{t_8}.RA = (UA_{xj_1} \setminus UA_{as_1}) \cap (RA_{t_8} \setminus RA_{as_1}) = \{d_9\}$ is not empty and thus, a conflict write anomaly occurs for d_9 due to *parallel read and update*.

6. Comparisons

6.1. Comparisons of anomalies identified in the approaches

The artifact usage anomalies identified in this paper are classified according to the classification proposed by Sun and Zhao (2004), Sun et al. (2004, 2006) who claimed that the definition of the types of artifact usage anomalies – missing data, redundant data, and conflicting data – is sufficient in analyzing the data flow at a conceptual level. Each anomaly, addressed by the two groups, Sadiq et al. (2004) and Sun and Zhao (2004), Sun et al. (2004, 2006), working on the analysis in a data-flow dimension, can be found a correspondence in our model. The mapping relations are shown in Table 6.1. Besides, Theorems 1–3 can be applied to help avoid the anomalies.

However, Sadiq et al. (2004), Sun and Zhao (2004), Sun et al. (2004, 2006) are not concerned with destroy operations, each of which is essential for deleting the artifact instance which is not required by the succeeding activities or a member of the process outputs. In their approaches, a missing production anomaly caused by an activity destroying an artifact before it is consumed cannot be identified. A lack of recognizing the critical anomalies endanger the accuracy of analyzing the workflow execution result. The destroy operations are concerned in our previous work (Hsu and Wang, 2007), but the anomalies are presented conceptually. There are three distinct anomalies – early destruction, conditional destruction, and uncertain destruction – caused by destroying an artifact requested in a succeeding process, formulated here.

Table 6.1
The mappings of the anomalies addressed.

Our approach		Sun et al.		Sadiq et al.	Our Previous Work (Hsu and Wang, 2007)
Missing production	No production	Missing data	Absence of initialization	Missing data Insufficient Data Mismatched Data	No producer
	Delayed production		Delayed initialization	Misdirected data	
	Conditional production		Improper routing		Branch hazard
	Exclusive production			N/A	Branch hazard
	Uncertain production		Uncertain availability	Misdirected data	Parallel hazard
Redundant write	Conditional consumption after last write	Redundant data	Contingent redundancy	Redundant data mismatched data	Branch hazard
	No consumption after last write		Inevitable redundancy		No consumer
Conflict write	Multiple parallel productions	Conflict data	Multiple initializations	Lost data	Contradiction

In addition, the anomalies happening within a control block are illustrated conceptually in the methods proposed in Sadiq et al. (2004), Hsu and Wang (2007). There is no formal formulation for systematically discovering the artifact usage anomalies in a workflow model. Furthermore, the cases of causing conflict write anomalies: more than one activity initializes or updates an identical artifact in parallel or two activities perform read and update on the same artifact concurrently, are partially addressed in Sadiq et al. (2004), Sun and Zhao (2004), Sun et al. (2004, 2006). These anomalies are identified and formulated in this paper also. Regarding the conflict write anomalies, the comparison between the anomalies formulated in our method and those identified in Sun et al. (2006), Sadiq et al. (2004) and Hsu and Wang (2007) are shown in Table 6.2.

6.2. Comparison of the detection methods

Our previous work creates a table for each artifact to display the artifact usage. In contrast with the table, we utilize another presentation, e.g. Table 5.2, to show the operation of each activity performing on each artifact. Comparing with the distributed tables, it is easier to identify how each artifact is processed in a business process with the integrated presentation.

Sun et al. (2006) proposed a process data diagram through extending the UML activity diagram. The extension enriches the presentation power of an activity diagram to support artifact usage analysis. However, the irrelevant information could be represented in a diagram to interfere in the artifact usage analysis. Comparing to the extension, the methods published in Son and Kim (2005), Chang et al. (2002), Hsu and Wang (2007) are defined with their own notations. These custom-made notations are constructed without explaining the reasons of redefining a new one which corresponds to a specific one specified in BPMN. Adopting the BPMN notation instead of using BPMN is another case of redefinition. Our process model is defined by ridding unrelated elements and keeping related ones of the BPMN for fitting the model to analyze artifact usage. Three advantages obtained from the adaptation are that: (1) using a standardized graphical notation BPMN keeps the understandability held by it, (2) simplifying the arrangement of structures in a model reduces the analysis complexity and (3) eliminating the elements irrelevant to artifact usage analysis diminishes the possibility of design errors.

In addition, our detection methods work on a sequence of nodes transformed from a business process. Let the process specification be $BP = (G, VT, D, I_W, O_W)$. The transformation mechanism goes through each vertex and edge of $G(V,E)$ twice only to

Table 6.2
The mapping of the conflict write anomalies addressed in the four approaches.

Our approach		Sun et al.	Sadiq et al.	Our previous work
Conflict write	Multiple parallel productions	Multiple initializations	Lost data (conceptual)	Parallel hazard (conceptual)
	Multiple parallel updates	N/A		
	Parallel read and update		N/A	

transform the graph into a sequence of nodes. Thus, the complexity of the transformation is $O(|V| + |E|)$. Existing approaches (Sun et al., 2006, Hsu and Wang, 2007) detect the anomalies defined upon critical paths, but the computing time required for calculating critical paths is $O(|V|^2)$ more than the one of the transformations.

Our detection methods are assigned individually to detect a particular type of defined artifact usage anomalies over the sequence of nodes. During detection, the methods process each node once only and generate two or three sets of artifacts on each node. The maximum storage demand of the set is $(|D|)$, a bit-array is utilized for representing each element of the sets. Based on the data structure, time consumption of the operations, such as union, intersection, and complement, working on the elements is seen as constant.

If the number of nodes in *BP* is n, method *DetectRedundantWrite* can detect the anomaly in $O(n)$ time. If the *destroy* operation is not considered in method *DetectMissingProduction*, the time required by detecting missing production anomalies is $O(n)$ also. Otherwise, involving *destroy* operation, checking the missing production anomalies on each pair of branches is requisite for each *AND* control block. Thus, with *destroy* operations, the time required by executing method *DetectMissingProduction* is $O(n^2 + n) = O(n^2)$. In addition, checking the conflict write anomalies on each pair of branches is requisite for each *AND* control block, thus the time required by executing method *DetectConflictWrites* is $O(n^2 + n) = O(n^2)$, less than $O(n^3)$ time in previous approaches.

7. Conclusion and future work

Introducing an artifact usage analysis technique into workflow design phase is the main contribution of this paper. This paper presents a business process model for describing a business process and analyzes the artifact usages on this model to achieve this goal. Artifact usage in our model is characterized by its state transition diagram.

This work identifies thirteen cases of improper artifact usage affecting workflow execution and categorizes these anomalies into three types. The anomalies are identified by considering the destroy operations which are lack of concern before. A set of methods for discovering these anomalies is also presented. The execution time required by the set of methods is less than the methods proposed before. An example is demonstrated the usability of the proposed methods.

We currently continue our research in several directions. First, we plan to implement the proposed model and methods on current workflow management systems, such as Agentflow Flowring Technology Corp. (2006), so that our research result can be tested in real-world applications. Second, we continue the analysis on composite artifacts with more complex usages using *Revise* operations. Thirdly, we integrate resource constrains analysis techniques with our work to build a practical workflow design methodology.

References

Alves, A. et al., 2007. Web Services Business Process Execution Language 2.0, OASIS WSBPEL TC.

Bae, J., Bae, H., Kang, S.-H., Kim, Y., 2004. Automatic control of workflow processes using ECA rules. IEEE Transaction on Knowledge and Date Engineering 14 (8), 1010–1023.

Chang, D.-H., Son, J.H., Kim, M.H., 2002. Critical path identification in the context of a workflow. Information and Software Technology 44 (7), 405–417.

Curtis, B., Kellner, M.I., Over, J., 1992. Process modeling, CACM 35(9) 75–90.

Du, W., Shan, M.C., 1999. Enterprise workflow resource management. In: Proceedings of the Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, IEEE Computer Society, March 1999, pp. 108–115.

Flowring Technology Corp. <http://www.flowring.com> (accessed May 2006).

Gong, L., Wang, H.-Y., 2004. A method to verify the soundness of workflow control logic. Computer Supported Cooperative Work in Design 1 (May), 284–388.

Hitomi A.S., Le, D., 1998. Endeavors and component reuse in web-driven process workflow. In: Proceedings of the California Software Symposium, Irvine, CA, USA, October 1998, pp. 15–20.

Hsu, H.-J., 2005. Using state diagrams to validate artifact specifications on primitive workflow schema, National Chiao-Tung University, M.S. Thesis.

Hsu, C.L., J Wang, F., 2007. Analysing inaccurate artifact usages in workflow specifications. Software IET 1 (5), 188–205.

Jablonski, S., Bussler, C., 1996. Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, London, UK.

Karamanolis, C., Giannakopoulou, D., Magee, J., Wheater, S.M., 2000a. Model checking of workflow schemas. In: Proceeding of Fourth International Enterprise Distributed Object Computing Conference (EDOC'00), IEEE Computer Society, September 2000, pp. 170–179.

Karamanolis, C., Giannakopoulou, D., Magee, J., Wheater, S.M., 2000b. Formal verification of workflow schemas, Technical Report, Control and Coordination of Complex Distributed Services, ESPRIT Long Term Research Project.

Li, H., Yang, Y., Chen, T.Y., 2004. Resource constraints analysis of workflow specifications. Journal of Systems and Software 73 (2), 271–285.

Liu, C., Lin, X., Orlowska, M.E., Zhou, X., 2003. Confirmation: increasing resource availability for transactional workflows. Information Sciences 153 (1), 37–53.

Muehlen, M.Z., 1999. Resource modeling in workflow applications. In: Proceedings of the 1999 Workflow Management Conference, Münster, Germany, November, pp. 137–153.

Ouyang, C. et al., 2006. Translating BPMN to BPEL.

Russell, N., ter Hofstede A.H.M., Edmond, D., van der Aalst W.M.P., 2004. Workflow data patterns, QUT Technical Report, FIT-TR-2004-01, Queensland University of Technology, Brisbane.

Sadiq W., Orlowska, M.E., 1997. On correctness issues in conceptual modeling of workflows. In: Proceedings of the 5th European Conference on Information Systems (ECIS '97), Cork, Ireland, June 19–21, 1997.

Sadiq W., Orlowska, M.E., 1999. Applying graph reduction techniques for identifying structural conflicts in process models. In: Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99). Lecture Notes in Computer Science, vol. 1626, Springer-Verlag, Berlin, pp. 195–209.

Sadiq, W., Orlowska, M.E., 2000. Analyzing process models using graph reduction techniques. Information Systems 25 (2), 117–134.

Sadiq, S., Orlowska, M.E., Sadiq, W., Foulger, C., 2004. Data flow and validation in workflow modeling. In: Proceedings of the 15th Australasian database conference, Dunedin, New Zealand, January 2004, pp. 207–214.

Senkul, P., Toroslu, I.H., 2005. An architecture for workflow scheduling under resource allocation constraints. Information Systems 30 (5), 399–422.

Son, J.H., Kim, M.H., 2005. Extracting the workflow critical path from the extended well-formed workflow schema. Journal of Computer and System Sciences 70 (1), 86–106.

Sun, S.X., Zhao, J.L., 2004. A data flow approach to workflow design. In: Proceedings of the 14th Workshop on Information Technology and Systems (WITS'04), 2004, pp. 80–85.

Sun, S.X., Zhao, J.L., Sheng, O.R., 2004. Data flow modeling and verification in business process management. Proceedings of the AIS Americas Conference on Information Systems 5 (8), 4064–4073.

Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L., 2006. Formulating the data flow perspective for business process management. Information Systems Research 17 (4), 374–391.

The Workflow Management Coalition, "The workflow reference model", Document Number TC00-1003, January 1995.

van der Aalst, W.M.P., 1997. Verification of workflow nets. In: Proceedings of the 18th International Conference on Application and Theory of Petri Nets, Toulouse, France, June 23–27, 1997, pp. 407–426.

van der Aalst, W.M.P., 1998a. The application of petri nets to workflow management. Journal of Circuits, Systems and Computers 8 (1), 21–66.

van der Aalst, W.M.P., 1998b. The application of petri-nets to workflow management. Journal of Circuits, Systems and Computers 8 (1), 21–66.

van der Aalst, W.M.P., Basten, T., 2002. Inheritance of workflows: an approach to tackling problems related to change. Theoretical Computer Science 270 (1–2), 125–203.

van der Aalst, W.M.P., ter Hofstede, A.H.M., 2000. Verification of workflow task structures: a petri-net-based approach. Information Systems 25 (1), 43–69.

van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P., 2000a. Workflow patterns, BETA Working Paper Series, WP 47, Eindhoven University of Technology, Eindhoven.

van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P., 2000b. Advanced workflow patterns. In: Proceedings of 7th International Conference on Cooperative Information Systems (CoopIS 2000), Lecture Notes in Computer Science, vol. 1901, Springer-Verlag, Berlin, pp. 18–29.

van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P., 2003. Workflow patterns. Distributed and Parallel Databases 14 (1), 5–51.

Verbeek H.M.W., van der Aalst, W.M.P., 2000. Woflan 2.0: a petri-net-based workflow diagnosis tool. In: Proceedings of the 21st International Conference of Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark, June 26–30, 2000, pp. 475–484.

Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P., 2001. Diagnosing workflow processes using woflan. The Computer Journal 44 (4), 246–279.

Wang, F.-J., Hsu, C.-L., Hsu, H.-J., 2006. Analyzing inaccurate artifact usages in a workflow schema. In: Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), vol. 2, September 17-21, pp. 109–114.

White, S.A., 2008. Business Process Modeling Notation (BPMN) Version 1.1, Business Process Management Initiative, BPMI.org.

Yau S.S. et al., 2007. An approach to adaptive distributed execution monitoring for workflows in service-based systems. In: Proceedings of 31st Annual International Computer Software and Applications Conference, 2007, pp. 211–216.

Yau S.S. et al., 2008. An intelligent control architecture for adaptive service-based software systems with workflow patterns. In: Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference, July, 2008, pp. 824–829.

Zhuge, H., 2003. Component-based workflow systems development. Decision Support Systems 35 (4), 517–536.

Ching-Huey Wang received the MS degree in computer science from Tung Hai University, Taiwan, in 2002. She is a PhD student of computer science at Chiao Tung University, Taiwan. Her research interests include service-oriented application modeling and analysis, social network service, and pervasive workflow application. She has published several papers in international journals and conferences. She is a member of the IEEE Computer Society.

Prof. Feng-Jian Wang completed his Ph.D. program in Dept. of E.E.C.S., Northwestern University, 1988. Since then, he worked in National Chiao-Tung University, Taiwan. During his Ph.D program, he worked on incremental analysis of data flow. Thereafter, he worked on the development, reuse, and data analysis based no object-oriented programming language. Since 1995, he worked on the analysis and design of workflow programs and his laboratory constructed a workflow management system named Agentflow, where he studied a series of supporting analysis and tools associated with editing activities. Currently, he is focused on how to apply workflow and grid computing techniques on pervasive workflow systems, and developing the design patterns of FLASH programs.