Detecting and Repairing Unintentional Change in In-use Data in Concurrent Workflow Management System

Phan Thi Thanh Huyen and Koichiro Ochimizu

School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, 923-1292, Japan
{huyenttp, ochimizu}@jaist.ac.jp

Abstract. Workflow verification has attracted a lot of attention, especially control flow aspect. However, little research has been carried out on data verification in workflow literature although data is one of the most important aspects of workflow. This paper proposes an approach for detecting and repairing **Unintentional Change in In-use Data** (*UCID*) in a Concurrent Workflow Management System at build time. We define UCID as a situation in which some data values are lost or some data elements are assigned values different from the intentions of workflow designers due to non-deterministic access to shared data by different activities. Differently from previous studies, we consider UCID in two different ways: between concurrent activities in a single workflow (*intra-UCID*) and between activities in different concurrent workflows (*inter-UCID*). In this paper, we first investigate UCID situations in a workflow management system, and then we define a Time Data Workflow, an extension of the WF-Nets with time and data factors, with many attributes supporting UCID detection and correction. Based on these definitions, we develop an algorithm which helps to detect potential intra/inter-UCID at build time, along with algorithm evaluation and UCID resolution methods. Finally, we introduce a concrete project on building a change support environment for cooperative software development using UCID theory.

Keywords: Unintentional Change in In-use Data, Time Data Workflow, concurrent workflows, algorithm, Workflow Nets

1 Introduction

Correctness of a workflow model is very important, because any errors in workflow can lead to execution failure of the corresponding process. Therefore, workflow should be verified carefully before execution to reduce risks to the target process. Workflow verification has received a lot of attention since the birth of the workflow concept. However, researchers have only focused on structure verification, temporal verification and resource verification [2] [4] [7] [9]. Most verification techniques ignore data aspect and there is little support for data flow verification. Previous works on the data flow aspect have concentrated on detecting common data flow errors such as missing data,

Recent Advances in Petri Nets and Concurrency, S. Donatelli, J. Kleijn, R.J. Machado, J.M. Fernandes (eds.), CEUR Workshop Proceedings, ISSN 1613-0073, Jan/2012, pp. 331–351.

redundant data, inconsistent data, garbage data, etc. Among them, Unintentional Change in In-use Data (UCID) is perhaps one of the most dangerous and common problems. We define UCID as a situation in which some data values are lost or some data elements are assigned values different from the intentions of workflow designers due to non-deterministic access to shared data by different activities. Assuming that workflow is free of control errors, and activities in workflow can be scheduled within temporal constraint, we aim to support data verification in the workflow model by concentrating on UCID detection and correction.

Existing approaches have addressed this problem by detecting potential UCID patterns, limited to concurrent activities of a single workflow. Unfortunately, this error can cross a single workflow boundary. In a Workflow Management System (WFMS), in fact, there exist many workflows executing at the same time, which we call *Concurrent Workflows*, and they may be correlated if two activities from different workflows use shared data. Even if the data flow of each workflow is correct, we cannot ensure correctness of the whole system because of the mutual interactions among workflows. The problem is how to detect non-deterministic access to shared data of activities belonging to not only the same workflow but also different workflows and how to repair this kind of data abnormality.

Reference [19] is our first efforts in handling the UCID problem. Potential UCID situations, Time Data Workflow (TDW) concepts, along with two algorithms for detecting intra-UCID and inter-UCID have been introduced in [19]. This paper is a refined and extended version of the [19]. In this paper, we redefine TDW as an extension of Workflow Nets (WF-Nets) [8] instead of Petri Nets as before. Based on these definitions and two algorithms for detecting intra/inter-UCID in [19], a revised version of UCID detection algorithm is built. Compared with the previous ones, this revised algorithm is more accurate and useful. Furthermore, some heuristics for making the algorithm more flexible and effective are discussed. UCID resolution methods are also proposed in this paper. Then, we illustrate this theory in practice by using it in designing workflows which represent change activities in a software change process.

Our approach in UCID detection is to observe behaviors of concurrent activities having data relation. In the case of activities in the same workflow, their total orders can be decided based on control flow. However, control flow does not help in the case of activities in different workflows. Therefore, we must use activities' execution time attribute to identify their total orders. Regarding UCID resolution, we take advantage of composition features of the Petri Nets to create new workflows with UCIDs removed.

The rest of this paper is organized as follows. Section 2 discusses the motivation of our research. Section 3 defines the Time Data Workflow (TDW), an extension of the Workflow Net with time and data factors. Section 4 introduces UCID situations caused by concurrent activities in the same workflow (*intra-UCID*) or activities in different concurrent workflows (*inter-UCID*) [19]. An algorithm for detecting potential UCID in both cases of intra/inter-UCID at build time, along with algorithm evaluation, is given in Section 5. Section 6 presents UCID resolution methods. Section 7 introduces our project on building a change support environment for cooperative software development. Theory about UCID problem is employed in this project to detect and repair data abnormalities among concurrent Change Support Workflows. Section 8 reports on

related work and finally, Section 9 concludes the paper and discusses points to future work.

2 Motivation

Let's take an example. We have two workflows W_1 and W_2, which are being executed independently. Workflow activities are modeled by rectangles, and data modified by an activity are written inside the corresponding rectangle. A small arrow is attached to a rectangle to denote an activity which is being executed. Data of the system are stored in a central repository. W_1 has five activities which modify A, X, B, C and D respectively. B and D are modified based on the value of X created by A_{12}. W_2 also has five activities which modify E, X, F, G and H respectively. Both A_{12} and A_{22} will modify X, but designers of W_1 and W_2, who don't have a comprehensive view of the whole system, may not recognize this problem. This is a common problem, especially in a big system with many workflows.

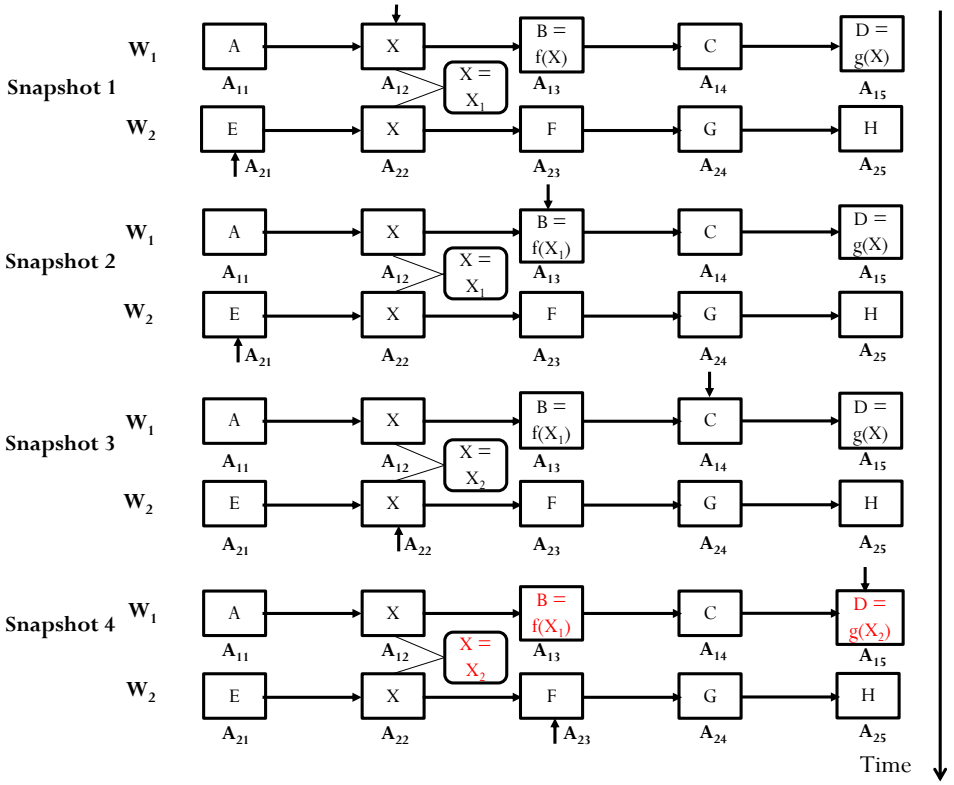


Fig. 1. Motivating example

Figure 1 describes some snapshots of the system at different time. For simplicity, we concentrate on describing the change in value of data elements relating to shared data X. In snapshot 1, A_{12} changes value of X to X_1. In snapshot 2, A_{13} changes value of B based on the value of X, X_1. In the next snapshot, A_{22} changes value of X from X_1 to X_2. In the last snapshot, A_{15} changes value of D based on the current value of X which is X_2. If X_1 is different from X_2, there are two problems in this scenario: X_1 is lost and D is assigned an unexpected value because D is modified based on the value X_2 instead of the value created by activity A_{12}, X_1. This is different from the intentions of the designers of the workflow W_1 and may cause an inconsistency between B and D. Regarding our definition of UCID, these errors are categorized into inter-UCID errors.

The first problem is similar to the lost update problem in database theory. Lost update problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect [20]. In this case, version control systems (VCSs) can be used if data of the system are individual artifacts like documents, source codes, etc. Version control is the management of changes to documents, programs, and other information stored as computer files. Changes are usually identified by a number or letter code, termed the "revision number". Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

Unfortunately, VCS cannot help to avoid the second problem. In this situation, if data of the system are stored in a central database, the database management system (DBMS) can provide some concurrency control techniques, which are used to ensure the noninterference or isolation property of concurrently executing transactions such as locking techniques, timestamp ordering based techniques, etc. A database transaction is a transaction which satisfies the ACID (atomicity, consistency, isolation and durability) properties. These properties should be enforced by the concurrency control and recovery methods of the DBMS [20]. However, in this method, we must specify the boundary of each transaction. This requirement is difficult to implement because there are many people involved in a workflow and people in a workflow may know nothing about other workflows. If the whole workflow is considered as a unique database transaction, it is impractical because a workflow may use many data elements and may happen for a long time.

If this type of errors is discovered at runtime, a recovery mechanism must be performed to ensure the correctness of the whole system. However, recovery is a rather expensive work, especially in a cooperative environment with many concurrently executing workflows. Therefore, detecting these errors as soon as possible is necessary to reduce risk to the target process.

This paper examines UCID situations in a general basic system without concerning which type of workflow data is stored in the central repository of the system and the implementation of the central repository as well.

Regarding inter-UCID, our problem domain is workflows whose data and estimated execution time can be decided at the design phase, for example workflows in the software evolution process. In these cases, an early UCID detection will help workflow designers to have a more comprehensive view of the system, and make timely adjustments to the original workflows to avoid error at runtime. We assume that the

following steps are conducted before workflow execution: identifying workflow activities and their orders, assigning activity properties (data, time…), and checking error using UCID detection and correction theory. If some potential UCID errors are detected, the first and second steps should be re-executed, based on suggested solutions given by UCID detection system.

With reference to workflows in which estimated execution time is not available at design time, UCID patterns and detection method will be used to detect UCID errors from workflow execution histories. However, this is out of the scope of this paper.

3 Time Data Workflow (TDW)

There are many ways to model a workflow, such as directed graphs, UML activity diagram, PERT, etc. In this paper, we chose the WF-Nets based approach to model workflow process, because it has many useful features needed in the area of business process modeling besides the mathematical nature of the underlying Petri Nets formalism [17].

WF-Nets is a subclass of Petri Nets dedicated for process/workflow modeling and analysis. Petri Nets is a popular graphical and mathematical modeling language in describing and analyzing systems which are characterized as concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic [17]. Formally, Petri Nets is a tuple PN = (P, T, F) where P is a finite set of places, T is a finite set of transitions ($P \cap T = \emptyset$) and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation) [8]. A Petri Nets PN = <P, T, F> is a WF-Nets if and only if there is one source place $i \in P$, one sink place $o \in P$ such that $\bullet i = \emptyset$, $o\bullet = \emptyset$, and every node $x \in P \cup T$ is on a path from i to o [8].

Our Time Data Workflow (TDW) is an extension of WF-Nets with time and data factors. Time and data are represented as attributes of transitions in a TDW. In this paper, we consider two types of relationships between an activity and a data element. First, an activity may *read* a particular data element as its input data. Second, an activity may *write* a particular data element as its output data. This means that this data element is assigned a new value. Inside an activity, *read* always happens before *write*. Assuming that durations of activities can be estimated at build time, we augment each activity A with two time values *min(A), max(A)* which describe the minimum and maximum execution durations of A respectively. The time unit is selected depending on specific workflow applications. Based on reference point P, which is the start time of its corresponding workflow, we can infer the *Earliest Start Time, EST(A)*, and the *Latest Finish Time, LFT(A)*, of A at run time. If *S(A), F(A)* are the *Start Time* and *Finish Time* of this activity at run time respectively, we can conclude that the *Active Interval* of A, *[S(A), F(A)]*, is within its *Estimated Active Interval, [EST(A), LFT(A)]*, that is, [S(A), F(A)] $\subseteq$ [EST(A), LFT(A)] [19].

In a TDW, activities are modeled by transitions, and causal dependencies are modeled by places and arcs, as shown in Figure 2 [19]. Building blocks such as the AND-split, AND-join, OR-split, OR-join are used to model sequential, conditional, parallel and iterative control structures of workflows. AND-split and OR-split transition correspond to transitions with two or more output places, while AND-join and OR-join transition

correspond to transitions with multiple incoming arcs. Different symbols are attached to original rectangles to distinguish normal transitions from transitions containing branching conditions. Figure 2a illustrates a typical transition in a TDW, with execution duration ranging from d1 to d2; data elements a, b are inputs and c, d, e are outputs. The other parts of Figure 2 show how basic constructions of a workflow are represented by TDW's notations [19]. For the sake of simplicity, each activity is represented by a transition. Therefore, the terms 'activity' and 'transition' are interchangeably used in this paper.

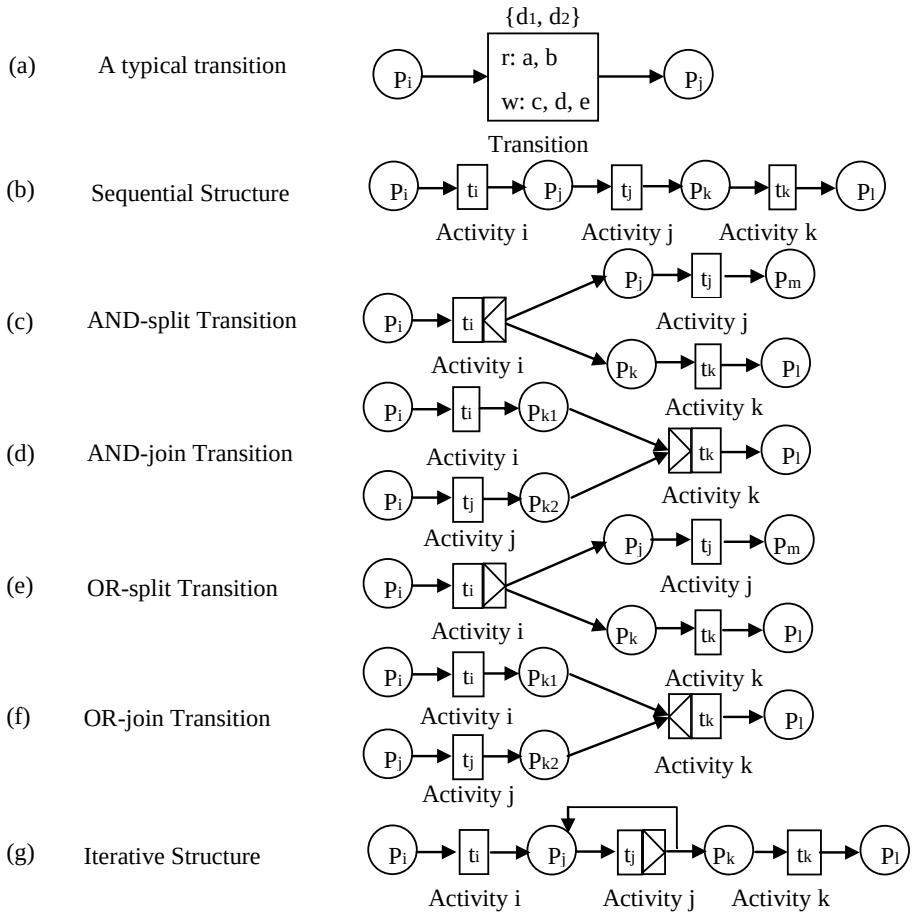


Fig. 2. Workflow primitives specified by TDW

As an extension of WF-Nets, TDW specifies the time and data properties of a single case in isolation, assuming that different cases are completely independent from each other. Therefore, UCIDs are caused by activities in a single TDW instance or activities belonging to workflow instances of different TDWs. Without the loss of generality, we assume that each TDW has one instance only.

Definition 1 (Time Data Workflow – TDW) A TDW, w, is a tuple <P, T, F, id, D, R, DE, TI > where:

- <P, T, F> is a WF-Nets with places P, transitions T and arcs F
- id is the workflow identifier.
- D is a set of data elements.
- R = {r, w, u} is a set of possible access rights to data elements (r: ***read***, w: ***write***, u: ***use*** (either ***read*** or ***write***)).
- DE: T x R$\rightarrow 2^D$ is a function that returns a set of data elements associated with a transition and an access right.
- TI: T $\rightarrow$ R^+ x (R^+ x ∞) is a time interval function that returns minimum and maximum execution durations of a transition.

Definition 2 (Concurrent Time Data Workflow Model) A Concurrent TDW Model cwm = (W, T_{wm}) is a collection of TDWs which have overlapping execution times (concurrent TDWs):

- W = {w_1, w_2… w_n} is a set of concurrent TDWs, where w_i = < P, T, F, id, D, R, DE, TI >.
- $T_{cwm} = T(w_1) \cup T(w_2) \cup \ldots \cup T(w_n)$ is the set of all transitions (activities) in cwm.

Given a TDW w as in Definition 1, we have the following definitions [19]:

Definition 3 (Path) A Path is a sequence of consecutive arcs.
A sequence p = (x_o, x_1, …, x_k) is a Path iff $\forall$ i, 0 < i < k – 1: (x_i, x_{i+1}) $\in$F

Definition 4 (Transition Path) A sequence p = (t_0, p_1, t_1,… , t_k) is a Transition Path iff it is a path and t_0, $t_k \in$ T.

Definition 5 (Transition Reachability) Transition t_i is reachable from t_j if there exists a transition path (t_i,... , t_j) on wm.
Reachable (t_i, t_j) = true iff $\exists$ transition path p = (t_i,... , t_j)

Definition 6 (Transition Distance) Given two transitions t_i, t_j where Reachable (t_i, t_j) = true or where Reachable (t_j, t_i) = true, the Transition Distance between t_i, t_j is the length of the shortest path between them.

Definition 7 (Nearest Common Transition) Given two transitions t_i, t_j where Reachable (t_i, t_j) = false and where Reachable (t_j, t_i) = false, their Nearest Common Transition is the common transition which has the shortest distances to both of them, denoted as t_{nct}.

Definition 8 (Closest Data Relation Transition) Given two transitions t_i, t_j, where their nearest common transition is not an OR-split transition, t_j is called the Closest Data Relation Transition of t_i on data element d if t_j just precedes t_i in terms of time, and both t_j and t_i *use* (*read*/*write*) d, denoted as t_{cdrt}.

4 UCIDs in a Concurrent TDW Management System

A Concurrent TDW Management System is a workflow management system which is responsible for TDW construction and management. A module of UCID detection and correction is also integrated into this system.

Data flow can be implemented explicitly as a part of the workflow model by using a separate channel to pass data from one activity to another. Otherwise, it can also be implemented implicitly through a control flow or process data store [3]. The process data store is basically a central repository where all workflows' activities can access or update their data. We choose implicit data flow through the process data store as a basis for our approach. In this implementation model, UCID may occur, particularly in cases involving concurrent execution paths.

Given a Concurrent TDW Model cwm as in Definition 2, we have the following definitions:

Definition 9 (Data Relation) Two activities a_i, a_j ($i \neq j$) have data relation if $DE(a_i, u) \cap DE(a_j, u) \neq \emptyset$ [19].

Definition 10 (Concurrent activities) Two activities are called concurrent activities iff they belong to two parallel branches of a TDW or they are in different TDWs and have overlapping Active Intervals.

Definition 11 (Unintentional Change in In-use Data) A situation in which some data values are lost or some data elements are assigned values different from the intentions of workflow designers due to non-deterministic access to shared data by different activities [19].

Here we distinguish two kinds of UCID: intra-UCID and inter-UCID. The former considers UCID situations concerning concurrent activities in the same workflow, while the latter is related to concurrent activities in different workflows. Definition 12, 13 are based on definitions of read-write conflict and write-write conflict in [1].

Definition 12 (RW Intra-UCID) A situation in which an activity A tries to ***read*** data from a shared variable x and an activity B tries to ***write*** data to the same shared variable *x* and vice versa, where A, B are concurrent activities in the same workflow.

Definition 13 (WW Intra-UCID) A situation in which two concurrent activities in the same workflow, A and B, try to ***write*** data to the same shared variable.

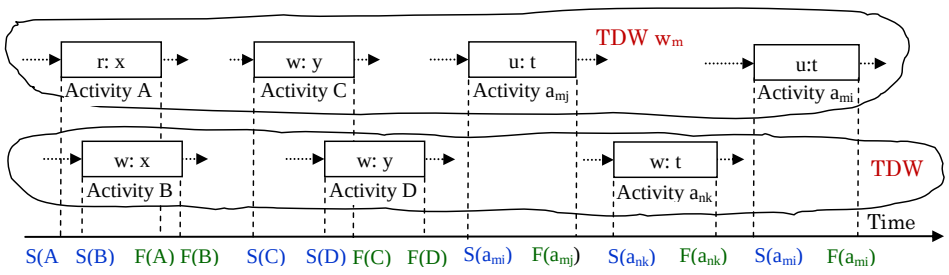


Fig. 3. Inter-UCIDs

Definition 14 (RW Inter-UCID) A situation in which an activity A tries to ***read*** data from a shared variable x and an activity B tries to ***write*** data to the same shared variable *x* and vice versa, where A, B are in different concurrent workflows and have overlapping Active Interval ($[S(A), F(A)] \cap [S(B), F(B)] \neq \emptyset$).

Definition 15 (WW Inter-UCID) A situation in which two activities A and B try to ***write*** data to the same shared variable, where A, B are in different concurrent workflows and have overlapping Active Interval ($[S(A), F(A)] \cap [S(B), F(B)] \neq \emptyset$).

Definition 16 (UWU Inter-UCID) A situation in which there are inconsistent views of shared data by two activities in the same workflow, because their shared data are ***written*** externally by an activity in a different concurrent workflow.

As depicted in Figure 3, two activities a_{mi}, a_{mj} of TDW w_m ***use*** (***read*** or ***write***) data element t, where a_{mj} is the closest to a_{mi} in terms of time and $F(a_{mj}) < S(a_{mi})$, which means $t_{cdrt}(a_{mi}, t) = a_{mj}$. A UWU Inter-UCID happens because activity a_{nk} of a different workflow w_n ***writes*** to t within the time interval $[F(a_{mj}), S(a_{mi})]$. RW Inter-UCID and WW Inter-UCID also happen between activity A and activity B, activity C and activity D respectively.

5 Detection of Potential UCID in a Concurrent TDW Management System

Regarding UCID definitions, inter-UCIDs are identified based on the Active Interval of activities having data relation. However, Active Interval of an activity can only be determined at runtime when it has finished its execution, and hence Estimated Active Interval is used instead of Active Interval to find potential UCID at build time, before a new TDW is put into the Concurrent TDW Management System to start.

5.1 Calculation of Estimated Active Interval [19]

Designating the start time of a TDW w as a reference point, P_w, we can infer the Estimated Active Interval of an activity A [EST(A), LFT(A)] with respect to its minimum and maximum executing durations {min(A), max(A)} and basic control structures.

Let us say that A_s is the Start activity of a TDW w, then we have $EST(A_s) = P_w$ and $LFT(A_s) = P_w + max(A_s)$. For executing activity A, EST(A) = S(A) and LFT(A) = F(A) if A has been completed.

Sequential Connection (Figure 2b)

$EST(A_j) = EST(A_i) + min(A_i)$; $LFT(A_j) = LFT(A_i) + max(A_j)$

AND-Split Connection (Figure 2c)

$EST(A_j) = EST(A_i) + min(A_i)$; $LFT(A_j) = LFT(A_i) + max(A_j)$

$EST(A_k) = EST(A_i) + min(A_i)$; $LFT(A_k) = LFT(A_i) + max(A_k)$

AND-joint Connection (Figure 2d)

$EST(A_k) = MAX\{EST(A_i) + min(A_i); EST(A_j) + min(A_j)\}$

$LFT(A_k) = MAX\{ LFT(A_i), LFT(A_j)\} + max(A_k)$

OR-Split Connection (Figure 2e)

$EST(A_j) = EST(A_i) + min(A_i)$; $LFT(A_j) = LFT(A_i) + max(A_j)$

$EST(A_k) = EST(A_i) + min(A_i)$; $LFT(A_k) = LFT(A_i) + max(A_k)$

OR-joint Connection (Figure 2f)

$EST(A_k) = MIN\{EST(A_i) + min(A_i); EST(A_j) + min(A_j)\}$

$LFT(A_k) = MAX\{ LFT(A_i), LFT(A_j)\} + max(A_k)$

5.2 Potential UCID Detection Algorithm

Given a Concurrent TDW Model cwm = (W, T_{cwm}), where W = {w_1, w_2, …, w_k} and $T_{cwm} = T(w_1) \cup T(w_2) \cup \ldots \cup T(w_k)$, w = <P, T, F, id, D, R, DE, TI>. The main idea of this algorithm is to select one activity and compare it with the other activities. If two activities have data relation, we will check if there is a potential UCID. In the case of concurrent activities in the same workflow, potential intra-UCIDs can be detected with respect to Definitions 12, 13. If two compared activities are in different workflows and have overlapping Estimated Time Intervals, there is a possibility of an RW/WW inter-UCID occurrence (Definitions 14, 15). If only the data relation exists and one activity occurs before the other, we will compare this situation with the definition 16 and the pattern in Figure 3 to find out a potential UWU inter-UCID.

Step 1: Initialization:

1.1 Let *S* be a set of unchecked activities. S is initialized with all unfinished activities of T_{cwm};

1.2 Calculate Estimated Active Interval for all activities in S;

1.3 flag = TRUE is a Boolean variable*;*

Step 2: For every pairwise of activities (a_{mi}, a_{nk}) in S, execute the following steps:

2.1 Check their Data Relation

Let U_{mnik} be the set of shared data between a_{mi} and a_{nk}: $U_{mnik} = DE(a_{mi}, u) \cap DE(a_{nk},u)$;

2.1.1 If $U_{mnik} = \emptyset$, a_{mi} and a_{nk} do not have data relation. Therefore UCID cannot happen between a_{mi} and a_{nk};

2.1.2 If $U_{mnik} \neq \emptyset$, a_{mi} and a_{nk} have data relation. Take the next step.

2.2 If a_{mi} and a_{nk} in the same workflow, check intra-UCID possibility. Otherwise, check inter-UCID possibility;

2.2 Check intra-UCID possibility

2.2.1 If a_{mi} and a_{nk} belong to two parallel branches of a workflow, this means that their Nearest Common Transition, denoted as t_{nct} (a_{mi}, a_{nk}), is an AND-split transition, they are concurrent activities. Take the next step;

2.2.2 For every data element, denoted as d_{mnikl}, in U_{mnik}, check the access right to d_{mnikl} of a_{mi} and a_{nk}:

2.2.2.1 If both of them have *write* access right to d_{mnikl}, this means that $d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, w)$, then flag = FALSE. There is a potential WW Intra-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.2.2.2 If one activity has *write* access right to d_{mnikl} and the other has *read* access right to d_{mnikl}, this means that ($d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, r)$) or ($d_{mnikl} \in DE(a_{mi}, r)$ and $d_{mnikl} \in DE(a_{nk}, w)$), then flag = FALSE. There is a potential RW Intra-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.3 Check inter-UCID possibility /* Figure 3***/**

2.3.1 If a_{mi} and a_{nk} have overlapping Estimated Active Interval, this means that $[EST(a_{mi}), LFT(a_{mi})] \cap [EST(a_{nk}), LFT(a_{nk})] \neq \emptyset$, they are potential concurrent activities: check RW/WW inter-UCID possibility. Otherwise, check UWU inter-UCID possibility;

2.3.2 Check potential RW/WW inter-UCID

For every data element, denoted as d_{mnikl}, in U_{mnik}, check the access right to d_{mnikl} of a_{mi} and a_{nk}:

2.3.2.1 If both of them have *write* access right to d_{mnikl}, this means that $d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, w)$, then flag = FALSE. There is a potential WW Inter-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.3.2.2 If one activity has *write* access right to d_{mnikl} and the other has *read* access right to d_{mnikl}, this means that $(d_{mnikl} \in DE(a_{mi}, w)$ and $d_{mnikl} \in DE(a_{nk}, r))$ or $(d_{mnikl} \in DE(a_{mi}, r)$ and $d_{mnikl} \in DE(a_{nk}, w))$, then flag = FALSE. There is a potential RW Inter-UCID between a_{mi}, a_{nk} on d_{mnikl};

2.3.3 Check potential UWU inter-UCID

Assume that $LFT(a_{nk}) < EST(a_{mi})$. For each data element, denoted as d_{mnikl}, in U_{mnik} where a_{nk} has *write* access right to d_{mnikl}: $d_{mnikl} \in DE(a_{nk}, w)$, perform the following steps:

2.3.3.1 Find out the Closest Data Relation Transition of a_{mi} on d_{mnikl}, denoted as a_{mj}: $a_{mj} = t_{cdrt}(a_{mi}, d_{mnikl})$. If $a_{mj} = \emptyset$, UWU inter-UCID may not happen;

2.3.3.2 If $[EST(a_{nk}), LFT(a_{nk})] \subset [LFT(a_{mj}), EST(a_{mi})]$, then flag = FALSE. There is a potential UWU Inter-UCID among a_{mi}, a_{mj}, a_{nk} on d_{mnikl};

Step 3: Return flag.

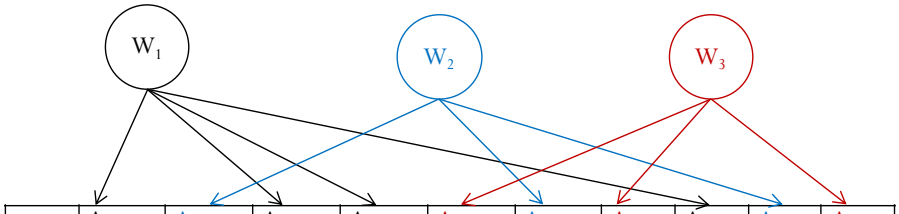
5.3 Algorithm Evaluation

Let's say *n* is the number of unfinished activities in a Concurrent TDW Model *cwm*. In general, we must inspect n^2 combinations of any two unfinished activities to find out some potential UCIDs. This approach allows us to detect not only potential UCID at build time of pre-executed TDWs, but also potential UCID at run time of running TDWs by recalculating the Estimated Active Intervals of their unfinished activities more accurately based on the Active Interval of finished activities. However, depending on applications, we can reduce the number of checking steps by considering some of the following heuristics:

- A two dimensional table can be used to record the access right on data elements of activities in a Concurrent TDW Model *cwm*. Figure 4 describes an example of data flow matrix of a Concurrent TDW Model with three TDWs W_1, W_2 and W_3. $\{D_1,\ldots, D_{10}\}$ is the data set of the Concurrent TDW Model. Parallelization can be applied here to reduce execution time. For each element in the data set of *cwm*, there is a thread being responsible for checking potential UCID caused by activities *using* this data element.
- After designing a new TDW, UCID check is conducted to find potential UCIDs before this TDW is put into the Concurrent TDW Management System for execution. Let's say *m, k, l* are the number of unfinished activities in the being considered pre-executed TDW, other pre-executed TDWs, running TDWs respectively, we have $n = m + k + l$. Because the other pre-executed TDWs have been checked in previous examinations, we can skip combinations of two activities in these TDWs to reduce the number of inspected combination to $n^2 - k^2$. If we just want to detect UCIDs caused by activities in the being considered TDW, we will

verify m x n activity combinations only. A parallel solution in this case is to create m threads. Each thread will be responsible for one activity in this TDW and will verify potential UCIDs on combinations created by this activity with the others in different TDWs.

– Because potential UCIDs just occur in activities that have shared data, we will verify activities having shared data only. Each data element will store identifications of unfinished activities using it. Therefore, the set of checked activities can be limited to unfinished activities having data relation in the Concurrent TDW Model. If the number of data elements is small, we can start from data elements of the being considered pre-executed TDW to pick out unfinished activities in the Concurrent TDW Model having data relations and use UCID patterns to find out potential errors.



	A_{11}	A_{21}	A_{12}	A_{13}	A_{31}	A_{22}	A_{32}	A_{14}	A_{23}	A_{33}
D1	W		R					R		
D2			R							
D3			W	R						
D4				W	R	R		R		
D5								W	R	
D6		W			R					
D7					W	R	W		R	R
D8									W	R
D9						W	R			
D10										W

Fig. 4. Data flow matrix example

6 Potential UCID Resolution

In general, if potential UCIDs happen, there may be some abnormalities in data flows or control flows of the concerned workflows. A review on the workflow design should be conducted to make sure that this situation is not made on purpose.

Our given solutions in which some of them will change the workflow structure are simply reference models. The final decision will depend on workflow designers to perform modifications that actually lead to a resolved model.

6.1 Potential Intra-UCID Resolution

Potential Intra-UCID may be caused by a mistake of workflow designers in designing parallel branches of a workflow. Therefore, our solution for Intra-UCID is to change the workflow structure by sequentializing or combining error-related activities. Two activities causing potential WW Intra-UCID are merged into one by place/transition fusion (Figure 5a). For RW Intra-UCID, sequentialization is applied to the related activities. One option is that read activity happens before write activity and the other is that write activity happens before read activity (Figure 5b). Resolution order will begin from WW Intra-UCID cases to RW Intra-UCID cases. With regard to potential UCIDs belonging to the same group, the priority is the happening order.

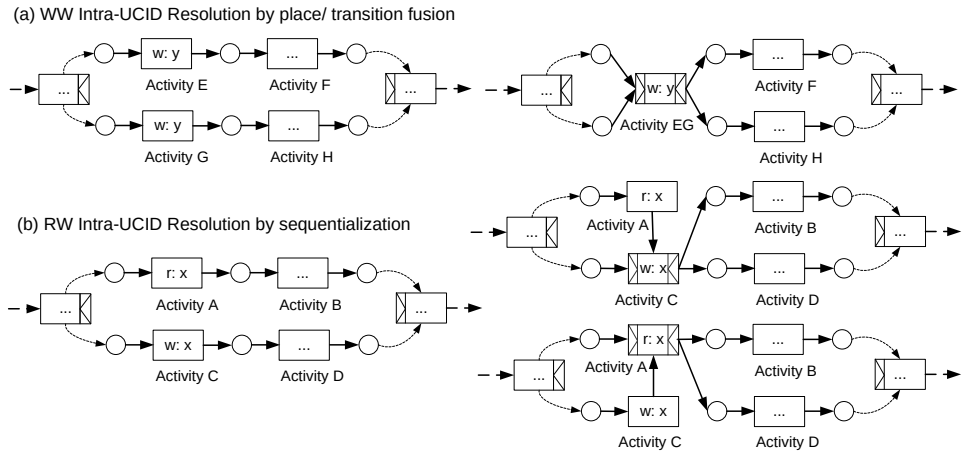


Fig. 5. Potential Intra-UCID resolution

6.2 Potential Inter-UCID Resolution

Resolving potential inter-UCID is more complex because workflows are designed for different purposes by different designers and a designer may know nothing about the work of the others. To resolve inter-UCID, the cooperation of different designers is necessary and the result will highly depend on the willingness of designers to communicate with each other.

A method which does not affect the workflow structures is to adjust the workflow schedule by modifying the workflow start time, maximum and minimum execution durations of activities in workflows so that inter-UCID patterns do not occur. Another solution is to change the workflow structure.

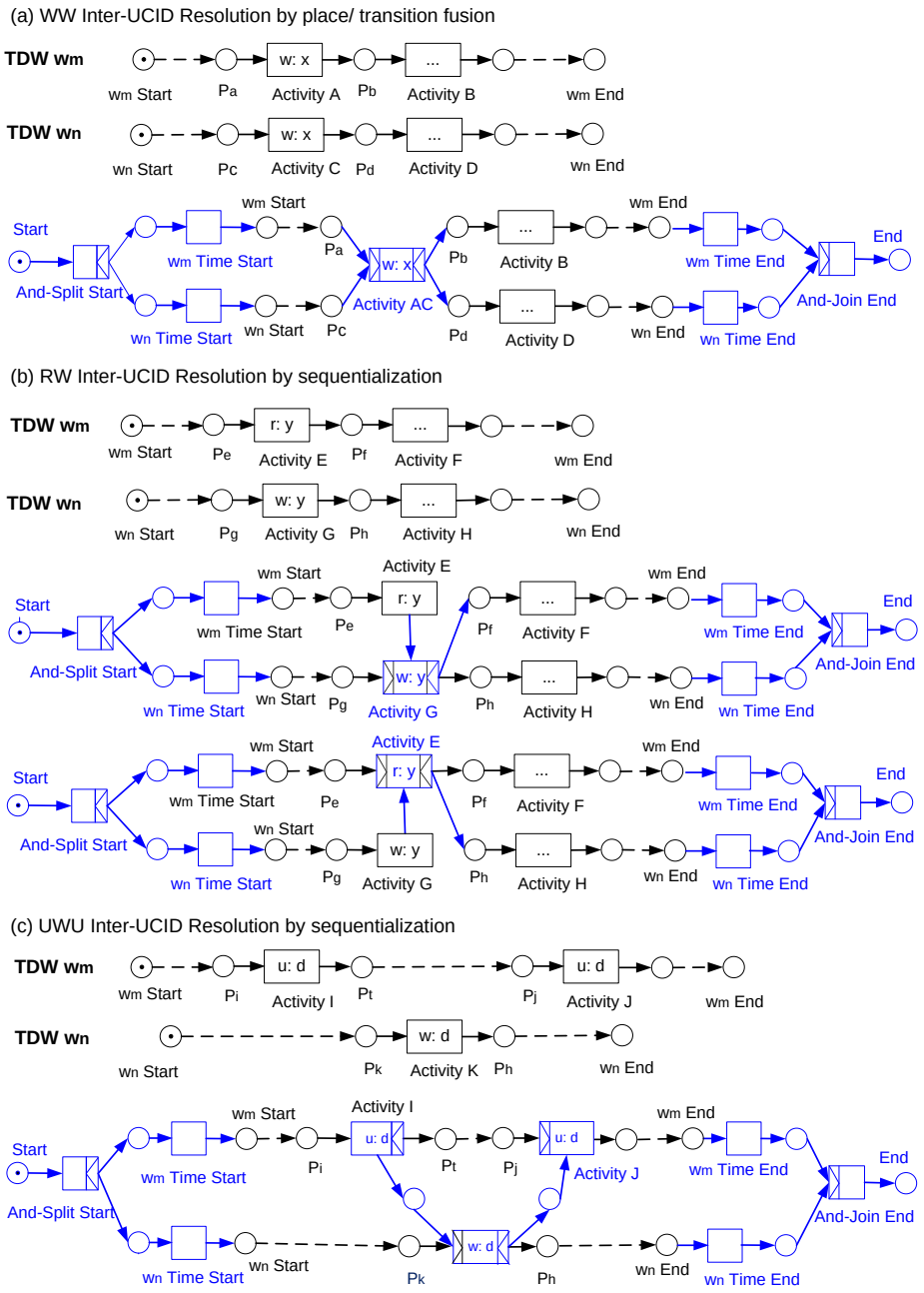


Fig. 6. Potential Inter-UCID resolution

First, we will combine related TDWs into one workflow. In order to preserve the structure of the original TDWs, in the new TDW, the Start place connects to an AND-Split transition and the End place is connected to an AND-join transition. Each merged

TDW corresponds to a subnet starting from the AND-split transition and ending at the AND-join transition. Because the merged TDWs are started at different times, we insert a Time Start transition between the Start place of each merged TDW and the AND-split transition, a Time End transition between the End place of each merged TDW and the END-join transition. Time activities are just null activities with some duration and they help to merge TDWs without modifying the workflow's schedule seriously. The AND-split transitions, AND-join transitions, Time Start transitions, Time End transition, places and arcs connecting the related workflows together represent the dependency relationships between different workflows which play an important role in the recovery process in the case of workflow failure. They will not be used to identify the total order of activities in detecting potential intra-UCID in the synthesis TDW. In the case of a running TDW, we can create a new TDW from the original workflow by removing its finished activities, and this new TDW will be combined with other TDWs in a normal way. Another simpler way is to combine the pre-executed TDWs only. After that, workflow designers can adjust the Estimated Active Interval of activities in the new TDW by modifying workflow start time, maximum and minimum execution duration of its activities so that UCID related activities happen after related activities of the running TDW.

Next, we will deal with activities causing potential Inter-UCID. The mechanism to handle potential WW/RW Intra-UCID is applied to WW/RW Inter-UCID cases (Figure 6a, 6b). Regarding UWU potential UCID, three activities related to this error are connected as shown in Figure 6c. If there are many potential Inter-UCIDs between the same two TDWs, the priority is Inter-UCID types (WW > RW > UWU) and occurring time of activities respectively.

As mentioned earlier, inter-UCID resolution is very complex, especially UWU inter-UCID. Currently, our proposed solution is just a reference model which helps workflow managers to have a more comprehensive view of data related workflows. We will try to improve them in the future work.

7 Application

In this section, we present a project on building a change support environment for cooperative software development. UCID theory is used in this project to detect potential UCID between concurrent workflows.

Software systems must be changed under various circumstances during development and after delivery, such as for new requirement, error correction, performance improvement, etc. However, software change is not an easy task, especially in a cooperative environment where software artifacts with very complex dependency relationships are created based on the cooperation of many people. Besides, other problems such as concurrency of works, synchronization of changes on shared artifacts, etc. also make this task more difficult. Therefore, a change support environment is strongly demanded.

In order to help change workers to perform change activities safety and efficiently in a cooperative environment, we use workflow to represent activities needed to implement

a change request. We define Change Support Workflow (CSW) as a sequence of activities required to implement a change. Activities in CSW are responsible for creating new software artifacts or modifying exiting ones. This means that data elements of CSW are software artifacts which need to be read, modified or created in the change implementation process.

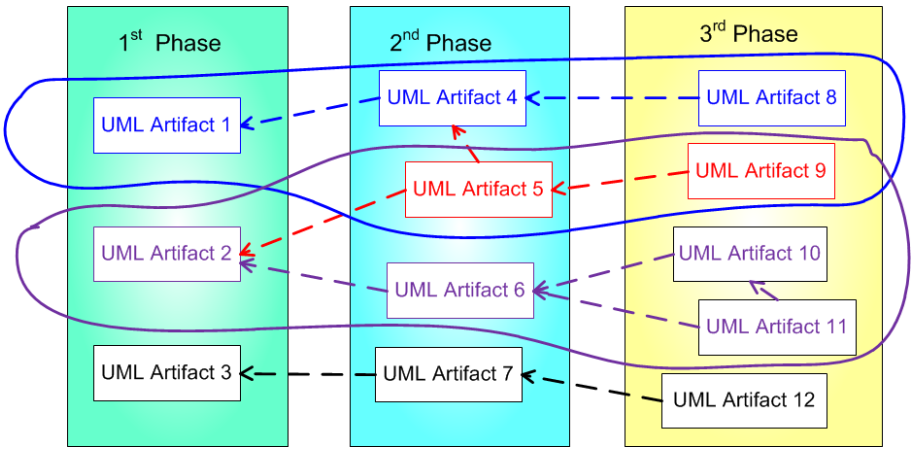


Fig. 7. Example of Relationships between UML Artifacts created during a software development process

In the first phase of the project, a method for automatically generating dependency relationships among UML elements was given [22]. Change impact analysis which identifies potential consequences of a change can be realized by tracing the generated dependency relationships. Result of this process will be used to generate CSW.

In large and cooperative system, there may be hundreds of CSWs executed at the same time to react to change requirements quickly. However, when there are many CSWs running on the same system, that UML artifacts are shared by different CSWs is unavoidable. If CSWs having shared artifacts are executed at the same time, inconsistencies among their data (UML artifacts) can happen. A version control system is used in our change support environment to deal with data loss; however this system does not help in this situation. Therefore, UCID theory is employed in this project to deal with this problem. Potential UCID can be detected automatically at build time to help workflow designers make timely adjustments to original workflows.

Our project supports constructing CSW based on the relationships between impacted UML model elements which are extracted from the result of impact analysis. CSW is modeled by TDW as follows. Each transition corresponds to an activity which creates or modifies at least one UML artifact. Total order of two transitions is identified by examining the dependency relationships between the artifacts modified by these transactions. Access role *write* is assigned to the artifacts which need to be modified or created; the artifacts for reference only are labeled with *read* access role. This draft of CSW will help workflow designers in developing the schedule of the change process.

The other steps in developing change schedule such as estimating activity resources and activity durations will be performed by workflow designers. From Activity Duration Estimates in the schedule, minimum and maximum execution durations of transitions in this CSW can be inferred. To reduce risks at runtime, UCID check on this CSW will be conducted. If some potential UCIDs are reported, data and control structure of this CSW should be adjusted in responding to suggested solutions of the change support system.

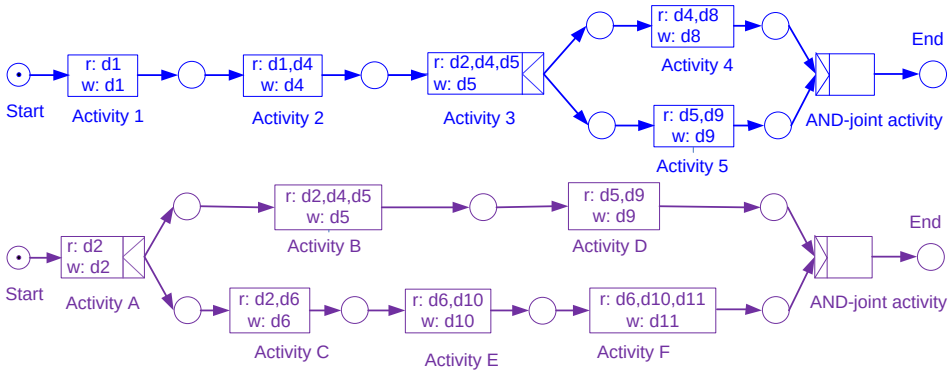


Fig. 8. Example of CSWs created based on the relationships between UML Artifacts

Table 1. Time aspect of activities in CSWs described in Figure 7

CSW ID	Start time P_w	Activity Name	Activity Duration Estimates (days)	Minimum and Maximum execution duration	Estimated Active Interval
W1	5	Activity 1	7.5 ± 0.5	{7,8}	[5,13]
		Activity 2	5.5 ± 0.5	{5,6}	[12,19]
		Activity 3	11 ± 1	{10,12}	[17,31]
		Activity 4	6 ± 1	{5,7}	[27,38]
		Activity 5	7 ± 1	{6,8}	[27,39]
		AND-joint	0	{0,0}	[33,39]
W2	15	Activity A	6 ± 1	{5,7}	[15,22]
		Activity B	5 ± 1	{4,6}	[20,28]
		Activity C	5 ± 1	{4,6}	[20,28]
		Activity D	10 ± 1	{9,11}	[24,39]
		Activity E	5.5 ± 0.5	{5,6}	[24,34]
		Activity F	6 ± 1	{5,7}	[29,41]
		AND-joint	0	{0,0}	[34,41]

Because CSW is constructed based on relationships between software artifacts, potential Intra-UCIDs seldom happen. Besides, if potential UCIDs are reported, the

possibility of control flow errors is low too. In this case, workflow designers should review data flow and pay attention to shared data elements among concurrent CSWs. With reference to potential inter-UCID, Estimated Active Intervals of activities play a very important role; therefore a change on project schedule may help overcome this error.

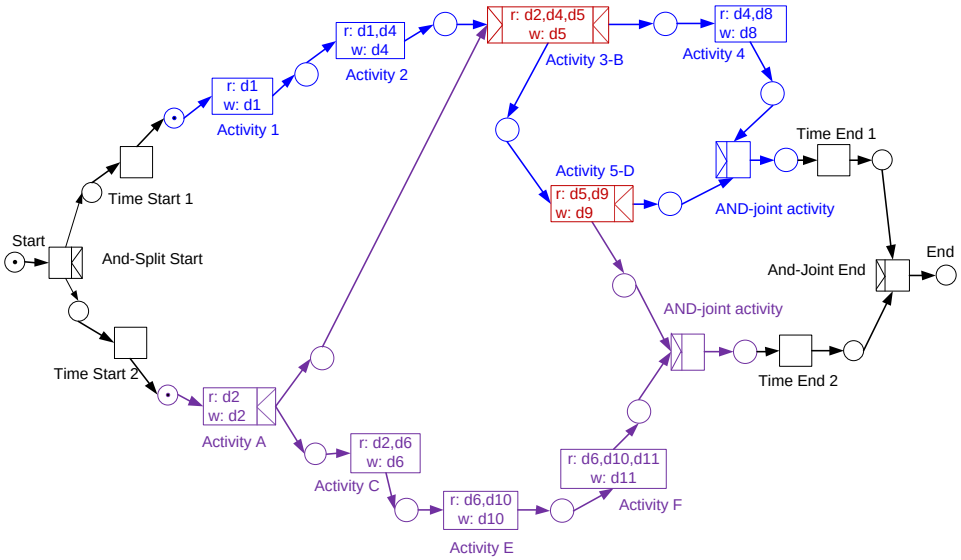


Fig. 9. Modified CSW with potential UCID corrected

Let's have an example. Figure 7 describes an example of relationships between UML artifacts created in different phases of a software development process. If we change UML Artifact 1, we need to change UML Artifacts 4, 5, 8, 9 because of the relationships between them. Similarly, if we change UML Artifact 2, we need to change UML Artifacts 5, 6, 9, 10, 11. By tracing the relationships starting from UML Artifact 1 and UML Artifact 2, we can create two CSWs to respond to change requirements on UML Artifact 1 and UML Artifact 2 respectively (Figure 8). Based on the generated workflows, project manager can conduct other steps in project time management such as estimating activity resources, estimating activity durations, etc. Information about activity duration is used to detect potential UCIDs. In Table 1, the minimum and maximum execution durations of each activity in CSWs described in Figure 8 are calculated from the Activity Duration Estimate, quantitative assessment of the likely number of work periods that will be required to complete an activity [18], of the corresponding activity in the project time management. Based on these values and the start time of the corresponding workflow, we can calculate the Estimated Active Intervals according to the formulas given in Section 5.1. After using the Inter-UCID detection algorithms, the following potential Inter-UCIDs are reported: WW Inter-UCID between activity 3 and activity B on artifact 5, WW Inter-UCID between activity 5 and activity D on artifact 9, RW Inter-UCID between activity 3 and activity A on artifact 2,

RW Inter-UCID between activity 5 and activity B on artifact 5. By applying the second Inter-UCID resolution method, modifying workflow structure, we get the synthesis CSW as described in Figure 9.

Because detecting potential UCIDs at build time is limited to workflows in which Estimated Active Intervals can be given before execution, solving this problem at runtime will be our next step. The model versioning system AMOR [21] offers some methods to resolve collaborative conflict in model versioning. Regarding this approach, all people who performed the changes are involved in eliminating the conflicts to obtain one consistent model version. We will consider applying this approach in our environment to increase the flexibility of the system.

8 Related Work

Workflow verification has attracted a lot of attention, especially control flow aspect. However, little research has been carried out on data verification in the workflow literature.

Reference [3] was one of the first studies to mention the importance of data-flow verification, and identified possible errors in the data-flow, like missing data, redundant data, conflict data, etc. Some general discussions on data flow modeling, specifications and verifications have been given, but without any detailed solution. The authors in [12] used data flow matrix and UML activity diagram to specify data flow. Based on this specification, an algorithm for detection of some data anomalies, such as missing data, redundant data, and potential data conflicts, was given [3]. In [11], a new workflow model, named Dual Workflow Nets, was defined to explicitly describe both control flow and data flow. A graph traversal approach was used in [10] to build an algorithm for detecting lost data, missing data and redundant data. More data flow errors were recognized and conceptualized as data flow anti-patterns and expressed in terms of temporal logic CTL* [5, 6]. By using temporal logic, available model checking techniques can be applied to discover these anti-patterns.

Nevertheless, all of these studies consider data flow errors in a single workflow only and no error removal method is given at all. In contrast to previous work, we address not only the interactions of concurrent activities inside a single workflow, but also the mutual influences between concurrent workflows, which are the sources of data flow errors. In [19], we focused on identifying UCID situations and defining a new workflow model as an extension of Petri Nets. Two algorithms for detecting intra-UCID and inter-UCID were also given in this work. However, there are still many unsolved problems in [19] and this paper is its refined and extended version. In this paper, TDW is defined as an extension of Workflow Nets (WF-Nets) instead of Petri Nets. Because the two algorithms in [19] had many common steps, if we use them separately, execution cost would be high. Therefore, these two algorithms are combined to reduce the cost and to form a more accurate and useful algorithm. Algorithm evaluation is also included in this version. Besides, some heuristics are provided to make the algorithm more flexible and effective. After that, some UCID resolution methods are proposed to help remove UCID

errors. Finally, building a change support environment for cooperative software development is introduced as an application domain for our work.

Concerning the mutual influences of the concurrent workflows approach, the research closest to us is [7]. However [7] addressed the verification of workflow resource constraints, and in this work, by nature, handling the resource problem is simpler than the data problem. A Time Constraint Workflow Net was defined to model workflow. Then, they identified the problem of resource constraints in WFMS and proposed a pseudocode algorithm which checked the resource dependency between every two activities. Reference [4] used hybrid automata to model the influences between concurrent workflows, and adopted a model checking technique to detect resource conflict problems.

9 Conclusion and Future Work

In this paper, we have presented ***Unintentional Change in In-use Data*** *(UCID)* concept and classified types of UCID which can occur, between activities in a single workflow or in different concurrent workflows. We have also proposed a Time Data Workflow based on the WF-Nets with many attributes supporting UCID estimation. An algorithm which helps detect intra/inter-UCIDs in a Concurrent TDW Management System has been developed too. After that, algorithms evaluation and some solutions to resolve UCID problem are given. Finally, we have introduced a concrete project supporting software change development process in a cooperative software environment as an application using UCID theory to verify change processes at build time.

As future work, we will implement a prototype of Concurrent TDW Management System and evaluate the effectiveness of UCID detection algorithm by runtime analysis. Then, we will improve inter-UCID resolutions and refine the generated TDW after applying UCID resolution methods in the Concurrent TDW Management System. Detecting and correcting UCID at runtime are our next targets. We also plan to investigate formal verification methods to verify the correctness of our model and method. Finally, we will integrate our system into the open source WoPeD [17]. Another direction of our research is to extend the TDW and improve UCID detection algorithms to address errors in resource and access control constraints.

References

1. Lee, M., Han, D., Shim, J.: Set-based access conflicts analysis of concurrent workflow definition. In: Proceedings of Third International Symposium on Cooperative Database Systems and Applications, pp. 189--196. Beijing, China (2001)
2. Li, H., Yang, Y., and Chen, T. Y.: Resource constraints analysis of workflow specifications. J. Syst. Softw. 73, 2, pp. 271--285 (2004)
3. Sadiq, S., M. Orlowska, W. Sadiq and C. Foulger.: Data flow and validation in workflow modeling. In: Proceedings of 15th Australasian Database Conference. LI, H. pp. 207--214 (2004)

4. Kikuchi S., Tsuchiya S., Adachi M., and Katsuyama T.: Constraint Verification for Concurrent System Management Workflows Sharing Resources. In: Third International Conference on Autonomic and Autonomous Systems (2007)
5. Trˇcka N., van der Aalst W.M.P., and Sidorova N.: Analyzing Control-Flow and Data-Flow in Workfow Processes in a Unified Way. Technical Report CS 08/31, Eindhoven University of Technology (2008)
6. Trˇcka N., van der Aalst W.M.P., and Sidorova N.: Data-Flow Anti- Patterns: Discovering Data-Flow Errors in Workflows. In: 21st International Conference on Advanced Information Systems (CAiSE'09). LNCS, vol. 5565, pp. 425--439. Springer-Verlag Berlin Heidelberg (2009)
7. Zhong, J. and Song, B.: Verification of resource constraints for concurrent workflows. In: Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 253--261 (2005)
8. Wil van der Aalst, Kees Max van Hee: Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA (2004)
9. Zeng, Q., Wang, H. and Xu, D: Conflict detection and resolution for workflows constrained by resources and non-determined duration. Journal of Systems and Software 81(9), pp 1491--1504 (2008)
10. Sundari M.H., Sen A.K., and Bagchi A.: Detecting Data Flow Errors in Work-flows: A Systematic Graph Traversal Approach. In: 17th Workshop on Information Technology & Systems (WITS-2007). Montreal (2007)
11. Fan S., Dou W.C., and Chen J.: Dual Workflow Nets: Mixed Control/Data-Flow Representation for Workflow Modeling and Verification. In: Advances in Web and Network Technologies, and Information Management (APWeb/WAIM 2007Workshops), LNCS, vol. 4537, pp 433--444. Springer-Verlag, Berlin (2007)
12. Sun S.X., Zhao J.L., Nunamaker J.F., and Liu Sheng O.R.: Formulating the Data Flow Perspective for Business Process Management. Information Systems Research, 17(4), pp 374--391 (2006)
13. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: Proceedings of the 17th International Conference on Data Engineering (ICDE '01), pp. 243–252 (2001)
14. Workflow Patterns, http://www.workflowpatterns.com
15. Russell N., van der Aalst W.M.P., and ter Hofstede A.H.M.: Designing a Workfow System Using Coloured Petri Nets. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) III, 5800, pp 1--24 (2009)
16. Awad, A., Decker, G. and Lohmann, N.: Diagnosing and Repairing Data Anomalies in Process Models. In: 5th International Workshop on Business Process Design. LNBIP, pp 1--24. Springer, Heidelberg (2009)
17. Workflow Petri Net Designer, http://193.196.7.195:8080/woped
18. PMBOK Guide Fourth Edition. Project Management Institute (2008)
19. Phan Thi Thanh Huyen and Koichiro Ochimizu: Detection of Unintentional Change on In-use Data for Concurrent Workflows. In: Proceedings of the 2010 International Conference on Software Engineering Research and Practice (SERP 10). Las Vegas, Nevada, USA (2010)
20. Elmasri, R. and Navathe, S. B.: Fundamentals of database systems, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA (1989)
21. Adaptable Model Versioning, http://modelversioning.org/
22. Masayuki Kotani and Koichiro Ochimizu: Automatic Generation of Dependency Relationships between UML Elements for Change Impact Analysis. Journal of Information Processing Society of Japan, vol. 49, no.7, pp 2265—2291 (2008)