Error Detection of Structured Workflow Definition Using Set Constraint System

Jaeyong SHIM†, Minkyu LEE††, and Dongsoo HAN†, Nonmembers

SUMMARY An workflow definition containing errors might cause serious problems for an enterprise especially when it involves mission critical business processes or inter-organizational interaction. So workflow definitions should be defined in a strict and rigorous way. In this paper, we suggest a workflow definition language and analysis methods for the language to support strict and rigorous workflow definitions. Faults or mistakes like provoking communication deadlock, access conflicts and improper exception specification in workflow definitions can be detected and notified automatically using the methods. The workflow definition language borrows structured constructs of conventional programming languages because they have many good features in expressing workflow processes also. With slight modifications and scope restrictions, the proposed techniques can be used in any workflow definition languages and workflow designers can define workflow processes much more concretely and safely.

key words: Set Constraint System, Workflow Definition, Communication Deadlock, Resource Conflict, Exception

1. Introduction

A workflow automates procedures where documents, information or tasks need to be passed between participants according to a defined set of rules to achieve a business goal. An activity in workflow could be performed either by a human, device, program or another WfMS. A workflow definition specifies a set of steps, data and control flows between the activities and a workflow instance is a real case of a business process. That is, a workflow definition is a template and numerous workflow instances are generated from the template. Thus if a workflow definition comprises erroneous specifications, it can result in unintended execution of business processes. Especially, when the workflow is critical to an enterprise, failure or unintended execution of the workflow may bring serious problems. An incorrect part of workflow definition may cause long-lasting workflow instances to be invalid and lead to the loss of goodwill between trading partners participating in virtual business processes in the case of inter-organizational workflow.

An incorrect workflow definition generally results from the mistakes of workflow designers and lack of consideration for various situations that might be encountered at runtime. Thus it is desirable to provide automatic ways to detect faults in the workflow definition and to notify process designers of the errors at workflow build time.

There are a lot of workflow languages for single organizational workflow. WPDL[5], Valmont[1], WFSL[19], BPEL4WS[20], etc. belong to single organizational workflow language. And, for inter-organizational workflow definition, there are workflow languages, such like InterWorkflow[7], Contract Language[4], BPML[8], and so on. However, most of the workflow languages only consider how to expand expression power for specifying various situations in workflow definition without providing verification methods. Not so much work has been done on the research of workflow verifications and most of them are based on Petri-net and graph theory. Aalst[10] and Kindler[9] adopted Petri-net to find potential errors in the design of the single workflow and inter-organizational workflow respectively. Arpinar[11] used set theory and graph theory to formalize correctness criteria of workflow definition including concurrent composition. However, their analysis methods are intricate to understand and difficult to reason about their properties. In addition, it is almost impossible to apply the analysis frameworks to other problems without significant modification.

In this paper, we adopt Set Constraint System approach, which is developed in a programming language society, to detect specification errors in workflow definition. It gives simple and more intuitive and understandable analysis rather than the computational algorithms of other approaches. The uniform approximation framework of this method provides the possibility of stable and scalable analysis. This paper also defines a language, called Community Process Definition Language(CPDL), as a target language of our analysis technique. CPDL is capable of defining business process, inter-workflow interaction and exceptional situation. The structured constructs of CPDL prevent users from invalid definition at syntax-level and they are better for users to grasp the control flow of workflow definition than other workflow specification languages such as WPDL[5].

In this paper, three fundamental issues for correct workflow specification has been resolved using aforementioned set constraint system. The first one is to detect latent communication deadlock in an inter-organizational workflow definition. The second one is to analyze potential access conflict and the third one is to find exception declaration that cannot be handled.

The previous researches on workflow languages and verification methods are described in section 2 and the syntax and semantics of CPDL are in section 3. In
section 4, we explain the set constraint system to help understand our analysis techniques. Through section 5, section 6 and section 7, we show how to analyze the problems mentioned above respectively and they expose uniformity of our analysis techniques. Section 8 shows applicability of CPDL and section 9 draws conclusion.

2. Related Work

Because business processes are getting more diverse and complex, improvements on workflow specifications are going on. Much of the developments in the expressive power of workflow specifications have been established with workflow models for business requirements.

WPDL (Workflow Process Definition Language) [5], a standard for workflow specification, was proposed by WfMC. It consists of five perspectives of workflow definition, such as Workflow Process Definition, Activity Definition, Transition Information Definition, Participant Definition, Reivan Data Definition and Application Definition. Valmont [1] was introduced in order to improve understanding of the workflow paradigm, enable studying of interaction between different features, and facilitated the development of more advanced features. Unlike many other workflow languages, Valmont captures all the fundamental elements of the workflow paradigm at an abstraction level that is suitable for workflow application designers.

In recent years, workflow definition was introduced into the web environment. WSFL [19] has a goal to enable web services as implementations for activities of business processes. Each activity is associated with a service provider responsible for the execution of the process step. WSFL supports two types of composition and choreography such as, flow models: describes business processes and global models: describe overall partner interactions. BPEL4WS[20] superceding WSFL provided a language for the formal specification of business processes and business interaction protocols. The former is called executable business process describing actual behavior of a participant in a business interaction and the latter is abstract process that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior.

Contrary to the above single organizational workflow definition, several researches handled interorganizational workflow definitions. The InterWorkflow presented the InterWorkflow language[7] and implementation of it with experimental proof using WMC’s standard protocol. It assumes that interactions among WfMSs of organizations are predefined in one place at build time. CrossFlow[4] defined XML-based contract description language to specify data exchange between trading partners and it implemented the framework including from contract conceptual model to interaction between WfMSs. In e-commerce society, many XML-based languages have been developed for data exchange among organizations. BPML[8] of BPML.org is a XML-based language for coordination of collaborative business processes among trading partners. It involves exception handling, transactions and rich format to enable the different types of activities.

However, these studies did not consider the correctness of specification and did not provide any verification methods. Only a few papers explicitly focused on the problems of verifying the correctness of inter-organizational workflows. Aalst[12] and Kindler[9] extended the WF-net[10] derived from Petri-net to be able to verify inter-organizational workflow definition and presented the interaction between workflows in terms of message sequence charts. It focused on the control flow errors, such as isolated task, control flow deadlock and livelock. XRL[13] is also Petri-net based inter-organizational workflow language focusing on workflow instance for ad-hoc interoperation. It supports less structured and more dynamic process and provides formal methodology for detecting anomalies.

On the other hand, for verification of single organizational workflow definition, Arpinar[11] formalized components of workflow definition related to its correctness in the presence of concurrency. Set theory and graph theory are bases of the verification mechanism and it developed constraint based concurrency control techniques. Adam[15] proposed concrete verification procedures based on Petri-net and the target of them is checking the consistency of transactional workflow. Petri-net is quite probably most frequently used process modeling technique in workflow verification. WF-net[10] has some criteria demonstrating the correctness of workflow definition.

There have been many researches in static analysis for data flow verification of concurrent software. They also focused on detection of deadlock and access conflict (i.e. race-condition) problems. Most of them, such as flow graph method[21] and Petri-net method[22], are based on the concept of state enumeration using concurrency graph that represents all possible states a program may have during its execution. The main problem with them is that the concurrency graph includes large number of redundant states that are unessential on the error detection. Set based analysis of CPDL in this paper is also based on state enumeration. However, to specify inter-organizational workflow definition, CPDL provides syntactic instruments which make it possible to represent data exchanges between parallel workflow instances running on different runtime environments respectively and it is possible to present read-write variable sharing among activities executed concurrently within a workflow instance. As a result, these instruments of CPDL provide runtime information at build time and it gives a chance to form a Wait-For-Graph including data flow information. By the CPDL, an inter-organizational workflow definition
language, we are able to apply precise runtime information at static analysis and avoid state explosion of static analysis. That is the point of difference of CPDL analysis from other data-flow analysis methods.

3. Community Process Definition Language

We expand CPDL\cite{17} to involve exception declaration and in/out parameters for clear description for our error detection methods. It focuses on control flow perspective and in part data flow perspective of inter/intra-organizational workflow. The benefits of using structured constructs are obvious in guiding designers to define correct workflow expression.\cite{18}. As you can see in Figure 1, the structure of CPDL is the same as $D^* E^* W^*$. To say, $D$, corresponds to data declaration, can be placed many times and numbers of $E$, correspond to exception declaration, can be also placed. Subsequently, number of workflow definition $W$ can be defined. In this section, we explain each syntactic element detail.

3.1 Workflow

In defining a workflow schema, the following is the essential part of the definition. Tasks constituting the workflow and control flow presenting execution order among the tasks should be defined. Data flow presenting information interchanging among the tasks also needs to be defined. The semantics of each syntactic element are as follows:

- "0" is used to specify that there is no task to be activated. It is developed to keep the expressive power without losing the structured nature of CPDL.

- "task $t(p_1, \ldots, p_n, m_1, \ldots, m_n)\ \text{pre}\ (c : e)^* \ \text{post}\ (c : e)^*"" means the execution of a task named $t$. The task of CPDL involves manual tasks and automatic tasks. Sometimes, we can regard even a subflow as a task because a task of CPDL means only an execution step having in/out parameters and messages of workflow process and CPDL don’t care the types of activities or tasks.

The task may have zero or more parameters and messages from other workflow instances. Each parameter is either input parameter, denoted by $\text{in}$, or output parameter, denoted by $\text{out}$ and each message is either received message, marked by $\text{recv}(u, x)$\(\text{i.e.}\) before the task is started, it must receive message $x$ from task $u$, or sent message, marked by $\text{send}(v, x)$\(\text{i.e.}\) after the task is finished, it sends its output $x$ to task $v$. As a community process definition involves inter-organizational workflow interactions, message denotations present data interchange between tasks included in respective organizations. The task may also have pre-condition followed by $\text{pre}$ keyword and post-condition followed by $\text{post}$ keyword. The semantic of the execution can be described informally as follows. A WfMS checks the pre-condition of the task. If the condition $c$ is violated, the exception $e$ is raised and the later execution flow will be determined by the exception handler. After all pre-conditions are satisfied, the task must receive all the input messages from its interaction partners through the communication facilities provided by the WfMS and the task reads input parameters from shared database by pass-by-value manner. After the task evaluation, the post-condition is checked and processed in the same way as that of pre-condition. Receiving messages and input parameters block the execution of the task until all of them are ready but out parameters passing and sending messages are performed in asynchronous manner.

\[
P ::= D^* E^* W^* \quad (\text{program})
\]
\[
D ::= x \quad (\text{data declaration})
\]
\[
E ::= \text{exception} e \ [\text{isa} e] \quad (\text{exception declaration})
\]
\[
W ::= \text{workflow} / \text{begin} \ \text{w} \ \text{end} \quad (\text{workflow definition})
\]
\[
w ::= 0 \quad (\text{inert task})
\]
\[
(\text{w}) \quad (\text{priority})
\]
\[
\text{task} \ t(p^*; m^*) \ \text{pre}\ (c : e)^* \ \text{post}\ (c : e)^* \quad (\text{sequential composition})
\]
\[
w ; w \quad (\text{p-to-p composition})
\]
\[
\text{ifthen} \ w \ \text{else} \ w \ \text{end} \quad (\text{loop composition})
\]
\[
\text{while-do} \ w \ \text{end} \quad (\text{try block})
\]
\[
try \ w \ (\text{catch} e : h)^* \ \text{end} \quad (\text{catch block})
\]
\[
p ::= \text{in} x \quad (\text{input parameter})
\]
\[
\text{out} x \quad (\text{output parameter})
\]
\[
m ::= \text{recv}(u, x) \quad (\text{receive msg from} u)
\]
\[
\text{send}(v, x) \quad (\text{send msg to} v)
\]
\[
h ::= \text{retry} w \quad (\text{retry handler})
\]
\[
\text{resume} w \quad (\text{resume handler})
\]
\[
\text{terminate} w \quad (\text{terminate handler})
\]
\[
\text{break} w \quad (\text{break handler})
\]
\[
x \quad (\text{variable name})
\]
\[
\text{e} \quad (\text{exception name})
\]
\[
t \quad (\text{task name})
\]
\[
f \quad (\text{workflow name})
\]
\[
\text{c} \quad (\text{condition})
\]

Fig. 1 Abstract Syntax of CPDL

- "$w_0; w_1$" is a sequential composition that is the most essential control flow pattern of workflow and it specifies the dependency between consecutive task explicitly. It indicates that workflow process $w_1$ should be activated after completion of workflow $w_0$.

- "$w_0 || w_1$" is called concurrent composition used to specify that a single thread of control splits into two threads which are executed in parallel within a single workflow instance. It allows tasks in different threads to be executed concurrently and their order of execution is decided arbitrarily by the system situation. It is assumed that an AND-split\(\text{another name of concurrent composition}\) must be paired with another basic workflow pattern AND-join in CPDL. That is, two activated threads of workflow connected by a concurrent composition are converged into a single thread in synchronized manner. "$w_0 \& w_1$" means p-to-p composition. To enable CPDL to describe interaction among several workflows of different organizations, p-to-p composition operator is devised. Its semantic is very similar to concurrent composition but the split control threads by p-to-p composition are individual and local workflows that will be
running in different organizations respectively at runtime.

- “if-then $w_0$ else $w_1$”, called branch composition, corresponds to XOR-split pattern of workflow and it does the same role as the if-then-else control structure in programming language. To keep the language as structured form, like the case in the concurrent composition, a branch composition implicitly has its corresponding XOR-join pattern indicating the two alternative workflows get together without synchronization at the end of the branch composition.

- “while-do $w$” presenting iteration pattern of CPDL, called loop composition is a structured cycle containing only one entry point and one exit point. Many workflow models and commercial workflow systems support arbitrary loop allowing multiple entry and exit points but they can typically be converted into structured cycles connected by basic control flows.

- “()” is used only to bundle up a group of workflow expressions to modify the precedence relations between operators inside of the bundle and adjacencies to it. To clarify the semantic of control flow in community process definition, we define the precedence relations between pairs of operators as follow:

\[
	; \triangleright | \triangleleft \text{ if } b \trianglelefteq \text{ loop } b \triangleright \text{ }\]

where we use the notation $a \triangleleft b$ to specify $a$ yields precedence to $b$ and $a \trianglelefteq b$ for the precedence of $a$ and $b$ is the same. Besides, the sequence of equal precedence operators follows left-associative association rule.

- “try $w$ (catch $e : h$)” end” is used to catch and handle exceptions raised during execution of $w$. The raised exception is compared to $e$ described in catch clause and then if the exception is same as $e$ or sub-exception of $e$, that is handled by $h$.

### 3.2 Exceptions

Exceptions allow users to specify flexible control flow for unusual situation. CPDL provides elementary constructs for describing exceptional situations and handling mechanism. Generally, three things should be considered in specifying exceptions, such as structuring of exceptions, exception signaling and catching and handling exceptions.

In CPDL, exceptions to be used in workflow definition should be declared beforehand. There is inclusion relationships between exceptions, an exception including other exceptions is called super-exception and, conversely, an exception included in another exception is called sub-exception. This relationship is defined by isa keyword and its semantic is equal to “is-a” relation of object oriented system.

Exception handlers should be prepared for all exceptions that may be raised during the execution. If they would not be prepared, the workflow invoking unprepared exceptions could not proceed any more. In CPDL, four kinds of exception handlers are provided according to control flow after handling of exceptions.

- **retry $w$**: execute $w$ prepared to handle the exception and then retry the try-block.
- **resume $w$**: execute $w$ prepared to handle the exception and then resume the execution of the workflow. That is, continue the execution from the task which raised the exception.
- **terminate $w$**: execute $w$ prepared to handle the exception and then terminate the execution of the workflow.
- **break $w$**: execute $w$ prepared to handle the exception and then continue the execution from the first task followed by the try-block.

### 4. Set Constraint System

For detecting specification errors in a CPDL definition, we adopt Set Constraint System[3] which is a static analysis method used to approximate and analyze runtime properties of a program. A key feature of the set based analysis is that reasoning about a programs runtime behavior is reduced to reasoning about constraints on sets of program values. The constraints presented by set expression and containing partial run-time properties of a program are generated from the program text. The solution of set constraints yielded by resolving rules contains the desired information. It is helpful to generate CPDL definition avoiding redundant and superfluous computations.

Reasoning about a workflow definition by treating workflow variables as denoting sets of values leads to a simple, accurate and intuitively appealing notion of workflow approximation. This set based approach to workflow analysis consists of two phases as presented in Figure 2. The first phase constructs set constraints to approximate the behavior of the workflow. In the second phase, the constraints are solved to find their minimum solution indicating errors of workflow definition.
4.1 Construction of Set Constraints

A set constraint is an expression unit to denote runtime properties of workflow definition. Every syntactic element of CPDL has a set variable and each set variable has set constraints by construction phase and they are used to approximate the each runtime behavior of it. A set constraint is of the form $X \supseteq se$ where $X$ is a set variable and $se$ is a set expression. A set expression is either a set variable (denoted $X$, $\mathcal{Y}$, $\mathcal{Z}$, etc), or one of the forms $f(se_1, \ldots, se_n)$, $se_1 \cup se_2$ or $op(se_1, \ldots, se_n)$ where $f$ is a function symbol (data-constructor), $se_i$ is a set expression, and $op$ is a set operator. According to the problems we want to resolve in workflow definition, different set expressions are defined and they have their respective semantics. A solution to a collection of set constraints is an assignment of sets to set variables that satisfies each constraints.

In the construction of set constraints phase, set constraints are generated from the workflow definition by constraint generation rules. Generally, the generation rules are based on relationships between expressions of workflow language and constraints to be generated. The relationships generally constitute the rules as follow:

$$\text{subexpr}_1 \triangleright C_1 \ldots \text{subexpr}_n \triangleright C_n$$

$$\text{expr} \triangleright \{X \supseteq se_1, \ldots, X \supseteq se_m\} \cup C_1 \cup \cdots \cup C_n$$

The collection of constraints to be generated from the expression “expr” contains $X \supseteq se_1, \ldots, X \supseteq se_m$ and $C_1, \ldots, C_n$ that is generated by the subexpressions, such as $\text{subexpr}_1 \ldots \text{subexpr}_n$, of the “expr” respectively.

4.2 Resolving Set Constraints

The constraint solving is the second phase that finds the solution of constraints constructed in the first phase. Each of the constraint solving rules is of the form:

$$\frac{C_1 \ldots C_n}{N_1 \cdots N_m}$$

Using this form, one or more set constraints already contained are written above a bar ($C_1 \cdots C_n$) and new set constraints are written below the bar ($N_1 \cdots N_m$).

The structure states that if set constraints are found in written above a bar then add the new set constraints to the set of constraints. The minimum solution is computed by iterative application of constraint solving rules to set of constraints until no rule changes collection of constraints. This process is presented in Figure 3. The algorithm certainly terminates since the size of the collection of set constraints increases monotonously by the iterations and the size of the collection is limited by the number of $\binom{n}{2}$, where $n$ is the size of workflow expression.

A constraint is in explicit form if it has the form $X \supseteq ae$ where $ae$ is an atomic set expression that is not a set variable. If $C$ is a collection of constraints, then $\text{explicit}(C)$ denotes the each constraint in $C$ is in the explicit form. Furthermore, $\text{explicit}(C)$ determines the minimum solution of set constraint analysis that can be regarded as a concise collection having no unnecessary constraints.

```
input: a collection C of set constraints
repeat
  apply a solving rule if the rule generates new constraints
until no solving rule changes C
output: explicit(C)
```

Fig. 3 Set Constraint Solving Algorithm

The time complexity of the algorithm to estimate runtime behaviors of a program is $O(n^3)$ where $n$ is the size of input expression. The $O(n^3)$ bound is derived based on the following observations. First, the construction of constraints is proportional to the $n$. So the time complexity becomes $O(n)$. Second, at most $n^2$ new constraints can be added by the constraints solving algorithm, and the cost of “adding” each new constraint (i.e. determining what other new constraints need to be added, given this constraint is added) is bounded by $O(n)$. Thus, the sum of the first and the second phase becomes $O(n) + O(n^3) = O(n^3)$.

5. Latent Communication Deadlock Analysis

From this section, by analysis examples, we will explain how to detect errors in CPDL definition with Set Constraint System.

Several tasks of p-to-p composition in a CPDL definition are probably activated simultaneously on their respective WfMSs and might exchange messages with each other. At that situation, a task could be blocked if a task does not receive all input messages. When the dependencies among blocked tasks form a cycle, the workflow including the tasks is blocked permanently and we call this situation as communication deadlock[17]. In the case of workflow definition presented in Figure 4, workflow Org.A, Org.B, and Org.C are probably executed on different WfMSs simultaneously. The task groups, such as (task A1, task A2, task B1, task B2) and (task A3, task B3, task C2) will be in communication deadlock at runtime because they have cyclic dependencies caused by message sets, such as $\{x, y\}$ and $\{z, v, w\}$ respectively.

We defined set expressions which show runtime behaviors of workflow definition and they are necessary to detect communication deadlock in workflow definition. The notations, such as task(t), send(t, d, m) (i.e. task t sends message m to task d), receive(t, o, m) (i.e. task t waits for m from task o), p2p(x, y) (i.e. workflow
An p-to-p composition of three local workflows

Fig. 4

Constraint Generation Rules: $\triangleright_d$

$\text{[NULL]} : 0 \triangleright_d \phi$

$\text{[BRACE]} : \pi \triangleright_d \mathcal{C} \triangleright_d \mathcal{C}$

$\text{task} t (\text{rcv}(o_1, n_1), \ldots, \text{rcv}(o_n, n_n), \text{snd}(d_1, m_1), \ldots, \text{snd}(d_m, m_n))$

$\text{[TASK]} : X \triangleright_d \text{task}(t), X \triangleright_d X \triangleright_d X \triangleright_d \text{receive}(t, o_1, n_1), \ldots, \text{receive}(t, o_n, n_n), X \triangleright_d \text{send}(t, d_1, m_1), \ldots, \text{send}(t, d_m, m_n)$

$\text{[SEQ]} : w_0 \triangleright_d \mathcal{C}_0 \quad w_1 \triangleright_d \mathcal{C}_1$

$\text{[PAR]} : w_0 \parallel w_1 \triangleright_d \{X \triangleright_d X \triangleright_d X \triangleright_d X \triangleright_d X \triangleright_d X \triangleright_d \text{seq}(X_0, X_1) \cup \mathcal{C}_0 \cup \mathcal{C}_1$

$\text{[P2P]} : w_0 \triangleright_d \mathcal{C}_0 \quad w_1 \triangleright_d \mathcal{C}_1$

$\text{if-then} w_0 \text{ else } w_1 \triangleright_d \{X \triangleright_d X \triangleright_d X \triangleright_d \text{p2p}(X_0, X_1) \cup \mathcal{C}_0 \cup \mathcal{C}_1$

$\text{[IF]} : w_0 \triangleright_d \mathcal{C}_0 \quad w_1 \triangleright_d \mathcal{C}_1$

$\text{while-do} \quad w \triangleright_d \{X \triangleright_d X \}$

$w_i$ and $w_j$ connected by p2p composition of the p2p composition operator and $\text{seq}(X_i, X_j)$ (i.e., workflow $w_i$ and $w_j$ make up sequential composition) are constructed by syntax elements of CPDL at constraints generation phase. At set constraints solving phase, additionally, two set expressions are generated, such as $\text{arrow}(s, t)$ meaning that task $s$ waits for data from task $t$ or task $s$ is preceded by task $t$ and $\text{deadlock}(s, t)$ implying that task $s$ and $t$ wait for each other to be active. If we can find the set expression $\text{deadlock}(s, t)$ in result set of constraints, we can conclude that the workflow definition has a possibility of falling into permanently blocked state.

5.1 Construction of Set Constraints

Figure 5 illustrates the derivation rules, $\triangleright_d$, to generate set constraints for every expression of CPDL. The set variable $X$ is for the workflow to which the rule applies and the subscripted set variable $X_w$ is for the workflow $w$. The relation “$w \triangleright_d \mathcal{C}$” represents that “set constraints $\mathcal{C}$ are generated from workflow $w$.”

$X_1 \triangleright_d X_2 \quad X_2 \triangleright_d X_3 \quad X_1 \triangleright_d p2p(X_2, X_3)$

$X_2 \triangleright_d \text{receive}(C_2, A_3, w) \quad X_3 \triangleright_d \text{send}(C_2, B_3, v) \quad X_3 \triangleright_d \text{task}(C_2)$

$X_2 \triangleright_d \text{receive}(C_3, A_3, x) \quad X_3 \triangleright_d \text{send}(C_3, B_3, x) \quad X_3 \triangleright_d \text{task}(A_3)$

$X_2 \triangleright_d \text{receive}(C_3, A_3, y) \quad X_3 \triangleright_d \text{send}(C_3, B_3, y) \quad X_3 \triangleright_d \text{task}(C_3)$

$X_2 \triangleright_d \text{receive}(C_3, A_3, z) \quad X_3 \triangleright_d \text{send}(C_3, B_3, z) \quad X_3 \triangleright_d \text{task}(B_3)$

$X_2 \triangleright_d \text{receive}(C_3, A_3, w) \quad X_3 \triangleright_d \text{send}(C_3, B_3, w) \quad X_3 \triangleright_d \text{task}(A_3)$

$X_2 \triangleright_d \text{receive}(C_3, A_3, x) \quad X_3 \triangleright_d \text{send}(C_3, B_3, x) \quad X_3 \triangleright_d \text{task}(B_3)$

$X_2 \triangleright_d \text{receive}(C_3, A_3, y) \quad X_3 \triangleright_d \text{send}(C_3, B_3, y) \quad X_3 \triangleright_d \text{task}(C_3)$

$X_1 \triangleright_d \text{receive}(A_1, B_1, x) \quad X_2 \triangleright_d \text{receive}(A_1, B_1, y) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, z)$

$X_2 \triangleright_d \text{receive}(A_1, B_1, x) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, y) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, z)$

$X_2 \triangleright_d \text{receive}(A_1, B_1, x) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, y) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, z)$

$X_2 \triangleright_d \text{receive}(A_1, B_1, x) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, y) \quad X_3 \triangleright_d \text{receive}(A_1, B_1, z)$

Rule $\text{[TASK]}$ extracts information on task execution and data flow among tasks from the arguments declaration of task. It generates a set of corresponding set constraints, such as $X \triangleright_d \text{task}(t)$, $X \triangleright_d \text{receive}(t, o, m)$ and $X \triangleright_d \text{send}(t, d, m)$. Rules $\text{[P2P]}$ and $\text{[SEQ]}$ make set constraints $X \triangleright_d p2p(X_n, X_n')$ and $X \triangleright_d \text{seq}(X_n, X_n)$ respectively and inclusion relation between set variables. The other rules of constraints generation are for propagating set constraints to outer block. By this propagation rules, runtime properties...
implied in the set constraints of sub-expression can be held in the set variable of current expression.

During analysis, workflow definitions are recorded in a syntax tree, in which each node represents an operation as you can see in Figure 6. This syntax tree is equivalent to the example of Figure 4. Every node of syntax tree has its set variable and set constraints constructed by constraint generation rules. Set constraints generated from the syntax tree by applying constraint generation rules are presented in Figure 7.

5.2 Resolving Set Constraints

To solve the set constraints, we derive constraint solving rules \( S_d \) and Figure 8 illustrates them. \([\text{DIST}_{p2p}]\) means distribution rule of p2p-compositions, that is, \( X \) has two p2p-compositions of \((V, Z)\) and \((W, Z)\) if set variable \( X \) has a p2p-composition of \((V, Z)\) and set variable \( Y \) has a p2p-composition of \((V, W)\).

\( [\text{ARROW}_{\text{elu}}] \) presents explicit task dependence arising from control flow of tasks and \( [\text{ARROW}_{\text{da}}] \) shows implicit task dependence caused by data flow between tasks that belong to different local workflows. \( [\text{TRANSITIVE}_{\text{arrow}}] \) denotes the transitivity of task dependence relation and the other \( [\text{TRANSITIVE}] \)s propagate the constraints of current expression to super expressions. Rule \([\text{DEADLOCK}]\) finds a pair of tasks that are dependent on one another. The detected pairs of tasks are involved in latent communication deadlock.

The minimum solution is computed by iterative application of constraint solving rules \( S_d \) to set of constraints \( C \). This iterative application is denoted by \( S_d^\ast(C) \). The iteration continues until the results of consecutive iterations coincide. To get a more concise solution, it is useful to eliminate superfluous set constraints by using the function \( \text{explicit}(S_d^\ast(C)) \) and its semantics is defined as follows:

\[
\text{explicit}(S_d^\ast(C)) = \{ X \supset \text{deadlock}(s, t) \mid X \supset \text{deadlock}(s, t) \in S_d^\ast(C) \}
\]

In the case of our example, when \( C \) is a set of constraints of Figure 4, we can get the final solution of \( \text{explicit}(S_d^\ast(C)) \) as follows:

\[
\{ X_1 \supset \text{deadlock}(A_3, B_3), X_1 \supset \text{deadlock}(A_2, C_2), X_1 \supset \text{deadlock}(C_3, A_3), X_1 \supset \text{deadlock}(A_2, A_1), X_1 \supset \text{deadlock}(A_1, A_2), X_1 \supset \text{deadlock}(B_2, B_1), X_1 \supset \text{deadlock}(B_1, A_1) \}
\]

This solution indicates that the example CPDL process in Figure 4 has two groups of tasks that probably fall into communication deadlock at runtime. From the solution above, we can conclude that the two task groups, such as \( \{A_3, B_3, C_2\} \) and \( \{A_1, A_2, B_1, B_2\} \).

6. Access Conflicts Analysis

The activities in concurrent composition of CPDL may access the shared data in any order at runtime because their order of accessing data is dependent on runtime situation. But the non-deterministic access of concurrent activities to shared data may bring unexpected result. It is called race problem and it can be divided into two classes. The first one is \textit{read-write conflict} and the second one is \textit{write-write conflict}[18].

In workflow process definition presented in Figure 9, \( (\text{task } B(\text{out } x); \text{task } C(\text{in } x)) \) and \( (\text{task } D(); \text{task } E(\text{out } x); \text{task } F()) \) may be executed concurrently and they may access the shared variable \( x \). The first access conflict caused by \text{task } B and \text{task } E is \textit{write-write conflict}. The second access conflict is \textit{read-write conflict} caused by \text{task } C(\text{in } x) \) and \text{task } E(\text{out } x).

Set expressions constructed at constraint generation phase for resource conflict analysis are \( \text{task } R(t, x) \), \( \text{task } W(t, x) \), and \( \text{par}(X_{w_0}, X_{w_1}) \). The notation \( \text{task } R(t, x) \) means that \( t \) reads from the variable \( x \) and \( \text{task } W(t, x) \) means task \( t \) write to the variable \( x \). \( \text{par}(X_{w_0}, X_{w_1}) \) presents two workflows \( w_0 \) and \( w_1 \) are executed concurrently. At constraint solving phase, additional set expressions, such as \( \text{conflictRW}(s, t, x) \) and \( \text{conflictWV}(s, t, x) \), are generated by solving rules. The former presents read-write conflict between \text{task } s \) and \text{task } t \) and the latter shows write-write conflict.
6.1 Construction of Set Constraints

Figure 10 shows the rules to generate set constraints for every workflow expression. Like the case of communication deadlock analysis, the relation \( w \triangleright c \) represents that “constraints \( C \) are generated from workflow expression \( w \).”

Every workflow expression of workflow definition presented in Figure 9 is underlined and labeled. Each label will be used as subscript of its set variable. Because of the left-associative association of CPDL, in the point of constraints construction, the labeled workflow expression has the same operational meaning with the syntax tree in preceding section.

\[
\begin{align*}
\Delta_a : (B(out x), C(in x) \parallel D_f : E(out x) : F_f) : G_y : H_{x_{ah} y_h}
\end{align*}
\]

Set constraints for this example generated by \( \triangleright_c \) is presented in Figure 11 and the expected result this analysis is the minimal set which satisfies all the constraints.

6.2 Resolving Set Constraints

As you can see in Figure 12, we defined constraint solving rules \( S_c \) to solve the set constraints constructed in previous section.

The rules [TR-Rc], [TR-Wc], [TR-CRWc] and [TR-CWWc] are a kind of transitive rule such that if \( X \) includes \( Y \) and \( Y \) includes \( Z \), then \( X \) also includes \( Z \). [PAR1] and [PAR2] present commutativity and associativity properties of concurrent workflow threads, such as \( a || b = b || a \) and \( (a || b) || c = a || (b || c) \).

A read-write conflict situation is presented in the rule [CON-RW1] and [CON-RW2]. The meaning of the rules is that if two tasks are in parallel execution \( (par(Y, Z)) \) and one of the two reads from shared variable \( Y \) and the other writes to the shared variable \( Z \), then this situation is an read-write conflicts \( (X \subseteq conflictRW(s, t, x)) \). [CON-WWc] presents a write-write conflict situation like as [CON-RW1] and [CON-RW2]. We can obtain concise solution by applying \( explicit(S^*_c) \) to set of constraints \( C \) as follows.

\[
explicit(S^*_c) = \{ (X,e_0) \in S^*_c | e_0 = conflictRW(s, t, x) \} \cup \{ (X,e_0) \in S^*_c | e_0 = conflictWW(s, t, x) \}
\]

If \( C \) is same as Figure 11 then the final result is \( \{X_{ah} \subseteq conflictRW(C,E,x), X_{ah} \supseteq conflictWW(B,E,x)\} \).

7. Exception Analysis

The WfMS providing a means of defining exceptions and their handling routines confers flexibility on workflow definition[16][2]. It leads to reliable WfMS and great expression power on specifying business processes because unintended situations can be overcome by exception handlers without restarting the workflow instance. However, if workflow designers do not define all handlers for possible exceptional situations, the WfMS may go into undesirable state due to the unhandled exceptions.

In the workflow presented in Figure 13, the exceptions e0, e1, e2 and e3 are organized hierarchically. The task A() may raise exception e1 and the task B may raise exception e2 and e3 but the catchable exceptions are only e0 and e2. e1 can be caught by catch e0 since e1 is a sub-exception of e0. If exception e3 is raised, because the exception cannot be caught, we call it uncought exception and this fact should be notified to workflow designer.

In exception analysis, we define two set expressions
exception e0;
exception e1 isa e0;
exception e2;
exception e3;

workflow f begin
  try
    task A() pre (c1:e1);
    task B() post (c2:e2), (c3:e3)
    catch e0: retry
    catch e2: terminate
  end;
end;

Fig. 13 A workflow containing an uncaught exception

7.1 Construction of Set Constraints

The constraint generation rule, \( D_e \), is presented in Figure 14. The rule [TASK\(_e\)] collects the exceptions described in precondition and postconditions of a task. This rule assumes that all declared exceptions are candidates of uncaught exception. Rule [TRY\(_e\)] generates a constraint set including \( catch(\mathcal{X}, \mathcal{Y}) \). The other rules in Figure 14 propagate a set of exceptions to outer block. By this propagation rules, uncaught exceptions within the enclosing try-catch block can be eliminated by the outer try-catch block at constraint solving phase.

\[
[\text{TR}_e] \quad \mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq \mathcal{Z} \quad \mathcal{X} \supseteq \mathcal{Z}
\]

\[
[\text{CATCH}_e] \quad \mathcal{X} \supseteq \text{catch}(\mathcal{Y}, C) \quad \mathcal{Y} \supseteq \{e_1, \ldots, e_n\} \quad \mathcal{X} \supseteq \{e_1, \ldots, e_n\} \cup C
\]

where \( S \cup C = \{ x \mid x \in S, \forall c \in C. x \text{ is not subexception of } c \} \)

Fig. 15 Constraint Generation Rules : \( S_e \)

7.2 Solving Set Constraints

To solve the collected set constraints, we derive constraint solving rule called \( S_e \), as shown in Figure 15, which includes only two rules. The first one, [TR\(_e\)], is a simple transitive rule. The second one, [CATCH\(_e\)], selects uncaught exceptions from \( \mathcal{Y} \). \( \mathcal{Y} \) includes a set of exceptions generated from task declaration and \( C \) is a set of exceptions having their own handlers which come from catch clauses. The semantic of the operator \( S \cup C \) is to eliminate every element of \( S \) if the element is equal to or subexception of one of the elements in \( C \).

When \( \mathcal{X} \) is constraints set constructed at the first phase, we can get a concise solution by eliminating unnecessary constraints by function \( \text{explicit}(S'_e(\mathcal{X})) \) where the semantic of it is \( \{ \mathcal{X} \supseteq \{e_1, \ldots, e_n\} \mid \mathcal{X} \supseteq \{e_1, \ldots, e_n\} \in S'_e(\mathcal{X}) \} \).

8. Application to Other Workflow Languages

The CPDL proposed in this paper can specify inter-organizational workflow interaction, control flows within a single workflow and exceptions. Because the constructs for each part are very primitive and correspond to basic workflow patterns[14], complicate workflow expressions provided by many workflow languages can be translated into CPDL.

All of the languages devised for inter-organizational workflow provide forms of expression to specify workflow interaction as a common feature. They commonly have expression units of message declarations and message sending/receiving. The syntactic elements of data declaration and \texttt{recv/send} parameters in CPDL are correspondent to them. So the communication deadlock analysis is applicable to the inter-organizational workflow definition described with other languages.

The syntactic elements of CPDL provide basic workflow patterns, such as sequential composition, concurrent composition, conditional branch, and loop composition. Although workflow systems of these days have their different insights into expression capability and
provide diverse ways of representing control flow constructs, we consider them as syntactic sugar that can be translated into compounding of basic operators in CPDL. To enhance understanding of the idea, we use XPDL (XML Process Definition Language) \[6\] as an example workflow language. Because XPDL is a standard language, i.e. widely accepted model in workflow specification, it is weighty to compare expression units of CPDL with those of XPDL. In XPDL, a workflow process is defined by activities and transitions rather than structured control flow constructs. However, as you can see in Figure 16, the mapping example from XPDL to CPDL shows the possibility of complete language translator between the two languages. It means that access conflict analysis also can be applied to other languages with no modification.

<table>
<thead>
<tr>
<th>XPDL constructs</th>
<th>Corresponding CPDL constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal parameter</td>
<td>Input parameter (in), Output parameter (out)</td>
</tr>
<tr>
<td>Route Activity</td>
<td>Branch (if-then-else)</td>
</tr>
<tr>
<td>None Activity</td>
<td>Inert task (0)</td>
</tr>
<tr>
<td>Application Activity</td>
<td>General task</td>
</tr>
<tr>
<td>Subflow Activity</td>
<td>Workflow Definition</td>
</tr>
<tr>
<td>Loop Activity</td>
<td>Loop</td>
</tr>
<tr>
<td>Transition</td>
<td>Branch (if-then-else), Sequential composition</td>
</tr>
<tr>
<td>AND-split, AND-join</td>
<td>Concurrent composition</td>
</tr>
<tr>
<td>XOR-split, XOR-join</td>
<td>Branch composition</td>
</tr>
</tbody>
</table>

Fig. 16 A Mapping from XPDL to CPDL

Applying uncaught exception analysis to other workflow languages might require a little work because there is no standard or widely accepted model for exception mechanisms of workflow systems. However, to define exception mechanism in workflow definition, basically, workflow language has to support expression units for (i) exception declaration, (ii) when exception should be raised, (iii) how to catch raised exceptions and (iv) how to handle the caught exception. CPDL provides some operators for exception specification, such as isa relation for (i), pre/post condition for (ii), try – catch block for (iii) and exception handler for (iv). It is not difficult to translate any workflow language supporting elementary exception mechanism into CPDL.

Because CPDL concentrates on control flow and data flow on a single level of workflow process, CPDL lacks some features of workflow, such as interaction between activities of workflow and applications or human assigned to the activities and data flow on multi-level workflow definition including nested tasks or subflows. To apply our analysis techniques to nested subflow definition, the multi-level workflow definitions should be translated into single level flat workflows. The mapping between Subflow Activity and Workflow Definition in figure 16 shows the relation. But, the interactions between activities and applications(or human) cannot be presented within CPDL. Therefore, it is need to extend CPDL and analysis techniques to verify correctness of data and control flow between them.

9. Conclusion and Future Work

In this paper, we developed a set-based method to detect all possible communication deadlock, access conflict situations and uncaught exceptions in a workflow definition before runtime. We also proposed a workflow definition language, named CPDL, for the effective description of the method. Although CPDL lacks for some features to become a practically useful workflow definition language, it has sufficient features to show how to analyze the problems. Our intention was that the analysis methods could help us prevent losses and damages due to poorly defined workflow definitions.

Compared to other workflow analysis techniques, set based workflow analysis employs a simple and intuitive definition of program approximation. This is motivated by a desire to separate the definition of program approximations from the algorithms used to compute it, and leads to declarative program analysis which is easier to understand and reason about. In contrast, most of approaches in the literature of workflow definition analysis, such as Petri-net and directed graph, provide only an implicit algorithmic definition of runtime properties approximation. The other advantage of set based analysis is that the definition of approximation is very uniform. The three analysis methods in previous sections have same analysis process including constraints constructions phase and resolving them phase. Not only do the rules in each phase have the same form and same processing logic, the set constraints of the analysis methods have the same form and meaning with changing the set expressions of them. We believe that this uniformity has implications for the stability and scalability of the analysis. The uniformity is favorable to make a framework for the analysis of workflow definition.

A straightforward implementation of the set constraint algorithm leads to poor performance. To achieve very substantial improvements, we are studying on how to make appropriate representation schemes and minimization techniques. In the future, we are planning to research on the following topics. One is making a proof of soundness and completeness of our analysis methods. For the proof of the safety, dynamic semantics for CPDL should be defined beforehand using formal methods such as denotational or operational semantics. Especially, the semantics of task should be defined in CPDL. We expect that it gives the soundness of the analysis techniques in this paper. However, it is not easy work due to the absence of widely accepted formal model for workflow management. In addition, CPDL needs to expand the expression power to be practically used in real workflow system. Then, there will be probably many analysis points in workflow definitions be-
sides the problems mentioned in this paper, such as verification of interactions between a workflow and human or applications assigned to activities. And, logically unreachable paths and inappropriate usage of workflow relevant data are another good examples.

References


