Set-Based Access Conflicts Analysis of Concurrent Workflow Definition

Minkyu Lee
Plastic Software, Inc.
485-1 Youngdang-dong, Nam-gu, Pusan, Korea
niklaus@mail.plasticsoftware.com

Dongsoo Han and Jaeyong Shim
Information and Communications University
School of Engineering
Yusong P.O.Box 77, Taejon, Korea
{dshan, jaeyong7}@icu.ac.kr

Abstract

An error-comprising workflow definition might provoke serious problems to an enterprise especially when it is involved with mission critical business processes. Concurrency of workflow processes is known as one of the major sources causing such an invalid workflow process definition. So the conflicts caused by concurrent workflow processes should be considered deliberately when defining concurrent workflow processes. However it is very difficult to ascertain whether a workflow process is free from conflicts or not without any experimental executions at runtime. Which will be very tedious and time consuming work to process designers. If we can analyze the conflicts immanent in concurrent workflow definition prior to runtime, it will be very helpful to business process designers and many other users of workflow management system. In this paper, we propose a set-based constraint system to analyze possible read-write conflicts and write-write conflicts between activities which reads and writes to the shared variables in a workflow process definition. The system is composed of two phases. In the first phase, it generates set constraints from a structured workflow definition. In the second phase, it finds the minimal solution of the set constraints.

1 Introduction

A *workflow* is a collection of cooperating, coordinated activities designed to carry out a well-defined complex process, such as trip planning, insurance claiming, health care business processes[5]. An activity in workflow could be performed either by a human, a device, or a program. *Workflow management system (WFMS)* is a software system which provides tools to define workflow processes and enactment services to create and manage the execution of workflows.

Once a workflow is invoked in WFMS, the activities are executed along the control paths and data flow information in the process definition. Several activities can be in active state in a concurrent workflow process. We call them *concurrent activities* in this paper. Concurrent activities may access the shared data in any order because their order of accessing is situation dependent. But the non-deterministic access of concurrent activities to shared data may bring unexpected result from the workflow execution. The following race problems can be considered from the execution of concurrent activities :

1. *read-write conflict* is a situation when 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 where A and B are concurrent activities and *vice versa*.

2. *write-write conflict* is a situation when an activity A tries to write data to a shared variable x and an activity B also tries to write data to a shared variable x where A and B are concurrent activities.

Above race conditions are difficult to be detected when the workflow process is in execution state and can result in serious problems to business critical processes. Thus such access conflict-comprising definitions should be eliminated or cleared completely before the real execution of the processes. When designing relatively small workflow processes, such definitions might be avoided by careful designing of the processes. However when the workflow processes get complicated, it is not enough to leave all the responsibilities for the access conflict free definitions to only workflow designers. More systematic ways to detect the conflicts from the definitions and to notify them to the designers are required.

Many researches to analyze race conditions have been performed in programming language research communities. War-lock[10] is a static race detection system for ANSI C programs and Eraser[11] is a tool for detecting race conditions and deadlocks dynamically. Aiken and Gay[1] studied static race detection in the context of SPMD(Single

0-7695-1128 7/01 $10.00 © 2001 IEEE

w	::=	**0**	(inert task)
	|	(w)	(priority)
	|	**task** $t(p_1, ..., p_n)$	(task execution)
	|	w_0 ; w_1	(sequential composition)
	|	$w_0 \parallel w_1$	(concurrent composition)
	|	**if-then** w_0 **else** w_1	(branch)
	|	**while-do** w	(loop)
p	::=	**in** x	(input parameter)
	|	**out** x	(output parameter)

Figure 1. Abstract Syntax of SWDL

Program Multiple Data) style programs, and Flanagan and Freund[3] presented a static race detection analysis technique for multithreaded Java programs. In while these researches have been done in the context of programming languages, our analysis has done in different approach in the context workflow.

In this paper, we propose a set-based access conflict analysis method to detect all the possible access conflicts prior to the execution of workflow process. We define a small target workflow definition language for the description of the method focusing on the language. But the method can be easily extended to the general workflow definition languages like WPDL(Workflow Definition Language)[12]. The method is composed of two phases. In the first phase, it generates set constraints from a structured workflow definition. In the second phase, it solves the set constraints obtained from the first phase.

This paper is organized as follows. In Section 2, we introduce a simple workflow definition language that is used as target language of the analysis. Section 3 presents details of the analysis method with illustrations and Section 4 illustrate an implementation and experiment results. Finally, in Section 5, we draw conclusion and future work.

2 A Workflow Definition Language

We define a simple workflow definition language, named SWDL(Structured Workflow Definition Language), as target language for the succinct and clear description of our access conflicts analysis method. Figure 1 shows abstract syntax of SWDL. The SWDL only contains the features that are necessary to express control flow and data flow of a workflow process because they contain enough information to analyze the access conflicts of a language. The semantics of each feature are described as follows:

- "**0**" : Inert workflow process.
- "(w)" : This is used only to bundle up.
- "**task** $t(p_1, ..., p_n)$" : This means the execution of a task named t. The task may have zero or more parameters. Each parameter is either input parameter, denoted by **in**, or output parameter, denoted by **out**. The semantics of execution is that the task reads all the input parameters from shared database by *pass-by-value* manner and evaluates the task with the parameters and then replaces the shared data with the output parameters of the evaluated task. The *pass-by-value* parameter passing is more reasonable than *pass-by-reference* in two reasons. The first reason is that recent workflow management systems are implemented in concerning with mobile environment. In mobile environment, each actor is mobile so the actor may be disconnected to workflow management system[4]. To perform the activity in disconnected state, all the input values should be copied to the disconnected activity site before the activity to be started. The second reason is that the activities may be distributed in different locations. We cannot assume that each activity is always in connected state with other activities because network bandwidth is amenable to change and the connections are not stable. Input parameters of an activity may not be delivered in time during the processing of the activity. Thus, the assumption that all the input parameters are prepared by *call-by-value* mechanism before an activity starts its work is more reasonable.
- "w_0 ; w_1" : Two workflow processes w_0 and w_1 are executed sequentially. So w_1 starts its execution after the end of w_0.
- "$w_0 \parallel w_1$" : Two workflow processes w_0 and w_1 are executed concurrently in interleaved manner. So race conditions may occur between w_0 and w_1.
- "**if-then** w_0 **else** w_1" : This is the same control structure as *if-then-else* statement in programming languages. One of the two workflow processes w_0 and w_1 are selected and executed. Condition expression to determine which one is selected is omitted in SWDL language because the selection is not necessarily required in our analysis.
- "**while-do** w" : Workflow process w is executed repetitively. Repetition condition is omitted because of the same reason as the above item.

Note that data flow of workflow process is not explicitly defined but implicitly included in SWDL. It is obvious that the features of SWDL are not sufficient but most features necessary to analyze access conflicts between activities are included in the SWDL specifications.

The control structure of SWDL is similar to that of structured programming languages such as C and Pascal. So it can define structured control flow of a workflow process.

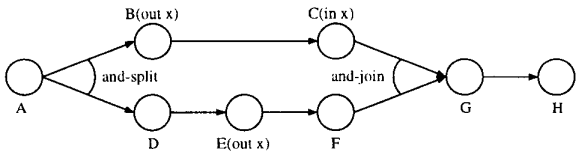


Figure 2. An example of workflow definition

Structured workflow process definition has two advantages over WPDL[12]-standard workflow specification languages in which activities and control flow among them are defined in separate manner.

1. *Syntax-level prevention of invalid definition*: Structured definition of workflow process is very useful in preventing various invalid workflow definitions by syntax-level grammar checking. Isolated activities and transitions from outer-loop into inner-loop are the examples of invalid definitions. Some of invalid definitions can be forced not be defined in SWDL and some of them can be checked during the parsing phase.

2. *Readibility*: Defining activities and transitions among them in separate manner like WPDL makes it very difficult for one to read the flow of process directly from the process definition. Since control structure of SWDL-like the approach of [2] is similar to that of popular structured programming languages such as C and Pascal, it is more friendly to users and users can grasp the control flow of the process more easily.

Figure 2 shows a simple workflow process definition. Activity name is written in upper case letters and shared variable is written in lower case letters. After activity A is executed, $(B;C)$ and $(D;E;F)$ are executed concurrently and then G and H are executed sequentially. Activity B and activity E write to the variable x and activity C reads the value of the variable x. This workflow process is represented in SWDL as follows:

$$A\ ;\ (B(\textbf{out}\ x)\ ;\ C(\textbf{in}\ x)\ \|\ D\ ;\ E(\textbf{out}\ x)\ ;\ F)\ ;\ G\ ;\ H$$

3 Access Conflict Analysis

In workflow process definition presented in Figure 2, $(B;C)$ and $(D;E;F)$ may be executed concurrently and they may access the shared variable x. In this case, two access conflicts can be provoked. The first access conflict is *write-write conflict* caused by B, E. The second access conflict is *read-write conflict* caused by C and E.

To analyze all the possible conflicts, we adopt set constraint system that is used to analyze runtime features of programming languages[8][9][6][7]. The method consists

$$[\text{Null}]\quad \mathbf{0} \rhd \phi \qquad [\text{Pri}]\quad \frac{w \rhd C}{(w) \rhd C}$$

$$[\text{Task}]\quad \begin{array}{l}\textbf{task}\ x(\textbf{in}\ i_1, \cdots, \textbf{in}\ i_n, \textbf{out}\ o_1, \cdots, \textbf{out}\ o_m) \rhd \\ \{\mathcal{X} \supseteq taskR(x, i_1), \cdots, \mathcal{X} \supseteq taskR(x, i_n), \\ \mathcal{X} \supseteq taskW(x, o_1), \cdots, \mathcal{X} \supseteq taskW(x, o_m)\}\end{array}$$

$$[\text{Seq}]\quad \frac{w_0 \rhd C_0 \quad w_1 \rhd C_1}{w_0;w_1 \rhd \{\mathcal{X} \supseteq \mathcal{X}_{w_0}, \mathcal{X} \supseteq \mathcal{X}_{w_1}\} \cup C_0 \cup C_1}$$

$$[\text{Par}]\quad \frac{w_0 \rhd C_0 \quad w_1 \rhd C_1}{w_0 \| w_1 \rhd \{\mathcal{X} \supseteq \mathcal{X}_{w_0}, \mathcal{X} \supseteq \mathcal{X}_{w_1}, \mathcal{X} \supseteq par(\mathcal{X}_{w_0}, \mathcal{X}_{w_1})\} \cup C_0 \cup C_1}$$

$$[\text{While}]\quad \frac{w \rhd C}{\textbf{while-do}\ w \rhd \{\mathcal{X} \supseteq \mathcal{X}_w\} \cup C}$$

$$[\text{If}]\quad \frac{w_0 \rhd C_0 \quad w_1 \rhd C_1}{\textbf{if-then}\ w_0\ \textbf{else}\ w_1 \rhd \{\mathcal{X} \supseteq \mathcal{X}_{w_0}, \mathcal{X} \supseteq \mathcal{X}_{w_1}\} \cup C_0 \cup C_1}$$

Figure 3. Constraint Generation Rules : $\rhd$

of two phases. In the first phase, it generates set constraints from the source and in the second phase, it finds the minimal solution from the set constraints generated at the first phase. In our analysis, *every* workflow expression w of input workflow process definition has set constraints $\mathcal{X}_w \supseteq se$. The set variable $\mathcal{X}$ is used to collect(represent) w's possible access conflicts. For example, suppose that $(A \ \|\ B\ ;\ C)$ is an input workflow, *every* workflow expression A, B, C, $B\ ;\ C$, $(A \ \|\ B\ ;\ C)$ has its own set variables $\mathcal{X}_a$, $\mathcal{X}_b$, $\mathcal{X}_c$, $\mathcal{X}_{bc}$, $\mathcal{X}_{abc}$ respectively. Finally, $\mathcal{X}_{abc}$ will have all the possible conflicts of the input workflow. Each set constraint is in the form of $\mathcal{X} \supseteq se$ where se is a set expression. The meaning of set constraint $\mathcal{X} \supseteq se$ is intuitive: that is, set $\mathcal{X}$ contains the set represented by the set expression se.

In the next subsection, we present how to generate set constraints from an input workflow definition and then show how to solve the set constraints with an example.

3.1 Construction of Set Constraints

Figure 3 shows the rules to generate set constraints for every workflow expression. The set variable $\mathcal{X}$ is for the current workflow expression to which the rule applies and the subscripted set variable $\mathcal{X}_w$ is for the workflow expression w. The relation "$w \rhd C$" represents that "constraints C are generated from workflow expression w."

Every workflow expression of workflow definition presented in Figure 2 is underlined and labeled. Each label will be used as subscript of its set variable.

$$\underline{A}_a\ ;\ (\underbrace{\underbrace{\underline{B(\textbf{out}\ x)}_b\ ;\ \underline{C(\textbf{in}\ x)}_c}_{bc}\ \|\ \underbrace{\underline{D}_d\ ;\ \underbrace{\underline{E(\textbf{out}\ x)}_e\ ;\ \underline{F}_f}_{ef}}_{df}}_{bf}\ )\ ;\ \underbrace{\underline{G}_g\ ;\ \underline{H}_h}_{gh}}_{bh}}_{ah}$$

Set constraints for this example generated by $\rhd$ is presented in Figure 4 and the expected result is the minimal set which satisfies all the constraints.

$\mathcal{X}_{ah} \supseteq \mathcal{X}_a$ $\quad \mathcal{X}_{ah} \supseteq \mathcal{X}_{bh}$
$\mathcal{X}_{bh} \supseteq \mathcal{X}_{bf}$ $\quad \mathcal{X}_{bh} \supseteq \mathcal{X}_{gh}$
$\mathcal{X}_{bf} \supseteq \mathcal{X}_{bc}$ $\quad \mathcal{X}_{bf} \supseteq \mathcal{X}_{df}$ $\quad \mathcal{X}_{bf} \supseteq par(\mathcal{X}_{bc}, \mathcal{X}_{df})$
$\mathcal{X}_{gh} \supseteq \mathcal{X}_g$ $\quad \mathcal{X}_{gh} \supseteq \mathcal{X}_h$
$\mathcal{X}_{bc} \supseteq \mathcal{X}_b$ $\quad \mathcal{X}_{bc} \supseteq \mathcal{X}_c$
$\mathcal{X}_{df} \supseteq \mathcal{X}_d$ $\quad \mathcal{X}_{df} \supseteq \mathcal{X}_{ef}$
$\mathcal{X}_{ef} \supseteq \mathcal{X}_e$ $\quad \mathcal{X}_{ef} \supseteq \mathcal{X}_f$
$\mathcal{X}_b \supseteq taskW(B, x)$ $\quad \mathcal{X}_c \supseteq taskR(C, x)$ $\quad \mathcal{X}_e \supseteq taskW(E, x)$

Figure 4. Set Constraints Generated by $\triangleright$

3.2 Solving Set Constraints

In the previous subsection we showed how to generate set constraints. In this subsection we present how to compute the solution from the set constraints. To solve the set constraints we introduce constraint solving rules $\mathcal{S}$, which is presented in Figure 5. Each rule in $\mathcal{S}$ is written in the following way:

$$\frac{\mathcal{C}_1 \cdots \mathcal{C}_n}{\mathcal{C}_1 \cdots \mathcal{C}_m}$$

Using this notation, one or more set constraints already contained are written above a bar and new set constraints are written below the bar. 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 $\mathcal{S}$ to set of constraints $\mathcal{C}$ and the iterative application is denoted by $\mathcal{S}^*(\mathcal{C})$. Although $\mathcal{S}^*(\mathcal{C})$ certainly denotes the solution, we can have more concise solution by eliminating unnecessary and redundant constraints. Final result is in the followings:

$$\{(\mathcal{X} \supseteq se) \in \mathcal{S}_c^*(\mathcal{C}) \mid se = conflictRW(s,t,x)\}$$
$$\cup \quad \{(\mathcal{X} \supseteq se) \in \mathcal{S}_c^*(\mathcal{C}) \mid se = conflictWW(s,t,x)\}$$

If $\mathcal{C}$ is same as Figure 4 then the final result becomes:

$$\{\mathcal{X}_{ah} \supseteq conflictRW(C, E, x), \mathcal{X}_{ah} \supseteq conflictWW(B, E, x)\}$$

The time complexity of the algorithm to estimate access conflicts is $O(n^3)$ where n is the size of input workflow 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 $O(n) + O(n^3) = O(n^3)$.

4 Implementation and Evaluation

We have implemented the access conflict analysis system for SWDL in Java. First, we made a parser for SWDL

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq taskR(t,x)}{\mathcal{X} \supseteq taskR(t,x)}$$

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq taskW(t,x)}{\mathcal{X} \supseteq taskW(t,x)}$$

$$\frac{\mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{Z}) \quad \mathcal{Y} \supseteq par(\mathcal{V}, \mathcal{W})}{\mathcal{X} \supseteq par(\mathcal{V}, \mathcal{Z}), \mathcal{X} \supseteq par(\mathcal{W}, \mathcal{Z})}$$

$$\frac{\mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{Z}) \quad \mathcal{Z} \supseteq par(\mathcal{V}, \mathcal{W})}{\mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{V}), \mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{W})}$$

$$\frac{\mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{Z}) \quad \mathcal{Y} \supseteq taskR(s,x) \quad \mathcal{Z} \supseteq taskW(t,x)}{\mathcal{X} \supseteq conflictRW(s,t,x)}$$

$$\frac{\mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{Z}) \quad \mathcal{Y} \supseteq taskW(s,x) \quad \mathcal{Z} \supseteq taskR(t,x)}{\mathcal{X} \supseteq conflictRW(s,t,x)}$$

$$\frac{\mathcal{X} \supseteq par(\mathcal{Y}, \mathcal{Z}) \quad \mathcal{Y} \supseteq taskW(s,x) \quad \mathcal{Z} \supseteq taskW(t,x)}{\mathcal{X} \supseteq conflictWW(s,t,x)}$$

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq conflictRW(s,t,x)}{\mathcal{X} \supseteq conflictRW(s,t,x)}$$

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq conflictWW(s,t,x)}{\mathcal{X} \supseteq conflictWW(s,t,x)}$$

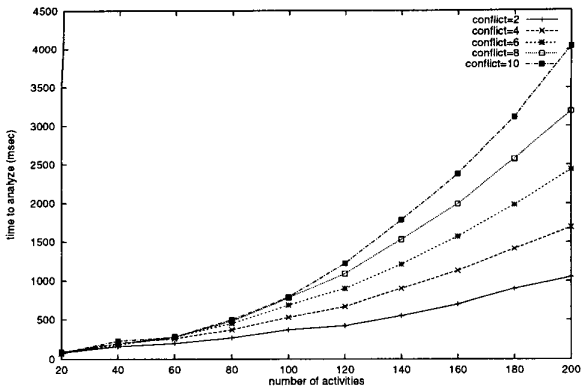
Figure 5. Constraint Solving Rules : $\mathcal{S}$

using an automatic parser generator and implemented an AST(abstract syntax tree) builder. The implementation consists of two phases. The first phase traverses the AST and generates a set of constraints based on the constraint generation rules presented in Figure 3. In the second phase, we apply constraint solving rules iteratively until the set of constraints does not change. The algorithm certainly terminates since the size of constraint set increases monotonically by the iterations and the size of the set is limited by the number of $\binom{n}{2}$, where n is the size of workflow expression.

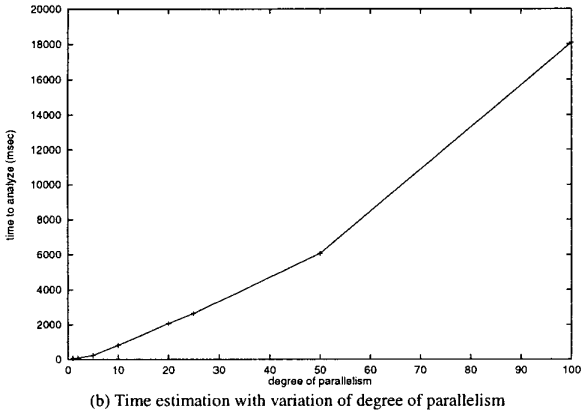
The implemented system is tested for the various input conditions and the execution time is measured. The inputs are constructed by changing the number of activities, conflicts, and degree of parallelism. The empirical results are presented in Figure 6. The graph (a) shows that as the number of activities and conflicts increases so does execution time as expected by the time complexity. In the graph (b), we measured the analysis time of processes having the same 100 activities with no conflict but difference degree of parallelism. It shows that the analysis time is influenced by the degree of parallelism. Since most practical workflows rarely exceeds hundreds of activities, and the result can be obtained within a few seconds we can conclude that the proposed method is practically useful.

5 Conclusion and Future Work

We have presented a set-based method to detect all possible access conflict situations in a workflow process definition before runtime. We also have proposed a workflow def-



(a) Time estimation with variation of number of activities and conflicts



(b) Time estimation with variation of degree of parallelism

Figure 6. Experiment Results

inition language, named SWDL, for the effective description of the method. Although SWDL lacks for some features to become a general purpose workflow definition language, it has sufficient features to analyze access conflicts in concurrent workflow definition. Thus we expect that the method developed in this paper can be applied to general purpose workflow definition languages fairly easily.

Our method is to predict the access conflicts among concurrent activities in a workflow instance not those among inter-workflow instances. Actually in workflow management system, the situation where multiple instances of workflow processes try to access shared data simultaneously can happen. So the access conflicts among inter-workflow instances also must be considered. It seems that they are inherently the same problem but more in-depth analysis will be required to be convinced and to solve such a problem.

The other direction of our research is to generate new conflict free workflow process definition automatically using the obtained conflict information from our analysis. One possible approach is simply to put *lock* and *unlock* operation on shared variables in the front and rear of activities which may conflict. Such approach can free business process designers from the concerning of provoking access conflicts when defining workflow processes.

References

[1] A. Aiken and D. Gay, "Barrier inference," Proceeding of the 25th Symposium on Principles of Programming Languages, pages 243-354, 1998.

[2] C. Dengi and S. Neftci, "Dflow Workflow Management System," Proceedings of 8th International Workshop on Database and Expert Systems Applications, 1997.

[3] C. Flanagan and S. Freund, "Type-Based Race Detection for Java," Proceedings of ACM Conference on Programming Language Design and Implementation, June, 2000.

[4] G. Alonso, R. Gunthor, M. Kamath, D. Agrawal, A. El Abbadi and C. Mohan, "Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System," 3rd International Conference on Cooperative Information Systems, Vienna, May 1995.

[5] H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V.Ramakrishnan, "Logic based modeling and analysis of workflows," In ACM Symposium on Principles of Database Systems, June, 1998.

[6] K. Yi and B. Chang, "Exception Analysis for Java," ECOOP'99 Workshop on Formal Techniques for Java Programs, June, 1999.

[7] K. Yi and S. Ryu, "A Cost-effective Estimation of Uncaught Exceptions in SML Programs," Theoretical Computer Science, Vol. 273, No. 1, 2000.

[8] N. Heintze, "Set Based Program Analysis," Ph.D.thesis, School of Computer Science, Carnegie Mellon University, October 1992.

[9] N. Heintze, "Set Based Analysis of ML Programs," Carnegie Mellon University Technical Report CMU-CS-93-193, July 1993.

[10] N. Sterling, "A static data race analysis tool," In USENIX Winter Technical Conference, pages 97-106, 1993.

[11] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A dynamic data race detector for multithreaded programs," ACM Transactions no Computer Systems, 15(4):391-411, 1997.

[12] Workflow Management Coalition, "Interface 1: Process Definition Interchange Process Model," Document Number WfMC TC-1016-P, October 29, 1999.