Lightweight query-based analysis of workflow process dependencies

W. Dai, D. Covvey, P. Alencar, and D. Cowan
The Journal of Systems and Software (2009)
Introduction

Background

Overall approach

Dependency model

Logical based dependency representation and query

Case study

Conclusion

Discussion
Changes in workflow processes are unavoidable
- Modification of regulations or laws leads to changes
- Modified activities or processes impact on other activities or processes

Need to identify the entities affected by proposed change
- Assess the effects of a change on workflow process to reduce the impact
Previous works have limitations

- **Adam’s work**[1]
  - Construct Petri-Net based framework for modeling, simulation, and analysis of workflows
  - Focus on only routing dependency
    - Lead to an incomplete understanding of dependencies
    - Not consider dependency among processes

- **Kim’s work**[2]
  - Design workflow management system for process execution
    - Not focus on change impact analysis
  - Introduce a dependency analysis framework consisting of four separate dependency

---


Previous works have limitations (cont’)

- Ryder’s work[1]
  - Analyze change impact for Object Oriented Programs by query mechanism
  - Not focus on workflow process dependency

In this paper,

- Analyze workflow process dependency
  - Focus on routing, data and role dependency model
    - For sufficient dependency analysis between entities
  - Using query mechanism based on logic programming
    - For more efficient and accurate than manual procedure
Prolog

- Logic programming language
  - State properties which are true of system as facts
  - State query mechanism as rules
  - Infer the new facts from existing knowledge base

Example

- Facts
  - RelatedTo(Act_1, Act_2)
  - RelatedTo(Act_2, Act_3)

- Rule
  - DependOn(A, B) :- RelatedTo(A, B)
  - DependOn(A, B) :- RelatedTo(A, C), DependOn(C, B)

- Query
  - ?- DependOn(Act_1, Result)
    => RelatedTo(Act_1, Result)
    => RelatedTo(Act_1, Act_2) -> True
    => RelatedTo(Act_1, Act_3) -> False
    => RelatedTo(Act_1, C), DependOn(C, Result) => .... =>
    => RelatedTo(Act_1, Act_2), RelatedTo(Act_2, Act_3) -> True

- Output
  Result = Act_2, Act_3
Overall approach

- Analyze dependency using Prolog
  - Routing dependency
  - Data dependency
  - Role dependency

- Identify dependency entities
- Build knowledge base through Prolog facts
- Define query rules
- Query for dependency analysis
Routing dependency

- Define the execution order of activities in a process
  - Order of activities represented by a directed edge corresponds to routing dependency relationship

- Use 5 routing types
  - Sequential, And-split, And-join, Or-split, Or-join

- Example of process representation

- Routing dependency
  - (A,B), (A,D), (D,E), etc.
 Dependency model (2/3)

- Data dependency
  - Define the flow of data associated with each activity
  - Example

  \[
  \begin{align*}
  \text{Data}_i & \rightarrow \text{Data}_m \\
  \text{Output data} & \rightarrow \text{Input data} \\
  \end{align*}
  \]

  - \(\text{Data}_i\) and \(\text{Data}_m\) are dependent directly
  - \(A_i\) and \(A_m\) may not be neighboring
Dependency model (3/3)

- Role replacement dependency
  - Define the changeable roles corresponding to a role associated with an activity
  - Viewed as hierarchical structure
    - Higher role can execute activities executed by lower role
      - Example
        - Example
          - Chief radiologist can execute activities which executed by CT radiologist
        - Replacement relationship is activity-based
          - $r_1$ is not replaceable to $r_2$
          - $r_1$ is only replaceable to $r_2$ executes $a_0$
## Process entities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Definition</th>
<th>Single entity representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Set of processes in analysis domain</td>
<td>( p )</td>
</tr>
<tr>
<td>A</td>
<td>Set of activities in process ( p )</td>
<td>( a )</td>
</tr>
<tr>
<td>C</td>
<td>Set of basic control routing in process ( p )</td>
<td>( c ) (sequential, or-split, or-join, and-split, and-join)</td>
</tr>
<tr>
<td>D</td>
<td>Set of output data generated from activity ( a ) in process ( p )</td>
<td>( d_1^a, d_2^a, \ldots, d_k^a )</td>
</tr>
<tr>
<td>R</td>
<td>Set of roles assigned to activity ( a ) in process ( p )</td>
<td>( r_a )</td>
</tr>
</tbody>
</table>
Logical based dependency representation and query (2/7)

- **Building knowledge base**
  - Define basic facts of process

<table>
<thead>
<tr>
<th>Fact</th>
<th>Argument types</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>$P$</td>
<td>process($p1$), process($p2$), …</td>
</tr>
<tr>
<td>activity</td>
<td>$A, P$</td>
<td>activity($a1, p1$), activity($a4, p2$), …</td>
</tr>
<tr>
<td>routing</td>
<td>$structure(C,P)$</td>
<td>routing(and-split(and-split_1, $P1$)), …</td>
</tr>
<tr>
<td>data</td>
<td>$D, A, P$</td>
<td>data($data_a4, a4, p2$), …</td>
</tr>
<tr>
<td>role</td>
<td>$R, A, P$</td>
<td>role($r1, a1, p1$), …</td>
</tr>
</tbody>
</table>

- **Example**

![Diagram showing process $p1$ and $p2$ with data $a4$ and role $r1$]
Logical based dependency representation and query (3/7)

Building knowledge base (cont’)

Define dependency facts

<table>
<thead>
<tr>
<th>Fact</th>
<th>Argument types</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>routingDep</td>
<td>A₁, P₁, [routing], A₂, P₂</td>
<td>routingDep(a₁, p₁, [and-split(and-split_1, p₁)], a₄, p₂), ...</td>
</tr>
<tr>
<td>dataDep</td>
<td>A₁, P₁, structure(D, A₂, P₂)</td>
<td>dataDep(a₅, p₂, input(data_a₄, a₄, p₂)), ...</td>
</tr>
<tr>
<td>roleDep</td>
<td>R₁, A, P, R₂</td>
<td>roleDep(r₁, a₁, p₁, r₂), ...</td>
</tr>
</tbody>
</table>

Example

- Inter-dependency among processes is supported
- All facts include the process information
Query rules

- Define at different level
  - Activity level
    - Find activities which depend on a activity to be changed
  - Process level
    - Find activities which depend on any activity in a process to be changed

- Define in different direction
  - Forward query
    - Find succeeding entities to be impacted based on given activity or process
  - Backward query
    - Find preceding entities to impact on given base activity or process
Logical based dependency representation and query (5/7)

Query rules for routing dependency

<table>
<thead>
<tr>
<th>Query rule</th>
<th>Result</th>
<th>Query rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>postActivityRouting(A, P, Results)</td>
<td>(A, [routings], P)</td>
<td>postProcessRouting(P, Results)</td>
</tr>
<tr>
<td>preActivityRouting(A, P, Results)</td>
<td>(A, [routings], P)</td>
<td>preProcessRouting(P, Results)</td>
</tr>
<tr>
<td>activityReachPaths(A, P, B, Q)</td>
<td>(A, P) path</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

- postActivityRouting(a1, p1, Results)
  - Results = activity(a2, [and-split(and-split_1, p1)], p1), activity(a4, [and-split(and-split_1, p1)], p2)

- activityReachPath(a4, p2, a7, p1)
  - Results = activity(a5, p2), activity(a6, p2)
Logical based dependency representation and query (6/7)

Query rules for data dependency

<table>
<thead>
<tr>
<th>Query rule</th>
<th>Result</th>
<th>Query rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>postActivityData(A, P, [D], Results)</td>
<td>(A, P)</td>
<td>postProcessData(P, Results)</td>
</tr>
<tr>
<td>preActivityData(A, P, [D], Results)</td>
<td>(A, P)</td>
<td>preProcessData(P, Results)</td>
</tr>
</tbody>
</table>

**Example**

- postActivityData(a1, p1, Results)
  - Results = activity(a2, p1), activity(a4, p2)
- preprocessData(a1, p1, dataA, Results)
  - Results = activity(a2, p1)

© KAIST SE LAB 2009
Query rules for role dependency

<table>
<thead>
<tr>
<th>Query rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleReplaceable(R1, R2, A, P)</td>
<td>Y or N</td>
</tr>
<tr>
<td>roleActivityReplace(R, Results)</td>
<td>(R, A, P)</td>
</tr>
<tr>
<td>roleReplace(R, A, P, Results)</td>
<td>(R)</td>
</tr>
</tbody>
</table>

Analysis example

- roleReplaceable(rA, rB, A1, p1)  => Results = N
- roleActivityReplace(rA, Results) => Results = role(rB, a2, p1)
- roleReplace(R, A, P, Results)    => Results = rB
Case study (1/3)

- Apply this approach to real case
  - Use healthcare workflow process of Grand River Region cancer Center
    - 4 processes, 16 activities and 14 data and 5 roles
  - Use prolog tool XSB
1. Build knowledge base from workflow process diagram and associated data and role table by identifying logic facts

---

**Case Study: Chemotherapy Workflow Processes with Routing Controls**

<table>
<thead>
<tr>
<th>Activity (Process)</th>
<th>Activity Name (ID)</th>
<th>Input/Output Data</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rebook patient for another assess (PC).</td>
<td>orsplit_pcBlood</td>
<td></td>
<td>Booking Staff</td>
</tr>
<tr>
<td>Doctor prepare for follow up cycle (PC).</td>
<td>pcOrderPreparation</td>
<td></td>
<td>Physician</td>
</tr>
<tr>
<td></td>
<td>pcOrderModification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>orJoin_pcOrder</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

% Routing Dependency

routingDep(pcAssessment, pc, [orsplit(orsplit_pcBlood, pc)], pcRebook, pc).

% Data Dependency

dataDep(pchConfirmation, pch, input(medicine, ctPreparation, ct)).
2. Query for dependency analysis

<postActivityRouting query>

```
postActivityRouting(ctPreparation, ct, Results).
```

Results = [activity pchConfirmation, pc], [landsplit(andSplit ctMedicine, ct), andJoin(andJoin pchMatch, pch)], activity ctFileOrder, ct, [landsplit(andSplit ctMedicine, ct)]]

<workflow process>

[Diagram of workflow process with nodes and edges indicating the flow of activities and their dependencies.]
Conclusion

- Analyze the dependency relationship between changed parts and potentially affected parts
  - Analyze 3 types of dependency
    - Routing, data and role
  - Analyze at different levels
    - Activity and process
  - Analyze in different directions
    - Forward and backward

- Analyze dependency using logical programming
  - by query from knowledge base
Discussion

❖ Contributions
  ▪ Propose solid foundation for other dependency relationship analysis
  ▪ Propose effective query based dependency analysis

❖ Limitations
  ▪ Identify the facts manually
  ▪ Not support dependency analysis of workflow process in distributed environment
Thank You.