
Reasoning about Repairability of Workflows at Design Time

Gaston Tagni, Annette ten Teije, Frank van Harmelen
Department of Artificial Intelligence
Vrije Universiteit Amsterdam, The Netherlands

Presentation given by: Claudia Picardi

1st International Workshop on QoS in Self-healing Web Services
September 1st, 2008. Milan, Italy

Introduction

- Composition/Integration of Web services has the potential to enable B2B applications
- Such compositions are usually modeled as workflows
- One of (many) desired properties of composite Web services is their [repairability](#)
- To some extent repairability is a design issue as it requires methodologies, tools and guidelines to support the design and specification of repairable workflow processes

Motivation

- Current approaches do not consider evaluating (at design time) the correctness of a workflow definition to ensure that repair actions can be applied to its activities
- We propose a method for reasoning about the repairability of workflows at design time.
- The results of the analysis can be used by designers to improve the repairability of a workflow

Outline of the talk

- ❖ Some preliminary definitions
- ❖ Repairability of Workflow Schemas
- ❖ Heuristic-based analysis
- ❖ Repairability Reasoning Algorithm
- ❖ Validation
- ❖ Conclusions

Workflow Model

- Workflows are modeled using directed graphs. Nodes can denote control-flow activities or primitive activities
- Workflow Instance: *a snapshot of a Wf execution at a given point in time that captures the state of variables and activities*
- A subset of the variables defined in the Wf is called **goal variables**. Such variables are those whose value is used for evaluating the correctness of Wf executions.
- Only activities that modify (directly/indirectly) goal variables are relevant for the repairability analysis

Repair model

- The repair model is based on the execution of **repair plans** and **repair actions**
- Given a **faulty instance** and the list of **faulty activities** (diagnosis information) a repair planner tries to generate a repair plan such that when executed, the state of the workflow is equivalent to a correct execution
- Repair actions considered:
 - **retry(X)**: activity X is retried. Equivalent to re-invoking the service that offers activity X's operation
 - **subs(X,Y)**: the service providing activity X's operation is substituted by a service providing activity Y's operation. X and Y are functionally equivalent
 - **comp(X,Y)**: the effects produced by activity X's operation are compensating by executing activity Y's operation.

Repairability of Workflows

- It is defined in terms of: **repairability of activities** and **repairability of workflow instances**
- Repairable activity: *an activity is repairable iff there is a set of repair actions or pre-defined repair handlers that can be applied to it.*
- (Weak) Repairability of Workflows: *a Wf schema S is repairable iff at least one faulty instance $I_f(S)$ is repairable.*
- Repairable Faulty instance: *an instance I of S is repairable iff every faulty and infected activity is repairable.*
- Faulty Instance: *a Wf instance where some of the activities are diagnosed as faulty*

Repairability of Workflows (cntd.)

- Given the lack of instance information at design time it is not possible to enumerate Wf instances. E.g.
 - No information about state of activities
 - No information about state of variables
- Therefore, we use heuristic information to estimate the repairability of a Wf
- Incomplete information makes impossible to guarantee the repairability of Wfs and **only necessary conditions for repairability can be checked.**

Heuristic-based analysis

- The reparability of a Wf schema is determined using a **heuristic-based analysis** that exploits the relation between repairable activities and repairable faulty instances.

The more repairable activities the Wf schema has the more repairable instances the schema supports.

Heuristic: $h(S)$ = *number of repairable activities of Wf schema S*

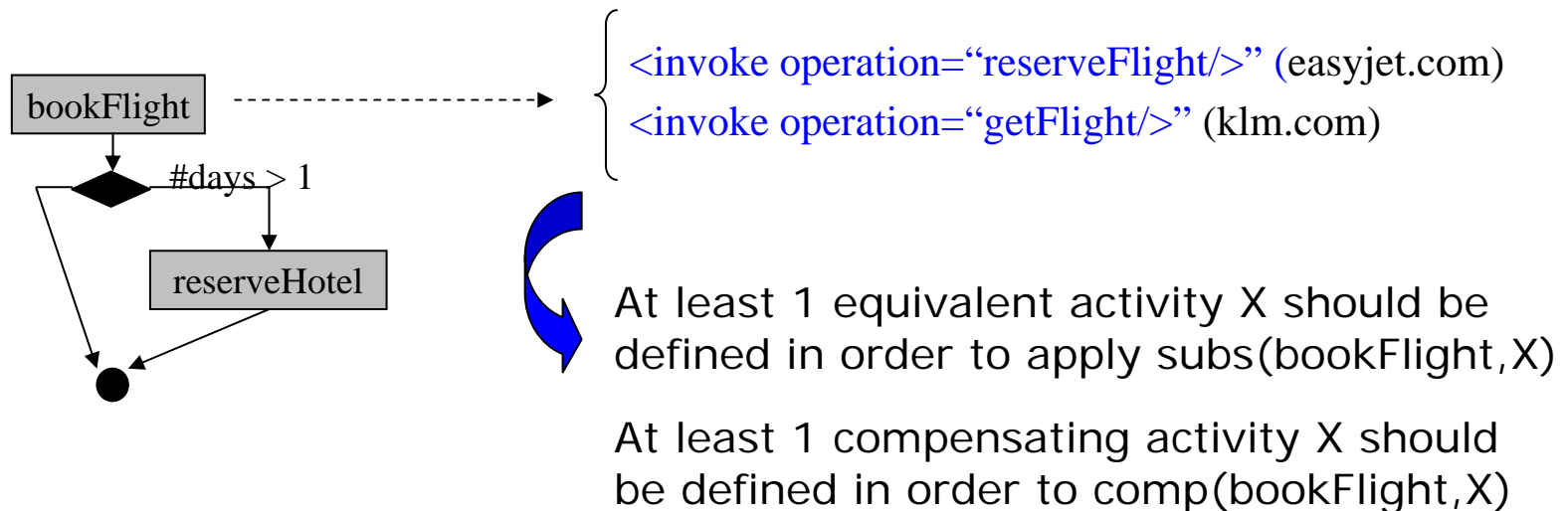
- The idea consists in analyzing a Wf schema in order to identify design flaws leading to non-repairable activities.
- The analysis considers different **reparability factors** affecting the reparability of activities/workflows

Repairability Factors

- The repairability of a Wf is influenced by:
 - Repairability of constituent activities
 - Set of repair actions
 - Data and control-flow dependencies
 - Existence of user-defined fault/compensation handlers
 - Workflow instance and state of the recovery process (**unavailable at design time**)
 - Type of Web service operations provided by partner services
 - Existence of (repairable) compensating/equivalent activities

Repairability Factors (cntd.)

- Existence of compensating/equivalent activities



- Pre-defined fault/compensation handlers (e.g. BPEL) improve the possibility to repair activities
- Operations provided by partner services may not support compensation/substitution/retry

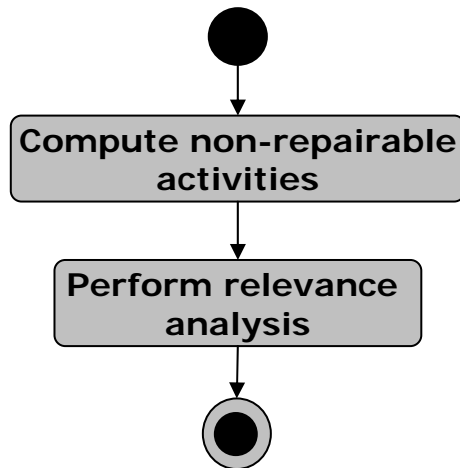
Types of Web service operations

Using a graph-based model of the choreographies supported by the Wf with every partner service it is possible to classify Web service operations and thus Wf activities.

- **Retriable:** the operation has no side effects neither on the workflow side nor on the service providing the operation
- **Compensatable:** these operations have side effects on either or both sides but they can be undone
- **Permanent:** their effects cannot be undone

Repairability Reasoning Algorithm (RR)

Basic steps of the algorithm:



Input:

- Extended Wf definition
 - Branching/failure probabilities
 - Equivalent/compensating activities
 - Pre-defined fault/compensation handlers
- Model of the supported choreographies
- Data dependencies

Output:

- Ranked/weighted list of non-repairable activities enriched with information about its repairability (e.g. #of equivalent activities, #pre-defined fault handlers, etc.)

Step 1: Computing non-repairable activities

Goal: determine whether a combination of repair actions (or pre-defined repair handlers) can be applied to an activity. If no such combination can be applied the activity is not repairable

- To determine the repairability of an activity the RR uses:
 - associated list of equivalent/compensating activities
 - associated pre-defined repair handlers
 - type of Ws operation used for implementing the activity (obtained from the choreography model)

- From this, rules for applicability of repair actions can be defined. Such rules capture **necessary but not sufficient conditions**

Rules for Repair Actions

- Rule for $\text{retry}(A)$:
 - if $\text{reliable}(A)$ then A is repairable. Repair pattern: $[\text{retry}(A)]$
 - if $\text{compensatable}(A)$ and, there is activity B : $\text{compensating}(A,B)$ then A is repairable. Repair pattern: $[\text{comp}(A,B), \text{retry}(A)]$
- Rule for $\text{compensate}(A)$:
 - if $\text{compensatable}(A)$ and, there is activity B : $\text{compensating}(A,B)$ then A is repairable. Repair pattern: $[\text{comp}(A,B), \text{retry}(A)]$
- Rule for $\text{substitute}(A)$:
 - if $\text{reliable}(A)$ and there is B : $\text{equivalent}(A,B)$ then A is repairable repair pattern: $[\text{sub}(A), \text{retry}(A)]$
 - if $\text{compensatable}(A)$ and, there is B : $\text{compensating}(A,B)$ and there is C : $\text{equivalent}(A,C)$ then A is repairable. Repair pattern $[\text{comp}(A), \text{sub}(A)]$

Step 2: Relevance Analysis

- Non-repairable activities are ordered according to their **branching and failure probabilities**
- Every activity is associated with a relevance index $ri(A)$, which is computed as follows:

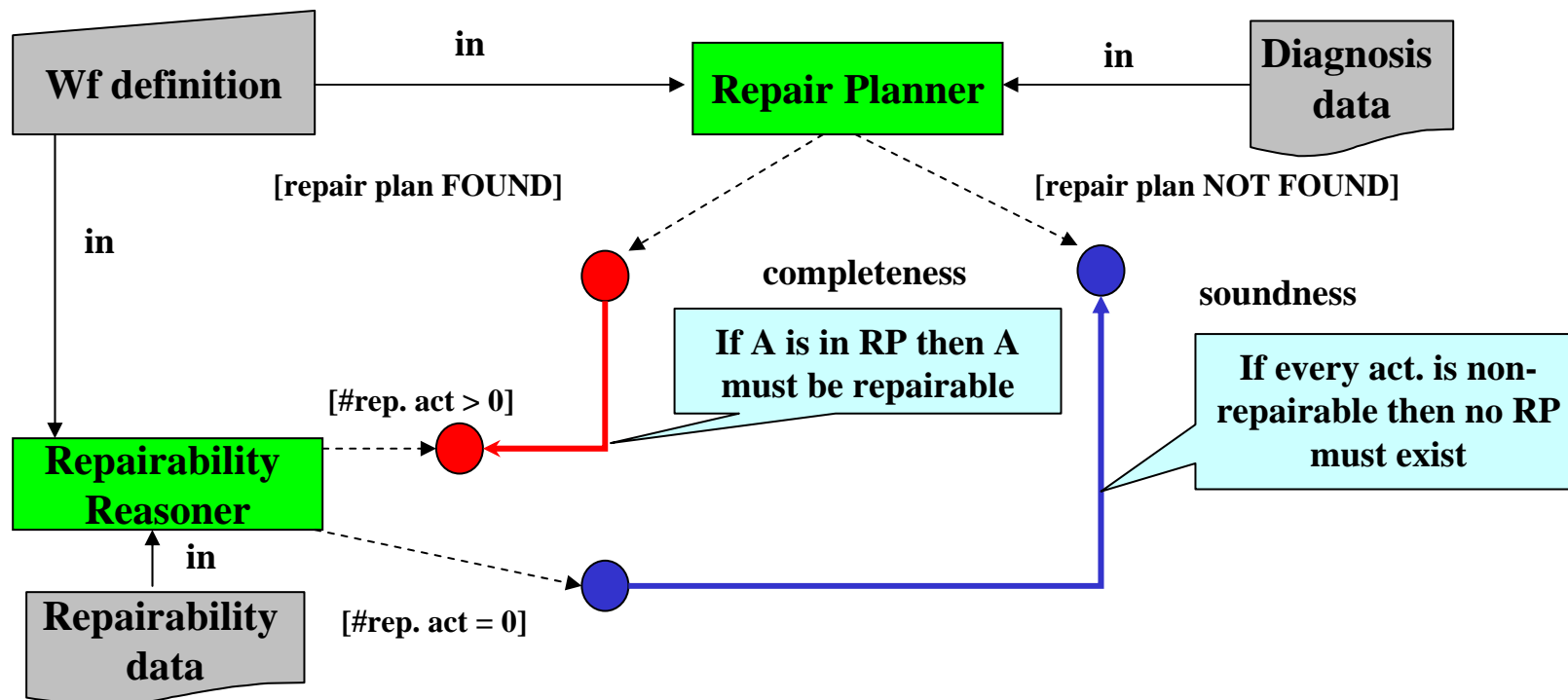
$$ri(A) = \text{branchingProbability}(A) * \text{failureProbability}(A)$$

Validation

- Correctness is defined in terms of soundness and completeness
- **Soundness:** The RR is **sound** if every time it concludes $\#rep.act. = 0$ (for a given Wf) the repair planner cannot generate a repair plan
- **Completeness:** the RR is **complete** if every time the repair planner generates a plan (P) for a given Wf the following holds:
 - *every activity for which P applies a repair action is regarded as repairable by the RR*

How to validate?

- The results of the RR are compared against the results obtained by running a repair planner



Experimental Settings

- We used a set of randomly generated Wf Schemas
 - random number of activities and variables
 - repairability data is randomly generated (e.g. failure and branching prob., equivalent/compensating activities)
 - random number of faulty instances with random number of faulty activities (diagnosis data)

Evaluation Results

	Wf. Schemas	Activities	Variables	XORs	Faulty act.	Wf. With RP	Complete	Non-rep. Wfs	Sound	
Test 1	40	12	20	2	4	6 (15%)	6 (100%)	3 (7.5%)	3 (100%)	
Test 2	50	20	20	4	2	21 (42%)	21 (100%)	0	N/A	
Test 3	100	10	20	2	4	19 (19%)	19 (100%)	6 (6%)	6 (100%)	
Test 4	150	12	20	2	4	17 (11.3%)	17 (100%)	7 (4,67%)	7 (100%)	
Test 5	1000	12	20	2	4	1000	1000 (100%)	52 (5.2%)	52 (100%)	

- Interpreting the results:
 - Test1: Repair planner found RP for only 6 (out of 40) Wfs. For each of these 6 the list of repairable activities returned by the RR is a superset of the list of repaired activities (completeness)
 - Test1: The RR found 3 (out of 40) Wfs contain an empty list of repairable activities. For each of these 3 the repair planner couldn't find a RP (soundness)

Conclusions

- A heuristic-based approach for determining the repairability of Wfs at design time was presented
- The algorithm considers different repairability factors and checks whether a Wf schema supports repairable execution of its activities
- It verifies that necessary but not sufficient conditions for repair action applicability are satisfied
- Designers can improve the repairability of a Wf by modifying its specification and adding compensating/equivalent activities or pre-defined repair handlers
- Evaluation results show the approach is sound and complete
- Further extensions may include the use of diagnosability information