A RISK REDUCTION FRAMEWORK FOR DYNAMIC WORKFLOWS

Prabhdeep Singh, Fatih Gelgi, Hasan Davulcu, Stephen S. Yau
Dept. of CSE
Arizona State University, AZ USA
Prabhdeep.Singh, fagelgi, hdavulcu, yau@asu.edu

Supratik Mukhopadhyay
Dept. of CSE
Utah State University, UT USA
supratik@cc.usu.edu

Abstract

Workflows tend to fail in real-world scenarios due to the uncertain/unreliable sensory information which sometimes needs to be updated during the execution of workflows. In a logic based framework, these dynamic predicates that can be updated are called non-monotonic predicates (NMPs). In this paper, we focus on reducing the risk of a given workflow due to the NMPs in that workflow. The main idea is to synthesize a backup workflow by augmenting the main workflow without introducing new NMPs. The backup workflow is generated by using expected values of NMPs if necessary instead of given values. The expected values are calculated from the execution history or provided by a domain expert. It is argued that total risk reduces to the square root of the main workflow itself.

1. Introduction

A workflow is a series of cooperating and coordinated activities. Since service-based systems usually consist of thousands of services, it is desirable that the systems can allow users to declaratively specify their goals, and automate the service composition based on the specified goals.

Consider a service-based system that connects the Police Departments (PD), Fire Departments (FD), and Ambulance Services (AMS) for coordinating various first responders (PD, FD and AMS) in handling traffic accident situations. PD, FD and AMS provide various capabilities as services in the system. The Accident Response Scenario given in Figure 1 illustrates an automatically generated workflow for coordinating field rescue operations according to the following dependencies:

1. A 911 call center gets a report specifying an accident at location L on a road, and informs a nearby Police Patrol Car (CAR), a Fire Engine (FE) and an Ambulance (AMB) about the accident. According to the initial report there are **two** injured passengers.

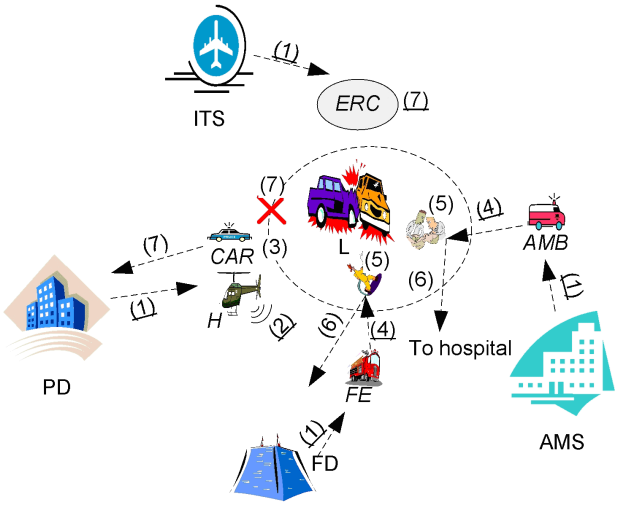

Figure 1. An accident-response scenario

2. The CAR, FE and AMB go to location L.
3. Upon arriving at L, the police officers set up a perimeter to secure the accident site, and inform the FE and AMB that the perimeter has been set up.
4. Upon arriving at L, the firefighters start to search for passengers involved in the accident, and rescue the passengers if they are trapped in the damaged vehicles.
5. Once the passengers are out of the damaged vehicles, the paramedics on the AMB assess the status of the injured passengers to decide the appropriate medical care for them.
6. After assessing the status of the injured passengers, the AMB takes the injured passengers to a nearby hospital.

When we consider real-world scenarios, such as the accident scenario above, workflows tend to fail due to uncertain or unreliable information mostly based on their sensing actions. For instance, during the search for passengers, it might be discovered that there are **four** injured passengers

instead of **two**. In the original workflow since there is only one ambulance allocated, it would not be possible to rescue all the injured passenger unless another available ambulance can be found and deployed to the accident location.

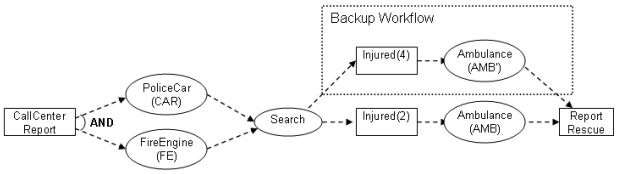

Figure 2. Overall goal for the accident response scenario service composition depicted as a control flow graph.

Figure 2 depicts a control flow graph representing a template of the service composition, which should be satisfied when responding to an accident. The control flow graph depicts the requirements that whenever the call center receives an accident report, it should notify a CAR and a FE, and then an AMB to rescue any injured passengers involved in an accident. If the information is updated to be four injured passengers then the system can activate the backup workflow to send another AMB' to the accident location.

In this paper, the focus will be on the non-monotonic predicates (NMPs) which might be updated during the execution of the workflow such as number of injured passengers. Updating NMPs is one of the major reasons for workflow failures. Main contribution of this paper is a Risk Management System for workflows that given a workflow it synthesizes a backup workflow which supports the main workflow and reduces the risk of failure to its square root.

Next section presents the architectural overview of the risk management system. The foundations for logic based modeling of workflows is presented in Section 3. Risk Management System is presented in Section 4 and its effects are discussed in Section 5. Related work is given in Section 6, and conclusion and future work is in Section 7.

2 Architectural Overview

A common architecture for the *Risk Management System* is presented in Figure 3. The *Workflow Synthesis* module generates the candidate workflows for Risk Management System to select the best work flow. Risk Management System is composed of three modules:

- **Risk Calculation**: Calculates the risk of each candidate workflow and returns the one which has minimum risk (denoted as WF^* in the figure). The risk calculation of a workflow is based on the probability of the workflow to fail due to it's NMPs. Probabilities might be obtained either from the execution history of the workflows or provided by a domain expert.

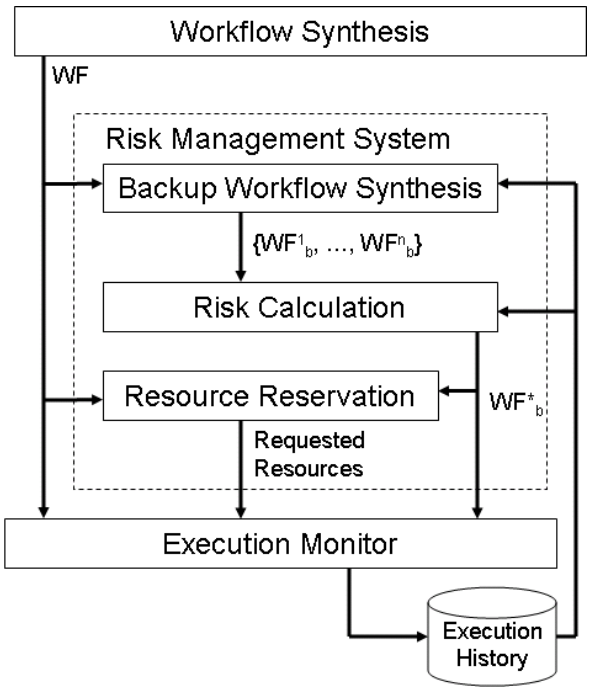

Figure 3. An overview of the architecture of the Risk Management System.

- **Backup Workflow Synthesis**: This module augments the main workflow WF^* without introducing any new NMPs, and generates a backup workflow to support it. The main idea to synthesize the backup workflow is to use expected values of NMPs if their current values are risky. Similarly, the risk and the expected values of NMPs are obtained either from the execution history of the workflows or determined by a domain expert. Hence the backup plan is at most as risky as the main workflow which is one of the main contributions of this paper.

- **Resource Reservation**: The backup resources which are not in the main plan but in the backup plan is determined by this module. Then reservations are requested to make the resources ready to use in case of a failure in the main plan due to the NMPs. That resource reservations make it available to switch to the backup plan if one of the NMPs fail in the main plan.

The details of the modules, calculations and algorithms will be presented in Section 4.

3 Logic Based Modeling of Workflows

In this section, we will present α-logic – a high level modal logic – which is expressive enough for modeling dynamic workflows. α-logic is a hybrid normal modal logic [2] for specifying service-based systems. The logic has both temporal modalities for expressing situation information as well as modalities for expressing communication, knowledge, and service invocation. Since α-logic is not the focus of this paper, we only intriduce parts of the syntax, semantics and proof system, which will be used in this paper here, and provide some intuition explanations for the logic.

3.1 Modeling Control Flow Graphs

We assume that every variable x has a type. Intuitively, the nominals act as identifiers to processes. The modality $serv(x; u; \sigma; \varphi)$ indicates that a process invokes service σ with parameter x, receives the individual named u as the result, and then satisfies φ. The formula $\langle u \rangle \varphi$ describes the behavior of a process after sending out the individual named u.

Since α-logic has connectives for services invocation, eventuality ($\Diamond$), parallel composition ($||$) and disjunction ($\vee$) as well as atomic constraints it is straightforward to model services composition goals as control flow graphs into α-logic statements. For example, the control flow graph in Figure 2 can be represented in α-logic as follows:

$\Diamond serv(L, T;' accident'; U; 911Callcenter)$
$\prec \Diamond serv(L, T;' CAR\_sent', CarID; X; PDAgent)$
$\prec (\Diamond serv(L, T;' FE\_sent', FeID; Y; FEAgent) \wedge$
$\Diamond serv(L, T;' AMB\_sent', AmbID; Y; AMBAgent))$
$\prec \Diamond serv(L, T;' rescued'; Z; 911CallCenter)$

3.2 Modeling Dependencies

α-logic can express a wide variety of temporal constraints. But here we focus on a relatively simple algebra of constraints for expressing the situational constraints identified in Section 4. Our algebra is as expressive as Singh's Event Algebra (Singh, 1995). These constraints are believed to be sufficient for the needs of services composition and workflow synthesis tasks, and their expressivity far beyond of the capabilities of the currently available commercial systems.

- *Atomic dependency*: $\Diamond serv(In; Out; Callee; Caller)$ indicates that a certain type of a service should eventually be invoked.
- *Negative atomic dependency*: $\neg\Diamond serv(In; Out; Callee; Caller)$ indicates that a service type should **not** be used during the rest of the composition.
- *Composite dependency*: Atomic situational constraints can be composed with conjunction ($\wedge$) and disjunction ($\vee$).
- *Existential dependency*: Situational constraints can be composed with implication ($\rightarrow$) to enforce insertion of additional services.
- *Ordering dependency*: Situational constraints can be composed with ordering ($\prec$) operator to specify that a pair of services should be invoked one after another.

For example, the following situational constraint related to the accident scenario captures the first situational requirement that in low visibility situation, upon arriving at the accident location, the police officers PD should first set up a perimeter to secure the accident site and then notify that the perimeter has been setup.

$serv(L, T; FE\_sent, low; PD; 911CC) \rightarrow$
$\Diamond serv(L, T; setup; PD; 911CC) \prec \Diamond < setup > PD$

Similarly the following ordering situational constraint captures the requirement that if there are critically injured passengers then a helicopter HELI should be requested by the AMB.

$serv(L, T;' critical'; U; AMB) \rightarrow$
$\Diamond serv(L, T;' helicopter\_sent', HeliID; U, AMB)$

3.3 Services Composition with α-Logic

In this section, we limit our attention to synthesis of non-iterative services compositions without loops. In general, the problem of constrained services composition and execution given a control flow graph and a set of dependencies requires that services are composed at design time and constraints are checked and enforced at run time alongside the execution of the control-flow graph. This may lead to unnecessary backtracking since each constraint violation at run-time requires that a new alternative execution path must be tried out. Also, such a strategy can not detect unsatisfiable requirements leading to plenty work done towards an achievable goal at run-time.

In this paper we propose a more efficient proof theory for the above classes of control flow graphs and depedencies that allows us to find executable services composition at design time.

Given an α-Logic formula G, describing a control flow graph and a set of dependencies C, as inputs our synthesis algorithm, called $Enforce$ produces a conjunction-free executable control flow specification G_C through a series of transformations. Our algorithm includes a proof procedure based on forward-chaining natural deduction presented in detail in the Background section to enforce existential constraints and a procedure to enforce ordering and other constraints. In order to enforce situational constraints at design

time, during each step of the forward chaining proof procedure, the left-hand side (or the body) of each situational constraint is evaluated using the current sequence of service composition. If the body of a constraint is evaluated to be true, the right-hand side (or the head) of the constraint is enforced using the procedure presented below.

- **Atomic dependency.** If M is new,
  Enforce($\Diamond serv(I; O; M; A), G) \equiv$
  $G||\Diamond serv(I; O; M; A)$

- **Composite dependencies.**
  Enforce(C1 $\wedge$ C2, G) $\equiv$
  Enforce(C2, Enforce(C1, G))

  Enforce(C1 $\vee$ C2, G) $\equiv$
  Enforce(C2, G) $\vee$ Enforce(C1, G)

- **Ordering dependency.**
  Enforce($\Diamond serv(I_1; O_1) \prec \Diamond serv(I_2; O_2)$, G) $\equiv$
  Synch(Enforce($\Diamond$ serv(I2;O2),Enforce($\Diamond serv(I1; O1)$, G)))

  where $Synch$ injects a send signal after every occurrence of $\Diamond serv(I_1; O_1)$ and a receive signal before every occurrence of $\Diamond serv(I_2; O_2)$

- **Existential dependency.**
  Enforce(C1 $\rightarrow$ C2, Goal) $\equiv$
  Enforce(C1, Goal) $\vdash$ Enforce(C2, Goal)

- **Negative atomic dependency.**
  Enforce($\neg\Diamond serv(I; O; M; A)$, G) $\equiv$
  Enforce($\Diamond serv(I; O; M; A) \rightarrow \perp$, G)

4 Risk Management for NMPs

The general framework for risk management system has three phases:

- Generate candidate backup workflows with the expected values of NMPs by augmenting the main workflow.

- Calculate the risk of each candidate backup plan, and select the best one as discussed in Section 4.3.

- Always do resource reservation in such a way that when the main workflow fails during execution, the scheduler can switch to the backup workflow utilizing the backup resources.

For the scenarios in the domains of our interest such as accident scenario, we consider the risk of the workflow as the probability of the workflow not to succeed. We will investigate the risk of the workflows in more detail below.

4.1 Case-based Analysis for the types of NMPs

We can categorize the variables of predicates into two forms:

- **Enumerated** values such as ambulance, helicopter.

- **Non-enumerated** values such as number of injured passengers, average speed of an ambulance.

Based on the category of the variables and the situation of being instantiated or un-instantiated, different problems may happen. In the following, we will give examples for each situation based on the accident-response scenario described above:

- *Case 1: Enumerated, instantiated*. In the scenario, suppose there are two injured passengers and their conditions are critical. During the rush hour, it is almost impossible for an ambulance to get to the accident location. Hence, it is more reasonable to send a helicopter instead of an ambulance. Here, the discrete instantiated non-monotonic predicate is `send(Ambulance)`. But, we need to re-instantiate it with 'Helicopter'; `send(Helicopter)`.

- *Case 2: Enumerated, un-instantiated*. Suppose the deductive proof includes the un-instantiated predicate `send(X)`. We know that X is a vehicle, but the question is, which vehicle is to be sent to the accident location.

- *Case 3: Non-enumerated, instantiated*. Suppose the capacity of an ambulance is 2 injuries and the number of injured passengers is initially reported as 2. By the constraints;

  ```
  capacity(Ambulance) = 2

  injured(Y), send(Ambulance,X),
       X>=Y/capacity(Ambulance)
  ```

  a workflow planner may generate the result;

  ```
  injured(2) => send(Ambulance,1)
  ```

  that is, only one ambulance is sent to the location. However, if the information of injured passengers is updated to 4, one ambulance won't be enough.

- *Case 4: Non-enumerated, un-instantiated*. In the scenario, another case is what we should do if the number of injured passengers is not known. In the urgent case, we have to do planning using the un-instantiated non-monotonic predicate injured(X).

4.2 Backup Workflow Synthesis

Due to the unreliability of the non-monotonic predicates, a good idea is to generate an alternative workflow to backup the main workflow. Assuming that the execution history gives us enough statistics, we synthesize a backup workflow by using the expected values of all the non-monotonic uninstantiated predicates and the non-monotonic predicates that are below the confidence levels with their given values. To keep the backup workflow consistent with the main workflow, the execution of the workflow is monitored. Whenever it is inconsistent with the backup workflow, the backup workflow is re-generated.

Consider the previous example and the main workflow is $Plan_2$. Suppose the expected number of injured passengers is 4 and the backup workflow is,

$Plan_b$: `injured(4), send(Ambulance2), send(Ambulance1).`

When the main workflow fails due to the number of injured passengers, we can switch to the backup workflow by just sending Ambulance2 to the accident location.

The idea of backup workflow is to generate a workflow that is executable in parallel with the main workflow. Let d be a given NMP and d' is the NMP of d with its expected value. Let $Before$ and $After$ be functions that returns the set of predicates that precedes and succeeds d in the original workflow:

- $Before(d, WF) \prec d'$,
- $d' \prec After(d, WF)$,

We can then use the $Enforce$ method presented earlier to synthesize the backup workflow. Here $\prec$ is a transitive relation, i.e, if $(a \prec b) \wedge (b \prec c) \rightarrow (a \prec c)$. In other words, upon its synthesis d' shouldn't violate any ordering constraints of d. Let D be the dependency set of the main workflow. Then the dependency set of backup workflow D' should augment the dependencies of the main workflow by inheriting the ordering dependencies of d for d'. That is,

$$D' = D \cup \{x \prec d' | x \prec d \in D\} \cup \{d' \prec x | d \prec x \in D\}$$

The next step is to find the sub-workflow WF_s that is related to d' which has the root as the least common ancestor of the predicates that appears together in any dependency of d'. All the descendants of that root is in WF_s. Then,

$$Enforce(d \wedge d', WF_s)$$

generates candidate the backup workflow one of which will be selected based on risk calculations as given in the next section. The given method is sound since the generated backup workflow is always consistent with the dependencies and does not introduce new NMPs into the workflow.

4.3 Workflow Risk Calculation

To evaluate the risk of a workflow, first we assume that there exists a workflow execution history or a domain expert. Our idea is to calculate the probability of the workflow to be unsuccessful by using history as the statistics.

For each NMP, we obtain a probability distribution over the values either discrete or continuous. We also assume that the characteristic of the probability distribution for each NMP is provided by the domain expert such as normal distribution, uniform distribution etc. Then the risk of each non-monotonic instantiated predicate in a workflow can be calculated. The source of the information is also very important in risk calculations. For instance, in the accident scenario, the police report must be much more accurate and has more probability to be correct than an anonymous report.

Definition 4.1. Let π be a workflow, and N_π be the set of non-monotonic predicates in π. The risk of the non-monotonic predicate $n(X, \{S\}) \in N$ with source S is defined as,

$$R_{n(X,\{S\})} = 1 - P[n(X, \{S\})]$$

where and $P[n(X)]$ is the probability of n with the value X in its distribution. Note that the argument S is presented when S is applicable.

We will examine the total risk of the workflow through analysis of four cases;

1. **Unconditional workflows**: Simple workflows, only a sequence of actions.
2. **Conditional workflows**: Workflows that contain decision points / conditional actions.
3. **Alternating workflows**: Workflows that contain alternative branches to accomplish the goal.
4. **Parallel workflows**: All the branches in the workflow have to be accomplished.

4.3.1 Unconditional Workflows

Analogous to the intersection of the risks, we consider the total risk of the workflow as the product of the risks of the non-monotonic predicates in that workflow. We assume independence among non-monotonic predicates for simplicity, efficiency and scalability.

Theorem 4.1. *Let π be a workflow, and N_π be the set of non-monotonic predicates in π. The risk of the overall workflow R_π is defined as,*

$$R_\pi = 1 - \prod_{n(X,\{S\}) \in N_\pi} 1 - R_{n(X,\{S\})}.$$

4.3.2 Conditional Workflows

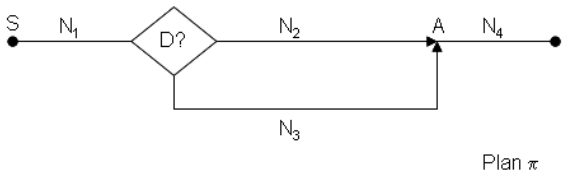

Figure 4. An instance of a conditional workflow.

For conditional workflows, consider the workflow π in Figure 4. The workflow starts at S and finishes at G. D is a decision point that workflow splits into two branches, and A is the reunion point[1]. N_1, N_2, N_3 and N_4 are the fragments of the workflow. N_1 is the fragment between start and the decision point. N_2 and N_3 are the branches from decision point to the reunion point. Lastly, N_4 is the fragment between the reunion point and the goal. We have two cases for the decision point D; D might contain non-monotonic predicates or not.

Case 1: D *does not* have any non-monotonic predicates, that is, the probability of 'D happens' is equal to 'D does not happen'. From Theorem 4.1, we can calculate the risk of each fragment by multiplying the risks of non-monotonic predicates in that fragment. Let R_i be the risk of the fragment $i \in \{1, 2, 3, 4\}$, then,

$$R_i = 1 - \prod_{n(X,\{S\}) \in N_i} 1 - R_{n(X,\{S\})}.$$

Considering each branch as an event, the branches are mutually exclusive since either one of the branches is executed. By the independence assumption and the equal probability of the decision point, the total risk of the branches, R_d will be,

$$R_d = (R_2 + R_3)/2.$$

We can generalize the above result by,

Theorem 4.2. *Let a workflow π has a decision point that yields k branches and reunions at the same point. Let N_i be the set of non-monotonic predicates in the branch i. Then the total risk of all the branches, R_d is,*

$$R_d = \frac{1}{k} \sum_{i=1}^{k} R_i$$

[1]Note that the reunion point has to exist for conditional workflows; at worst it would be the goal state since the goal is fixed in a workflow and the branches have to lead to the same goal

Proof. Workflow fail means the probability of all the branches to fail. That is,

$$R_d = \sum_{i=1}^{k} \frac{1 - P(N_i)}{k} = \frac{1}{k} \sum_{i=1}^{k} R_i$$

where $P(N_i)$ is the probability of N_i to succeed.

□

Case 2: D does have any non-monotonic predicates, that is, the probability of 'D happens' may not be equal to 'D does not happen'. Let R_c is the risk of D, then the only difference with the previous case is, we add the risk of the branch that D happens with $1 - R_c$ and the other branch with R_c.

$$R_d = 1 - (1 - R_c)(1 - R_2) - R_c(1 - R_3).$$

4.3.3 Alternating Workflows

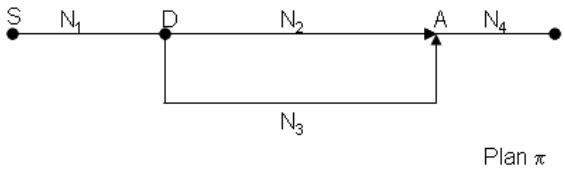

Figure 5. An instance of a alternating workflow.

Consider the workflow π in Figure 5 which has an alternating point at D, and A is the reunion point. Either N_2 or N_3 has to be successfully executed for workflow to succeed. By independence assumption, the total risk of the branches, R_d will be,

$$R_d = (1 - P(N_2)).(1 - P(N_3)) = R_2 R_3$$

where $P(N_i)$ is the probability of N_i to succeed. We can generalize the above result by,

Theorem 4.3. *Let a workflow has an alternating point that yields k branches and reunions at the same point. Let Ni be the set of non-monotonic predicates in the branch i. Then the total risk of all the branches, R_d is,*

$$R_d = \prod_{i=1}^{k} R_i.$$

Proof. The probability of all the branches to fail is

$$R_d = \prod_{i=1}^{k} 1 - P(N_i) = \prod_{i=1}^{k} R_i.$$

□

4.3.4 Parallel Workflows

(analogous to the intersection of the risks) For the same figure above, consider D is the splitting point for the parallel workflows. Since both of the branches have to be successful the total risk will be,

$$R_d = 1 - (1 - R_2)(1 - R_3).$$

From the result we can infer,

Theorem 4.4. *Let a workflow π has a decision point that yields k branches and reunions at the same point. Let N_i be the set of non-monotonic predicates in the branch i. Then the total risk of all the branches, R_d is,*

$$R_d = 1 - \prod_{i=1}^{k} 1 - R_i.$$

Proof. The probability of at least one branch to fail is,

$$R_d = 1 - \prod_{i=1}^{k} 1 - R_i = 1 - \prod_{i=1}^{k} P(N_i).$$

□

In short, the idea is *to multiply the probabilities in serial branches, and to take the intersection of the risks in parallel branches* analogous to the set intersection and union operations. We can also use combinations of the above workflow types and calculate their risks.

Since we would like to generate safe workflows, all the non-monotonic predicates in the generated workflow should also satisfy a certain pre-defined *confidence level*. It may be either discrete, continuous, instantiated or un-instantiated, for each non-monotonic predicate, we identify a *confidence interval* based on the calculated probability distribution and the characteristics of distribution.

Definition 4.2. Let C_n be the confidence level for the non-monotonic predicate n, and the confidence interval for n be $I_n = [a, b]$. Then the following equation holds,

$$\sum_{X} P(n(X)) \geq C_n$$

for enumerating and discrete variables,

$$\int_{a}^{b} P(n(X))dx \geq C_n$$

for continuous variables.

If a non-monotonic predicate in a workflow is instantiated, and below the confidence interval, then we consider that the workflow is not safe, hence not qualified even it has very low overall risk. The crucial reason to eliminate the workflow is that, the risk of one of the non-monotonic predicates might be too high to cause to fail the workflow. To make the idea of confidence level clear, consider the scenario: there is an accident with 2 or 3 passengers injured. According to the statistics the probability of injured passengers to be 2 and 3 is 0.86 and 0.99 respectively. AMS has two ambulances with the distances of 10 and 5 miles to the accident location, and the capacities of 3 and 2 passengers respectively. The accident is known to be serious, and the ambulance has to arrive at the location in 5 minutes. That means the first ambulance must drive with the average speed of 120 mph and the second must have 60 mph. The probabilities of the given speeds for the ambulances are 0.75 and 0.95. Assume that we can use only one of the ambulances.

Rule: arrivalTime = distance(X) / avgSpeed(X)

Constraint: `arrivalTime <= 5 min`

Plan1: `injured(3), send(Ambulance1), arrivalTime = 5 min`
`Risk1 = 1 - (1-Rinjured(3)) * (1-RavgSpeed(Ambulance1)=120) = 1 - 0.99 * 0.75 = 0.2575`

Plan2: `injured(2), send(Ambulance2), arrivalTime = 5 min`
`Risk2 = 1 - (1-Rinjured(2)) * (1-RavgSpeed(Ambulance2)=60) = 1 - 0.86 * 0.95 = 0.2604`

According to overall risk calculations, a rescue workflow using the first ambulance will have lower risk than the sending the second ambulance. However, it is not likely that the first ambulance can travel at the speed it is required yielding a more probable failure. On the other hand, each NMP of $Plan_2$ has low risk factor which makes the overall workflow safer.

5 Effect of Risk Management

In this section, we will show that risk management with backup plan synthesis is an effective method. Our risk management system reduces the risk of the main plan to its square root.

Theorem 5.1. *Let π_a and π_b be the main plan and the backup plan respectively. Then*

$$R_{\pi a} + R_{\pi b} \leq R_{\pi a}^2$$

Proof. From backup plan synthesis, πb contains at most same amount of non-monotonic predicates with πa. The

risk of each non-monotonic predicate in πb is less than or equal to the ones in πa since we use expected values or the value in πa if it is better. Hence, $R_{\pi b} \leq R_{\pi a}$. That leads,

$$R_{\pi a \wedge \pi b} = R_{\pi a}.R_{\pi b} \leq R^2_{\pi a}.$$

□

Consider the example below;

$Plan_a$: `injured(2), send(Ambulance2)`

$Plan_b$: `injured(4), send(Ambulance2), send(Ambulance1)`

Suppose $R_{Plan_a} = 0.05$ and $R_{Plan_b} = 0.03$

Then the overall risk reduces to,

$$R_{Plan_a \wedge Plan_b} = 0.05 * 0.03 = 0.0015 \leq R_{Plana_{a^2}}.$$

6 Related Work

Three most common frameworks for specifying service composition requirements are control flow graphs [10, 8], triggers [3] (also known as event-condition-action rules), and situational temporal constraints [1, 4]. The control flow graph is most appropriate for depicting the local execution dependencies between the key activities in a service composition. It is a good way to visualize the overall flow of control between the milestones that a workflow should satisfy. Control flow graphs are the primary specification means in most commercial implementations of workflow management systems. A typical graph specifies the initial and the final activities in a workflow, the successor-activities for each activity in the graph, and whether these successors must all be executed concurrently, or it suffices to execute just one branch non-deterministically.

[9] is the seminal paper of non-monotonic planning. A theory of planning that uses non-monotonic reasoning on the modal quantification logic Z is developed. It does forward reasoning and backward planning to reach the goal. The proposed theory uses frame axioms and the modal quantification logic Z to propagate the facts from the current situation to the next situation. The method states the most obvious consequences of each action, and it can not use domain knowledge or history to handle failures due to the non-monotonic predicates.

The idea of confidence level and degree of belief goes back to the papers [5, 6, 7]. In [5], Hawthorne described a range of non-monotonic conditionals that behave like conditional probability functions at various levels of probabilistic support. These conditionals were defined as semantic relations on an object language for sentential logic. Hawthorne extends the work to the most prominent family of these conditionals to a language for predicate logic in [6]. When the evidence is uncertain, the resulting degrees of belief in Bayesian updating appear to be sensitive to the order in which the uncertain evidence is acquired, a rather un-Bayesian looking effect. In [7] Hawthorne explores three models of sequential updating, the usual extension and two alternatives. He establishes necessary and sufficient conditions for order-independent updating, and shows that extended rigidity is closely related to these conditions.

7 Conclusion and Future Work

In this paper, we presented a risk management system for a given workflow that synthesizes a backup workflow to reduce the risk of the main workflow to at least its square root. A modal logic based framework is presented for managing the dependencies of the system.

References

[1] P. C. Attie, M. P. Singh, A. P. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings of the 19th Conference on Very Large Databases*, pages 134–145, 1993.

[2] P. Blackburn, M. deRijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2003.

[3] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 204–214, New York, NY, USA, 1990. ACM Press.

[4] E. A. Emerson. Temporal and modal logic. pages 995–1072, 1990.

[5] J. Hawthorne. On the logic of nonmonotonic conditionals and conditional probabilities. *Journal of Philosophical Logic*, 25(2):185–218, 1996.

[6] J. Hawthorne. On the logic of nonmonotonic conditionals and conditional probabilities: Predicate logic. *Journal of Philosophical Logic*, 27(1):1–34, 1998.

[7] J. Hawthorne. Three models of sequential belief updating on uncertain evidence. *Journal of Philosophical Logic*, 33(1):89–123, 2004.

[8] H. Hsu. Special issue on workflow and extended transaction systems. *Bulletin of the IEEE Technical Committee on Data Engineering*, 16(2), 1993.

[9] S. S. Hundal and F. M. Brown. A theory of nonmonotonic planning. In *CSC '91: Proceedings of the 19th annual conference on Computer Science*, pages 247–254, New York, NY, USA, 1991. ACM Press.

[10] WFMC. Workflow management coalition. terminology and glossary. *Retrieved December 4, 2005 from http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf*, 1996.