An Architecture for Supporting Network Fault Recovery Management

Feng Liu, Antonis M. Hadjiantonis, Ha Manh Tran, and Mina Amin

MNM Team, Ludwig-Maximilians-University Munich, Germany
liufeng@nm.ifi.lmu.de
Centre for Communications Systems Research, University of Surrey, UK
{a.hadjiantonis,m.amin}@surrey.ac.uk
Computer Science, Jacobs University Bremen, Germany
h.tran@jacobs-university.de

Abstract. Highly available and resilient networks play a decisive role in today's networked world. As network faults are inevitable and networks are becoming constantly intricate, finding effective fault recovery solutions in a timely manner is becoming a challenging task for administrators. Therefore, an automated mechanism to support fault resolution is essential towards efficient fault handling process. In this paper we propose an architecture to support automated fault recovery in terms of traffic engineering, recovery knowledge discovery and automated recovery planning. We base our discussion on an application scenario for recovery from border router failure to maintain optimized configuration of outbound inter-domain traffic.

Key words: Fault Management, Fault Recovery, Automated Planning, Policy-Based Management, Case-Based Reasoning, Peer-to-Peer, Inter-Domain, Traffic Engineering

1 Introduction

Availability of networks is becoming more essential in today's networked world. Whereas network faults are inevitable facts, an efficient fault management process is decisive to decrease the fault resolution time and to increase the availability of the network. In case of failure, an efficient fault recovery process is expected to find fault resolutions quickly and recover the impacted network in a timely manner. Hence, recognizing the importance of minimum system downtime to maintain compliance with accepted Service Level Agreements (SLA), we propose a dual stage fault recovery process. The first stage is a short-term system reaction to minimize the immediate effects of a fault. In order to allow for quick system response, this first reaction is pre-planned in sets of recovery policies able to anticipate faults. The second stage is the long-term recovery plan and aims to discover the recovery knowledge and plans the recovery process.

The presented architecture involves traffic engineering, policy-based management and artificial intelligence approaches. Our research focuses on providing

a comprehensive framework to facilitate an automated fault recovery process in large-scale networks. The motivations of the architecture are: (1) providing short-term recovery plans through fast recovery solutions, (2) providing long-term recovery plans through an efficient recovery knowledge discovery approach and (3) automating recovery planning for the long-term recovery process.

Application Scenario: To demonstrate the applicability of our approach, we choose a scenario related to inter-domain Traffic Engineering (TE) and examine a case study of automated recovery from an egress point (EP) failure. Since inter-domain links are the most common bottlenecks in the Internet [1], an efficient plan for fault recovery is necessary. We focus our interest on planning the recovery from border router faults to maintain optimized configuration of outbound inter-domain traffic. Once a border router (EP) fails, we follow a dual stage recovery process that will be detailed in section 4.

The first stage is to switch affected traffic flows to another EP, while at the same time optimize the EP selection (border router selection) for all outbound inter-domain traffic of a domain. To achieve a quick reaction that would minimize disruption, we execute in advance an outbound TE algorithm and store short-term recovery plans in a repository. The algorithm is designed with the goal of inter-domain link load-balancing and creates configuration sets for each case of the EP failures. In addition, the algorithm creates the initial configuration set for normal operation which is used until a failure is detected. Having responded to the border router failure and minimizing short-term disruption, the second recovery stage begins to discover a long-term solution and recover from the failure. Once notified about the failure, the recovery system relies on proposed policies and actions to decide an appropriate recovery plan. More specifically, the planning subsystem receives high-level directives and actions as input, then combines the input with received state information from the monitoring system in order to generate the appropriate sequence of actions for remedying the failed EP and recovering normal network operation.

This paper is organized as follows: in section 2 we present our proposed architecture and discuss the involved subsystems and their interactions. Section 3 summarizes the methodologies applied to our approaches and in section 4 we analyze our system based on the application scenario. Section 5 provides an overview on the related work and the paper concludes with section 6.

2 System Architecture

Our proposed system contains three subsystems: Automated Recovery Planning (ARP), Knowledge Discovery (KD) and Policy-based Management (PBM), see Fig.1. The planning subsystem obtains status reports from the monitoring component in order to carry out analyzing and executing recovery plans for the managed system with the support of the other subsystems that provide appropriate actions and policies respectively. The introduction of each subsystem is presented in the following subsections.

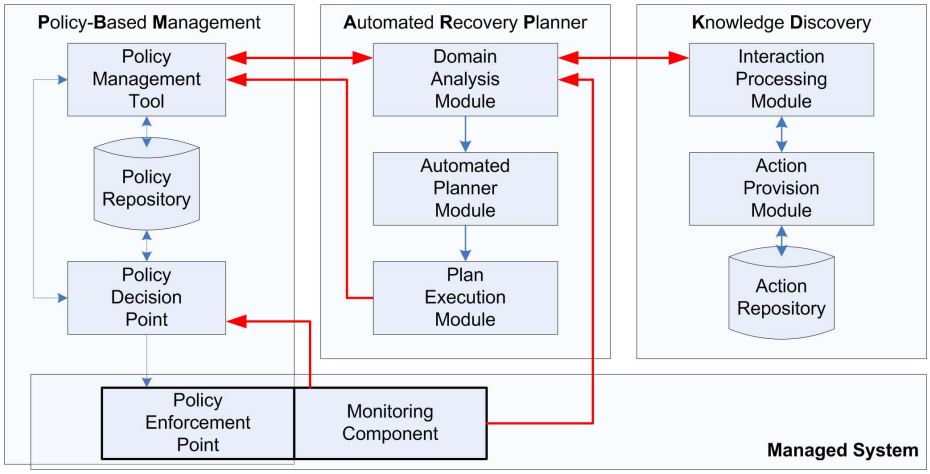

Fig. 1. Overview of System Architecture

Automated Recovery Planning: This central subsystem provides fault recovery plans for the managed system. Its tasks involve analyzing the status report from the monitoring component, collecting relevant information from the other subsystems for the recovery activities, producing relevant recovery plans and provisioning for plan executions on the managed system. In Fig. 1, the three modules responsible for these tasks include:

Domain Analysis (DA): This central module connects to `KD`, `PBM` subsystems and the monitoring component for aggregating the relevant information as plan knowledge to facilitate the operation of the automated planner module. Such plan knowledge includes the description on the current system state, e.g. which components are impacted by the failures and which components are still operational etc. Depending on the impacted components, it contacts `KD` for a set of relevant recovery measurements attached with meta-information. The meta-information describes under which situations a particular solution step could be applied and what consequences could be expected from it, i.e. how a particular solution step is going to affect the current system state. `DA` also contacts `PBM` to retrieve recovery policies.

Automated Planner (AP): This module derives recovery plans based on plan domain descriptions provided by `DA`. It contains an automated planning algorithm, which correlates relevant recovery knowledge and produces recovery plans by reasoning. Note that the selected planning algorithm is domain-independent and generic. The rationale to use domain knowledge as input is to accelerate the planning process and to increase the efficiency of the planning algorithm.

Plan Execution (PE): This module makes provision for the recovery plan executions. Plans generated by `AP` are described in a specific plan description language, therefore, to enable the plan executions, it is necessary to convert the format of the plan and map the recovery plan into executable actions. Since `PE` relies on `PBM` for plan execution, it converts the generated plan into a PBM-compatible format and sends the plan to `PBM`, where the mapping and execution

of the plan take place. The results will be observed by the monitoring component and sent back to DA to see if further recovery measurements are necessary.

Knowledge Discovery: Case-Based Reasoning (CBR) [2] resolves a problem by searching for similar problems and reasoning on their solutions found in the past. The reasoning capability of a conventional CBR system is restricted by a local case database. A distributed CBR system takes advantage of computation power and problem-solving knowledge at various sites, thus improving performance and maintaining huge federated case databases better. This system exploits the integrated framework of Peer-to-Peer (P2P) and CBR [3], which involve retrieving problems and inferring solutions, respectively.

This subsystem acts as a distributed CBR system to provide actions for recovery plans. Its tasks involve communicating with various knowledge sources (e.g. the Internet, the operators), dealing with various requests from and responses to other modules, inferring proper actions corresponding to the failed status of the managed system and managing a case database (e.g. failure cases and actions). These tasks belong to three modules, shown in Fig. 1:

Action Repository (AR): This module involves storing cases and maintaining the case database. It regularly checks the usage of cases and the similarity of cases in order to deactivate obsolete cases or consolidate cases. Maintaining the case database is essential since the CBR system tends to be lumbering and inefficient with a large number of cases.

Action Provision: This module serves as an independent CBR engine which takes requests and the case database as input to provide actions. Its main tasks include (i) retrieving similar cases from AR, (ii) reasoning on these cases to figure out the most promising case, and (iii) updating the case database. To improve AR, the module regularly updates new cases from various sources including adapting instructions from operators to solutions, or updating cases by learning problems from monitoring systems, or extracting cases from the response of peers.

Interaction Processing: This module is responsible for establishing and maintaining a P2P network. Its main tasks are to interact with various sources including operators, surveillance systems or other peers for information exchange, resource search and lookup. Particularly, it deals with requests from peers forwarded to other peers, instructions and updates from the operators processed to update AR.

Policy-based Management: PBM simplifies the complex management tasks of large scale systems, since high-level policies can be automatically enforced as appropriate network management [4]. In general, policies are defined as Event-Condition-Action (ECA) clauses, where on event(s) E, if condition(s) C is true, then action(s) A is executed. The components of PBM are shown in Fig. 1 along with their interactions with ARP and the managed system. The four functional elements, as defined by IETF, are described below:

Policy Management Tool (PMT): PMT is the interface between the operators and PBM. It allows the introduction and editing of policies and also provides

notification about critical events requiring a manager's attention. Using this interface, the operators carry out the specification of operational policies by selecting the appropriate policies from the supported policy types and selecting the required parameters. We extend PMT's functionality by interfacing it with ARP. PMT provides relevant policies to ARP to assist the planning process.

Policy Repository (PR): PR encapsulates the management logic to be enforced on all networked entities, as expressed in policies. It is the central point where policies are stored by managers using PMT and can be subsequently retrieved either by the Policy Decision Point (PDP) or by PMT.

Policy Decision Point (PDP): PDP is responsible for evaluating policy conditions and deciding when and where policy actions need to be enforced. Once relevant policies have been retrieved from PR, they are interpreted and PDP in turn provisions any decisions or actions to the controlled Policy Enforcement Point (PEP). PDP receives input from the monitoring component of PEP to form a closed control loop.

Policy Enforcement Point (PEP): PEP enforces policy decisions, as instructed by PDP. Within our framework, PEP is enhanced by the *monitoring component* that reports local information to assist decision making.

The *monitoring component* is integrated to provide feedback to both PBM and ARP for decision making. It is not formally part of our architecture since fault detection and diagnosis are out of the scope of this paper. Critical events such as failures are reported to both subsystems in order to initiate the dual stage recovery process.

3 Methodology

In this section, we give detailed views on the methodologies the subsystems apply. The discussion focuses not only on individual subsystems, but also tries to provide an overview on the interactions between them.

3.1 PBM and ARP Collaboration

As networks become more and more complex, unavoidably faults occur more frequently. It is evident that frameworks with automated recovery capabilities can significantly expedite and simplify management tasks. Within our framework, policies work in two layers to express on one hand high-level business objectives and on the other hand, low-level configuration policies to anticipate faults. Policies can encapsulate the overall management logic at two functional layers: the business layer and the device layer. Using sophisticated algorithms, we translate high-level policies defined at the business layer to low-level configuration policies at the device layer. At the same time, the PBM interacts with ARP to provide policies and constraints to be used in the planning procedure.

By executing an outbound TE algorithm in advance, we create different policy sets that express those high-level goals in normal operation and in addition anticipate possible system failures. These low level policies, constitute short-term

plans for the EP selection aiming for the recovery of outbound inter-domain traffic when an EP fails. The created recovery policy sets are stored in the Policy Repository and the policy set for normal operation is enforced. An outbound inter-Autonomous System (AS) TE algorithm that is able to optimize the EP selection (border router selection) for outbound inter-domain traffic of a domain has been developed in [5]. The algorithm is designed with the goal of inter-domain link load balancing. The output is |L|+1 (|L| number of border routers) sets of EP selection configurations, one set for normal operation (i.e. no failure) that can be called PrimaryEgressPoint selection and one set for each case of EP failures that can be called BackupEgressPoint selections. In other words, PrimaryEgressPoint selection determines EP selection under no inter-domain link failure and BackupEgressPoint selection determines EP selection under Failure States. A common implementation of EP selection is by adjusting the local-preference value in the Border Gateway Protocol (BGP) route attribute [6]. According to the BGP route selection process, local-preference has the highest priority and its value indicates the preference of the route. The higher the local-preference, the more preferred is the route. The algorithm takes inter-domain connectivity, BGP routing information, inter-domain traffic matrix as inputs and then deterministically calculates the EP selection through local search heuristics. The reader is referred to [5] for more details.

High-level policies are used to express goals of inter-domain traffic management, e.g. *"optimize EP selection to achieve inter-domain load balancing"*, or *"optimize EP selection avoiding routing traffic to destination prefix p1 using domain D2"*. For security reasons we may wish to avoid routing the inter-domain traffic flows towards some particular destinations through some specific domains. The benefit of policy-based management is the ability to use different policies to achieve different goals, e.g. minimize peering cost. Furthermore, high-level or recovery policies could also be used by `ARP` in order to find the recovery plans which are in compliance with the management goals. It is possible that the fault recovery plan composition may result in several alternative plans to reach the same goal. In such cases, `ARP` should refer to policies in order to choose the right one which complies with the management objectives. A recovery policy can be interpreted as constraints for plan composition. Considering the previous example on EP optimisation, obviously a link failure recovery plan that involves the step which uses domain $D2$ to reach destination prefix $p1$ could not be considered as valid, because there is a management policy which forbids using this step. In this case, `ARP` has to search for other alternatives. Another example of using high-level policies to recover from router failure, assuming there are alternative steps involved to recover the router failure: one of them involves using router B as temporal fallback router, to which the traffic could be shifted to, while another step requires recalculating and applying the alternative links to route the current traffic. Obviously, both steps lead to same goal, namely, keeping the disruption on the current traffic to minimum. Nevertheless, the fallback solution maybe preferred in terms of shorter traffic disruption compared to link recalculation. If we assume that there is a policy that states *Router B can only*

be used as temporal failover for A between time 20:00 - 23:00 and the link failure on router A happens on $10:00$, apparently, the first alternative cannot be considered by the automated planner, because the constraint derived from the high-level policy restricts the usage of router B as failover router at the time of failure on A.

3.2 KD and ARP Collaboration

`KD` contains a P2P network for knowledge exchange and update. A peer shares resources (e.g. failure cases and actions) with other peers and provides search facilities for other peers. A peer also bears a CBR engine for choosing best solutions. The CBR engine, upon receiving requests from `ARP`, works on the case database to propose solutions for `ARP`. The design of `KD` below explains core issues related to building P2P network, case retrieval and reasoning in CBR regarding the focused problem domain. Some detail has been addressed in our related studies.

Communication: `KD` uses a Gnutella-style backbone network of super peers as an appropriate overlay [7]. The overlay sticks to the characteristics of Gnutella super-peer networks. Such networks organize peers into several clusters, any cluster contains peers connected to a super peer, and the connections among super peers form an unstructured overlay network. Any peer communicates with its super peer for sending queries, publishing resources or receiving answers. Any super peer, upon joining the overlay, has to perform several non-trivial tasks such as search and lookup, reasoning, or maintenance.

Case Retrieval: This issue involves case representation and similarity function. A case including failures and actions is represented by field-value vectors, where values are binary, numeric or symbolic. In particular, a case contains a vector v_f with symptoms sym_i for failure descriptions, a number of vectors v_a with an action act and conditions cnd_i for action descriptions, and a goal vector v_g with symptoms sym_i for verification descriptions. The following example explains how a case is represented:

- $v_f = <$ sym_1: link_to_dest_failed, sym_2: no_traffic_flow, sym_3: primary_router_not_running, sym_4: second_brouter_exist, sym_5: second_brouter_ok, sym_6: bandwidth_reserv_required>
- $v_g = <$ sym_1: link_to_dest_ok, sym_2: traffic_flow_ok, sym_3: bandwidth_ok>
- $v_a = <$ act: use_alternative_link, cnd_1: link_to_dest_failed>
- $v_a = <$ act: failover_to_secondary_router, cnd_1: second_brouter_exist, cnd_2: second_brouter_ok>
- $v_a = <$ act: establish_link>
- $v_a = <$ act: reserve_bandwidth, cnd_1: bandwidth_reserv_required>

Note that a request sent to the subsystem only contains v_f and v_g, the subsystem works on symptoms appeared in the request by communicating with other peers and looking into the local case database for similar symptoms and actions. Using this representation, evaluating case similarity is straightforward

since it depends on comparing pairs of symptom and value; values avoid using textual fuzzy descriptions and the expert determines the weight values of symptoms. Similar cases are evaluated by the *global similarity* method $sim(r, c) = \sum_{i=1}^{n} w_i sim(r_i, c_i)$, where n is the number of matched symptoms; r_i and c_i are symptoms of request r and case c, respectively; $sim(r_i, c_i)$ is the distance between r_i and c_i, w_i is a weight value of the i^{th} feature such that $\sum_{i=1}^{n} w_i = 1$ with $w_i \in [0, 1]$ $\forall i$.

Case Reasoning: This issue involves case adaptation (or reasoning, inference) and decision making. The conventional inference process carries out distinguishing a retrieved case from the problem to clarify main differences, modifying the retrieved case following the differences to obtain the final case. There are several alternative inference methods depending on problem domains, such as probabilistic inference using Bayesian network, classification using machine learning, optimization technique using genetic algorithms (GA). For the focused problem domain, the reasoning engine in `KD` employs optimization techniques to search for an optimal set of actions that satisfies symptoms from the request provided constraints from the goal vector and conditions from the action vectors.

To enable the planning process, `ARP` needs to aggregate the recovery knowledge from `KD`. After receiving the fault report from the monitoring component, the domain analysis module dispatches the knowledge search request regarding the impacted component to the interaction processing module of `KD`. The search results are returned by `KD` in the field-value vectors format. They include knowledge such as fault symptoms and solution steps. However, the results cannot be directly used by planner algorithm, the domain analysis module needs to convert the received data into some specific planning language, such as Plan Domain Description Language (PDDL) [8]. In the next section, we present examples on the representation of recovery domain knowledge.

As mentioned in section 2, the core of this subsystem is the automated planner module. This module utilizes the AI-based planning algorithm to generate plans accordingly to the domain description. A planning problem is based on the restricted state-transition system[9]. A state-transition system Σ could be represented as $\Sigma = (S, A, E, \gamma)$, where $S = \{s_0, s_1, s_2, \ldots\}$ is the finite set of states; $A = \{a_1, a_2, \ldots\}$ is set of actions; $E = \{e_1, e_2, \ldots\}$ is set of events and $\gamma : S \times A \times E \rightarrow 2^S$ denotes a state transition function.

A planning problem can be defined as a triple $P = (\Sigma, s_0, g)$, where $s_0 \in S$ is the initial state of a problem and $g \subset S$ is the set of states which satisfy the goal. Provided with problem domain descriptions, a planning algorithm operates on the provided information and finds one or more plans accordingly. Additionally, a planning algorithm observes different constraints if they are provided. The use of constraints serves two purposes: first, it reduces the plan searching space and second, it guides the plan searching in a correct way, e.g. some of the actions should be avoided according to constraints (policies).

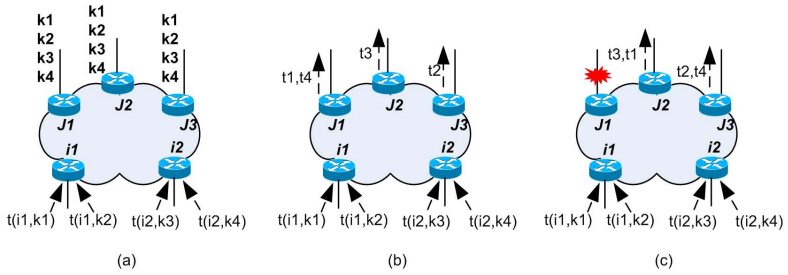

Fig. 2. Scenario Topology:(a) EP selection problem inputs,(b) PrimaryEgressPoint selection,(c) BackupEgressPoint selection if j1 fails

4 System Analysis - Case study

An outbound inter-domain TE scenario is chosen to investigate recovery from EP failure using the introduced architecture. Normally, upon an EP failure, traffic is shifted to another available EP in accordance to the BGP route selection policies. However, if a large amount of traffic is shifted, congestion is likely to occur on these new serving EPs. An intuitive approach to minimize this congestion is to redirect the traffic to another EP by adjusting BGP routing policies in an online manner until the best available EP has been found. Such online trial-and-error approach, however, may cause router misconfiguration, unpredicted traffic disruption and BGP route flooding, leading to route instability. It is desirable to have an efficient fault recovery plan and an optimization algorithm in order to proactively produce a configuration for optimal performance under normal and failure scenarios. Therefore, we focus our interest on outbound inter-domain TE and recovery planning in case of any EP failure. Once one of the EP fails, our recovery plan follows the proposed dual stage recovery process. First, `PBM` enforces appropriate policies (Table 2,P1-3) and a short-term solution is used to minimize disruption. Then `ARP`, through its interaction with `KD`, attempts to discover a long-term solution to recover from the fault. The topology for the described scenario is shown in Fig. 2.

4.1 Automated Recovery using Outbound TE Algorithm

Based on the proposed architecture, the first step after a failure is detected is to react with a short-term solution to minimize disruption. This solution is preplanned by executing an EP selection algorithm in advance and storing its output as policies on the Policy Repository. The algorithm's output is based on a generated synthetic inter-domain traffic matrix. The traffic matrix consists of a set of inter-domain traffic flows that originates from each ingress point towards each of the considered destination prefixes. Each inter-domain traffic flow is associated with a randomly generated bandwidth demand according to uniform distribution. We use a sample topology and traffic flow to demonstrate the applicability of our recovery methodology. Details about the algorithm and evaluation results can be found in [5].

Table 1. Assignment of Primary and Backup Egress Point selection

Prefix	Primary	Backup		
		if J1 fails	if J2 fails	if J3 fails
k1	J1	J2	J1	J1
k2	J3	J3	J3	J2
k3	J2	J2	J3	J2
k4	J1	J3	J1	J1

For a better understanding of our EP selection algorithm, we provide an example in Fig. 2. Figure2(a) illustrates the inputs for the EP selection problem, comprising ingress routers i1 and i2, EP j1, j2 and j3, inter-domain traffic demands $t1 = t(i1, k1)$, $t2 = t(i1, k2)$, $t3 = t(i2, k3)$ and $t4 = t(i2, k4)$. The destination prefixes k1, k2, k3 and k4 that can each be reached through all the three EP are also shown. Recall that the task of the EP selection problem is to determine, for each destination prefix, both a PrimaryEgressPoint to be used under no failure and a BackupEgressPoint to be used when its PrimaryEgressPoint has failed. Figure2(b) shows a potential solution of PrimaryEgressPoint selection, where t1 and t4 reach their destination prefix k1 and k4 respectively through EP j1, t2 reaches its destination prefix k2 through EP j3 and t3 reaches its destination prefix k3 through EP j2. This assignment corresponds to Table 1 column 2. In addition, Figure2(c) illustrates a potential solution of BackupEgressPoint selection when EP j1 has failed. As shown, t1 has been re-assigned to EP j2 to reach k1 and t4 has been re-assigned to EP j3 to reach k4 as their BackupEgressPoint. This assignment corresponds to Table 1 column 3 and according to the high-level goal, this solution achieves inter-domain link load balancing. To implement this solution, e.g. for prefix k1 the largest value of BGP local-preference, e.g. 100, should be assigned to its selected PrimaryEgressPoint (i.e. EP j1), the second largest value, e.g. 80, should be assigned to its selected BackupEgressPoint (i.e. EP j2) and any BGP local-preference value less than 80, e.g. 50, can be assigned to the remained EP (i.e. EP j3).Also for prefix k4 the largest value of BGP local-preference, e.g. 100, should be assigned to its selected PrimaryEgressPoint (i.e. EP j1), the second largest value, e.g. 80, should be assigned to its selected BackupEgressPoint (i.e. EP j3) and any BGP local-preference value less than 80, e.g. 50, can be assigned to the remained EP (i.e. EP j2).Moreover since the other two prefixes reachable through j1 (i.e. k2 and k3) are assigned to j1 for neither Primary nor Backup, their BGP local preference should be any value less than 80 e.g. 50. Table 2 shows the assignment of BGP local-preference setting for prefixes k1 to k4 on all EPs. These configuration settings are enforced on the EPs by their PEP based on policies (P1,P2,P3). An initial configuration policy (P0) is used by the network administrator, to configure the proper values for BGP local-preference. The benefit of combining a TE algorithm with a PBM approach is the creation of a flexible management environment able to quickly react to failures. In parallel, our framework automatically works to recover from the failure and recovery policies are used as input to the `ARP` subsystem. With

Table 2. Local-pref setting and policies for prefixes on Egress Points

Egress Point	Prefix	BGP local preference	Policies (Event==setupEP()
J1			*if* (EP==j1)
	k1	100	*then*[(set-local-pref(k1)=Prim-val)
	k2	50	(set-local-pref(k2)=Low-val)
	k3	50	(set-local-pref(k3)=Low-val)
	k4	100	(set-local-pref(k4)=Prim-val)][$P1$]
J2			*if* (EP==j2)
	k1	80	*then*[(set-local-pref(k1)=Back-val)
	k2	80	(set-local-pref(k2)=Back-val)
	k3	100	(set-local-pref(k3)=Prim-val)
	k4	50	(set-local-pref(k4)=Low-val)][$P2$]
J3			*if* (EP==j3)
	k1	50	*then*[(set-local-pref(k1)=Low-val)
	k2	100	(set-local-pref(k2)=Prim-val)
	k3	80	(set-local-pref(k3)=Back-val)
	k4	80	(set-local-pref(k4)=Back-val)][$P2$]
Initial configuration policy			
Event==BGP-conf *if* (-) *then* [(set-local-prefs([Prim-val,Back-val,Low-val],[100,80,50])) ,gen-event-setupEP(j1,j2,j3)] [P0]			

the cooperation of `KD`, `ARP` outputs a new recovery plan that `PBM` can enforce to the network and reinstate normal operation.

4.2 Automated Recovery using Planning Algorithm

To show how the planning and knowledge discovery subsystems collaborate, we present here an example based on the aforementioned border router (EP) failure scenario. The objective of collaboration between the two subsystems is to produce long-term fault recovery solutions. After the router failure is recognized, the first stage recovery procedure is activated in order to sustain the current traffic and minimize the disruption of the failure. Whereas such a solution can be regarded as a short-term measure, a long-term recovery measure is still needed to completely recover from the failure and reinstate normal operation.

The long-term recovery process is activated by the reports from then monitoring component. The domain analysis module analyzes the monitoring report and extract the current global state. The current global state includes the information on the impacted component and other relevant information. Note that the granularity of monitoring information will affect the later planning process. For example, a report states *The primary router is impacted by failure and it has OS version 2.3* will be more useful for planner to compose a better plan than just states *The primary router is impacted.* The global states are represented by the set of predicates.

Algo. 1: Automated Planning

Input: O: set of actions
s_0, g: initial state, goal state
Output: π sequence of actions

```
1  s ← s_0
2  π ← ∅
3  while True do
4     if s satisfies g then
5        return π
6     A ← {a|a ∈ O & precond(a) true in s}
7     applicable ← A
8     if applicable= ∅ then
9        return failure
10    choose a ∈ applicable
11    s ← γ(s, a)
12    π ← π.a
```

Algo. 2: Action Optimization

Input: C: set of retrieved cases c_i;
R, τ: request with symptoms r_j, threshold
Output: σ optimal set of actions

```
1  Σ ← ∅
2  Σ* ← ∅
3  for each c_i ∈ C & r_j ∈ R do
4        σ ← {a|a ∈ c_i & a satisfies any r_j ∈ R}
5        Σ ← Σ ∪ σ
6  while True do
7     for each σ ∈ Σ do
8        evaluate f(σ)
9        if σ is optimal then
10          return σ
11    Σ* ← {σ|σ ∈ Σ & f(σ) > τ }
12    Σ ← {σ|σ generalized by σ* & σ* ∈ Σ*}
```

After the failure source is known, the domain analysis module dispatches a search request to `KD` in order to find possible resolution steps to recover the component. `KD` collects solutions from various sources, then runs the reasoning engine on the retrieved solutions before returning the proposed actions to the analysis module. The reasoning engine uses the optimization technique based on the GA algorithm to provide an optimal set of actions, see Algo. 2. The discovered recovery actions have to be translated into a planning-specific language. For example, if a recovery action involves upgrading the router OS, this action could be described in PDDL [8] as:

```
(:action fetch_update
   :parameters(?r - router ?p - patch ?cv - currentversion
               ?nv - latest version )
   :precondition( and ( (failed ?r) (patch_at ?r ?p ?cv)
            ( < (?cv ?nv)  ) ) )
   :effect( and ( (updated ?r ?nv ?p) (= (?cv ?nv) ) ) ) )
```

The `parameter` field denotes which parameters are needed for this action. The field `precondition` describes the conditions, under which the `fetch_update` action could be applied. The `effect` field denotes the consequence of this action, i.e. how this action is going to effect the global state. The question mark denotes the variables. The initial state, goal state and recovery options formulate a planning domain. The automated planner operates on the planning domain for one or more feasible plans. The planning algorithm[9] is described by Algo. 1. Note that only those actions ($a \in O$) which cause state changes and the current state s are considered as parameters of γ function in this algorithm, events are left out for the sake of simplicity.

Recovery actions are selected based on their preconditions and current global state. Each action leads the current global state into a new state, this iteration finishes when the new state equals goal state. The generated plan is a sequence of actions which transit the initial state into the goal state.

5 Related Work

The study of Mark et al. [10] has proposed an automatic system for finding known software problems. The system matches the symptoms of the current problems with the symptom database to find the closest matches. The matching algorithm is designed to work with structured symptoms that contain program call stack, not arbitrary data.

The recent study of Stefania et al. [11] has proposed a CBR system for self-healing in software systems. The description of the system lacks some details. Any case is represented in features which contain binary and symbolic values. Cases are retrieved by using the k-NN algorithm; where the similarity distance function evaluates case features with the corresponding weight values, but the weight function for features is not provided. The system mainly depends on the retrieval process to classify problems.

Policy-Based Management (PBM) simplifies the complex management tasks of large scale systems, since high-level policies monitor the network and automatically enforce appropriate actions [4, 12, 13]. Industry and PBM have been closely related in autonomic computing and self-management approaches [14]. The main advantage which makes a policy-based system attractive is the functionality to add controlled programmability in the management system without compromising its overall security and integrity. Policies can be viewed as the means to extend the functionality of a system dynamically and in real time in combination with its pre-existing hard-wired management logic [4, 13]. Policies are introduced to the system and parameterized in real time, based on management goals and gathered information. Policy decisions generate appropriate actions on the fly to realize and enforce those goals.

Outbound inter-Autonomous System (AS) Traffic Engineering [5, 6, 1] is a set of techniques for controlling inter-AS traffic exiting an AS by assigning the traffic to the best egress points (i.e. routers or links from which the traffic is forwarded to adjacent ASes towards destinations. The general problem formulation of outbound TE is: given the network topology, BGP routing information and inter-domain Traffic Matrix (TM), determine the best Egress Point (EP) for each traffic demand so as to optimize the overall network performance,such as inter-AS link load balancing [1].

Srivastava et al. [15] discussed the feasibility and theoretical aspects on using planning methods in autonomic computing. They concluded that automated planning is an evolutionary next step for autonomic systems that possess the self-managing capabilities. Kephart [14] addresses in his paper the challenges of using AI-based planning methods to the autonomic computing. Arshad et al. [16] presented a planning based recovery system for distributed system. However the proposed approach has several disadvantages; e.g. the recovery knowledge elicitation is not considered and lack of details in many aspects of applying automated planning methods in fault recovery.

6 Conclusion

Motivated by the need for an efficient fault recovery management, we have presented our initial efforts towards an integrated architecture. By combining the strengths of three paradigms: Automated Recovery Planning, Knowledge Discovery and Policy-based Management, we attempt to provide an automated framework. We have demonstrated a dual stage recovery process based on a case study of border router (EP) failure, aiming to maintain optimized configuration of outbound inter-domain traffic and quickly reinstate normal operation.

Beyond initial architecture design, we intent to investigate further interactions among subsystems and define generic and reusable interfaces. This will allow the extension of the architecture to a variety of case studies. We will aim to increase the automation of recovery and plan execution, thus minimizing human intervention and recovery times. Our future work will focus on intelligent planning algorithms that will combine system knowledge and business goals, aiming to gradually migrate to fully automated fault recovery management.

Acknowledgments The work reported in this paper is supported by the EC IST-EMANICS Network of Excellence (#26854).

References

1. T. Bressoud, R. Rastogi, and M. Smith. Optimal configuration for bgp route selection. In *Proc. IEEE INFOCOM*, 2003.
2. A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
3. Ha Manh Tran and Jürgen Schönwälder. Distributed Case-Based Reasoning for Fault Management. In *Proc. 1st International Conference on Autonomous Infrastructure, Management and Security*, pages 200–203. Springer-Verlag, 2007.
4. D.C. Verma. Simplifying network administration using policy-based management. *IEEE Network*, 16(2), 2002.
5. M. Amin, K. Ho, M. Howarth, and G. Pavlou. An integrated network management framework for inter-domain outbound traffic engineering. In *Proc. IEEE/IFIP Management of Multimedia and Mobile Networks and Services (MMNS2006)*, 2006.
6. N. Feamster, J. Borkenhagen, and J. Rexford. Guidelines for interdomain traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 33(5):19–30, 2003.
7. H. M. Tran and J. Schönwälder. Heuristic Search using a Feedback Scheme in Unstructured Peer-to-Peer Networks. In *Proc. 5th International Workshop on Databases, Information Systems and P2P Computing*. Springer-Verlag, 2007.
8. D. McDermott et al. Pddl - the planning domain definition language. 1998.
9. Dana Nau, Paolo Traverso, and Malik Ghallab. *Automated Planning - Theory and Practic*. Morgan Kaufmann, 2004.
10. M. Brodie, S. Ma, G. Lohman, T. Syeda-Mahmood, L. Mignet, N. Modani, J. Champlin, and P. Sohn. Quickly finding known software problems via automated symptom matching. In *Proc. 2nd International Conference on Automatic Computing*, pages 101–110, Washington, DC, USA, 2005. IEEE Computer Society.
11. S. Montani and C. Anglano. Case-based reasoning for autonomous service failure diagnosis and remediation in software systems. In *Proc. 8th European Conference on Case-Based Reasoning*, pages 489–503. Springer-Verlag, 2006.
12. A.M. Hadjiantonis, M. Charalambides, and G. Pavlou. A policy-based approach for managing ubiquitous networks in urban spaces. In *Proc. IEEE International Conference on Communications (ICC2007)*, 2007.
13. P. Flegkas, P. Trimintzios, and G. Pavlou. A policy-based quality of service management system for ip diffserv networks. *IEEE Network*, 16(2), 2002.
14. Jeffrey O. Kephart. Research challenges of autonomic computing. In *Proc. 27th International Conference on Software Engineering (ICSE '05)*. ACM, 2005.
15. B. Srivastava and S. Kambhampati. The case for automated planning in autonomic computing. IEEE, 2005.
16. N. Arshad, D. Heimbigner, and A. L. Wolf. A planning based approach to failure recovery in distributed systems. In *Proc. 1st ACM SIGSOFT workshop on Self-managed systems*, pages 8–12, New York, NY, USA, 2004. ACM.