Towards an Optimized Model of Incident Ticket Correlation

Patricia Marcu
Munich Network Management Team
Leibniz Supercomputing Center
Boltzmannstr. 1, 85748 Garching, Germany
marcu@mnm-team.org

Genady Grabarnik, Laura Luan
Daniela Rosu, Larisa Shwartz, Chris Ward
IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
genady, luan, drosu, lshwart, cw1@us.ibm.com

***Keywords*—incident ticket, incident management**

***Abstract*—In recent years, IT Service Management (ITSM) has become one of the most researched areas of IT. Incident and Problem Management are two of the Service Operation processes in the IT Infrastructure Library (ITIL). These two processes aim to recognize, log, isolate and correct errors which occur in the environment and disrupt the delivery of services. Incident Management and Problem Management form the basis of the tooling provided by an Incident Ticket Systems (ITS).**

In an ITS system, seemingly unrelated tickets created by end users and monitoring systems can coexist and have the same root cause. The connection between failed resource and malfunctioning services is not realized automatically, but often established manually by means of human intervention. This need for human involvement reduces productivity. The introduction of automation would increase productivity and therefore reduce the cost of incident resolution

In this paper, we propose a model to correlate incident tickets based on three criteria. First, we employ a category-based correlation that relies on matching service identifiers with associated resource identifiers, using similarity rules. Secondly, we correlate the configuration items which are critical to the failed service with the earlier identified resource tickets in order to optimize the topological comparison. Finally, we augment scheduled resource data collection with constraint adaptive probing to minimize the correlation interval for temporally correlated tickets. We present experimental data in support of our proposed correlation model.

I. Introduction

IT Service Management (ITSM) has become a significant research area in the IT in the past few years as IT service providers have focused on the development of methodologies and tools that help provide high quality service with maximal efficiency. An important component of ITSM is Incident and Problem Management which provides mechanisms to recognize, isolate, correct and log problems and incidents which occur in a system and disturb the service provisioning.

The IT Infrastructure Library (ITIL), best practice in managing information technology (IT) infrastructure, development, and operations [1] addresses, IT Incident Management along with other processes related to IT Service Operation. Incident Management is the process that deals with incidents, defined in ITIL terminology as ”An unplanned interruption to an IT service or reduction in the quality of an IT service. Failure of a configuration item that has not yet impacted service is also an incident”[2].

The Incident Management Process is supported by various tools including Incident Ticket Systems (ITS). These are software systems used in an organization to record information about service failures or malfunctions and about the interventions made by technical support staff or third parties on behalf of the end user who reported the incident. This record is called *ticket*. Tickets can also be automatically issued by monitoring systems in response to degradation of the vital signs of the monitored IT system. Monitoring systems are deployed across computing infrastructure for proactive management of system-health and service quality. Upon detection of predefined conditions, the monitoring systems trigger events that automatically generate tickets.

While monitoring systems are useful tools for ITSM, it is unrealistic to expect that all elements of the IT infrastructure will be constantly monitored. Typically in a large data center monitoring of critical resources is done periodically. The variation of the monitoring interval depends on criticality and stability of the resource. For some resources the monitoring is set up to be triggered manually in order not to overload the network.

A ticket created by a monitoring system typically provides information from the point of view of the low level resource on which the service is based, such as reports of server failures or overload, or network router failure. Thus in an ITS coexist two categories of related tickets, namely tickets from the end user and the tickets from the monitoring system but relationships between them are not immediately identified. The link between tickets is typically realized manually but this, often, is an expensive process in terms of manpower and productivity. However as our study shows, it is critical for effective incident management to identify tickets which are redundant or potentially have the same root cause.

Ticket correlation must happen at the time of ticket creation in order to isolate the cause for a ticket reported by end-user as well as to support problem determination and root cause analysis. Accuracy in correlation of the tickets creates advantages for the user as well as for service provider. It contributes to the faster resolution of the ticket and the service provider enjoys a higher efficiency in root cause analysis, and problem determination, while, at the same time, keeps costs and resource utilization on a low level.

In this paper, we propose a novel method for correlation of the tickets reported by end users with those reported by monitoring systems regarding resource problems. In section II we review previous work in the area. Our multi-stage correlation process has a few advantages over related work, First, class-based filtering and initial focus on the critical resources for the failed service are meant to speedup the process by limiting the likelihood of expensive CMDB searches. Second, the adaptive resource polling increases the quality of the results by limiting the impact of the time lag in receiving monitoring tickets. Section III presents a motivating example. The model and the method for ticket correlation are described in Section IV. Section V has formalization and experimental results that validate the proposed correlation model. Conclusion and further work are provided in Section VI.

II. Related Work

This section reviews prior research related to the correlation of trouble ticket/symptoms/events for Incident and Problem Management and fault diagnosis.

In a seminal work related to fault diagnosis in integrated network and system management [3] Dreo proposes the use of trouble-ticket correlation for discovery of tickets and access to problem-solving expertise. Dreo argues that good models for the functional and topological (i.e., resource mapping) aspects of a service are key elements for high-quality correlation. In this paper, we use novel models for such dimensions of correlation as topology and time, namely the topology aspects are modeled by Configuration Management Database (CMDB) relationships; and the temporal aspects are handled with flexibility, based on constraint adaptive resource polling. In addition we employ a category-based correlation.

[4] proposes an algorithm for event correlation, extended in [5], based on the same service model as in [3]. Events are correlated for root-cause analysis using Rule-Based Reasoning (RBR) and active probing. While we follow a similar approach, we use a novel set of RBR-rules and use adaptive probing (an enhanced concept of active probing) to trigger the creation of relevant resource tickets.

[6] proposes a system for self-improvement help desk service that uses Case-Based Reasoning (CBR). This techniques emphases the importance of searching through the descriptions of a ticket. [7] describes a similar approach using RBR techniques for discovering the historical and predictive value of trouble ticket data. Both these approaches use keyword search. The likelihood of incorrect correlation results is relatively high because, often the highly relevant keywords are hard to determine.

Gupta et al. propose ([8]) an automated algorithm for correlating incoming incident with configuration items of the CMDB based on a keyword search of the CMDB. This algorithm can be used in our work to reduce the overhead of CMDB search.

Adaptive probing techniques [9], [10] use a measurement technique that allows fast on-line inference about current system state via active selection of only a small number of most informative probes. In our proposal, we use this technique to trigger the generation of relevant resource tickets. We augment this technique with a innovation required by a constraint on the overall duration of probing execution and on the number of probes running simultaneously.

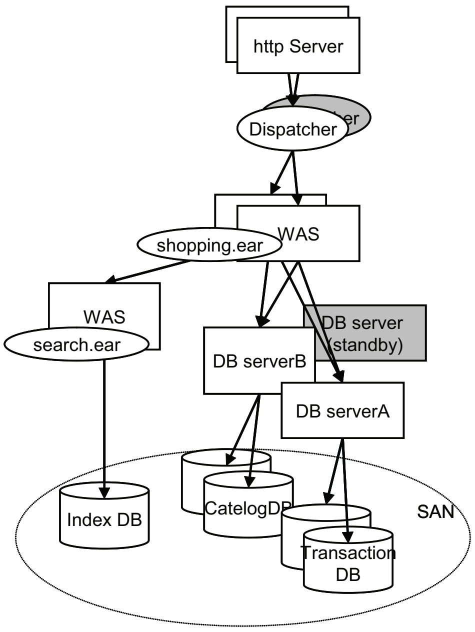

Fig. 1. Shopping Cart and Search Catalog e-commerce Service Realization

[11] proposes the use of the relationship between managed objects, as defined in CMDB, to correlate symptom events occurring in an event storm towards determining the root cause of the problem. While exploiting similar object relationships, our approach also uses additional service-specification details towards improving the accuracy and response time of the event correlator.

[12] exploits the relationships captured in CMDB regarding services, components and users to determine the impact of network outages on services and users. Namely, metadata in the network packets blocked by an outage identify the services and users immediately affected and CMDB relationships help determine the further impact.

III. Motivation

a) Insights from a large corporate account: Our work is motivated by an analysis of trouble tickets for a large corporate account. The account comprises a large variety of computing systems, ranging from personal computers to clusters of servers and up to mainframes. This infrastructure supports a large range of services ranging from personal-computing to enterprise services (like email) to business services (like application-service provider).

We consider a volume of over 6.5 million trouble tickets created in a 2.5 year period. We discover that multiple types of monitoring tools are employed. Some monitoring tools focus on system and application vitals. Samples system vitals include CPU and file system utilization, network interface status and file sizes. Sample application vitals include web application servlet response time, JDBC call response time and database table space utilization. In the context of this work, the tickets generated by these tools are considered 'resource tickets'.

For detailed insights related to the correlation of resource- and user-tickets such as relative volume and arrival patterns we focus our analysis on one of the largest organizations in the corporate account and select a period of 30 days with a relatively high number of resource tickets. We analyzed a base of about 16,000 tickets, of which about 900 are resource tickets. Out of the remaining trouble tickets about 100 were service tickets, while the rest related to workstation and personal account management. We identify several relevant challenges for the problem of ticket correlation:

- Handle delayed delivery of resource tickets, due to the specifics and configuration of the monitoring tools. This motivates our approach for additional resource pooling during the ticket correlation process.
- Handle a large number of redundant tickets. Redundancy is mainly observed for resource tickets and is caused by the use of threshold-based policies for notification of potential critical situations. Once the system vital reaches the threshold, tickets are generated periodically until the situation is cleared. Therefore, the time spent with manual analysis of redundant tickets can be relatively high, which motivates the need for automation of ticket correlation.
- Handle repeated service tickets at varied time distances from the related resource tickets. Service tickets can arrive within a few minutes from the relevant resource ticket, or after one or more days, while the root cause is being solved.

b) Motivating example: Figure 1 depicts a tiered J2EE enterprise application deployment includes front-end http servers, request dispatchers, WebSphere Application Servers (WAS), and back-end database servers. Multiple instances of *http* servers and *WAS* servers are used for load sharing. Standby servers are configured for fail-over protection of the request dispatcher and the database servers. The databases reside in a storage system and are connected via *SAN* to the database servers.

The e-commerce application is packaged as the enterprise archive file *shopping.ear*, which includes shopping cart and catalog search services. The shopping cart service employs two databases, one for the catalog records and shopping transaction records. Each database is deployed on a different database server for security and performance reasons. The catalog search service is a search engine, packaged as the enterprise archive *search.ear* and deployed on a different server from the shopping cart service. The deployed application accesses the index database in order to serve the search requests from *shopping.ear*.

Figure 2 gives a detailed view of a configuration of the systems depicted in Figure 1. This view is derived from the configuration data available in the configuration management system. It visualizes the relevant system artifacts (the circles)

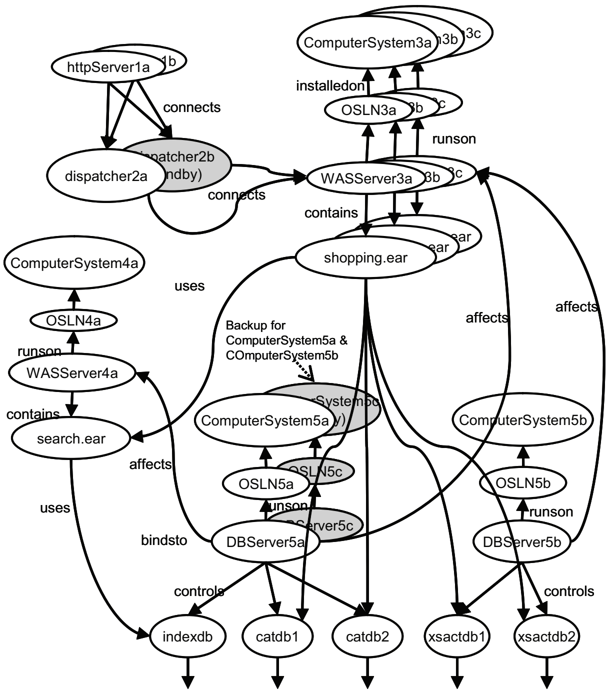

Fig. 2. Service System Mapping

and their relationships (the arrows and annotations). For instance, the *WAS* server in Figure 1 is represented by three configuration items:

- a computer system (e.g. *ComputerSystem3a*)
- an operating system (e.g. *OSLN3a*) which has an "installedon" relationship to the computer system, and
- a WAS server (e.g. *WASServer3a*) with a "runson" relationship to the operation system

Other note-worthy relationships are:

- the database servers affect the *WAS* servers, e.g. *DBServer5a* has an "affects" relationship with the WEB servers *WASServer3a, WASServer3b, WASServer3c, and WASServer4a;*
- the databases reside on the *SAN*, therefore they have "resideson" relationships to the storage subsystems in the SAN;
- storage subsystems are mounted to the operating systems in which the database servers run on; therefore they have a "bindsto" relationship to the operating systems;
- the applications use the databases, e.g. *search.ear* has a "uses" relationship with the *indexdb*.

The information of Figure 2 is used in the following sections to demonstrate how the proposed algorithms correlate end-user tickets with system generated tickets in order to help identify root causes.

IV. Models and Algorithm for Correlation of Incident Tickets

A. Concepts and definitions

This section introduces the concepts used in the design of our novel algorithm for ticket correlation. Namely, we formalize the concepts of ticket, service and configuration item (related and critical CIs) and also introduce Constraint Adaptive Probing.

A generic ticket definition: A ticket is a record of an incident, including all pieces of information related to the incident, such as the reporter of the incident (person or software component), the date of reporting, what priority the incident has, the person assigned to work on the resolution, the current ticket status, and other details.

Tickets are classified as 1) *resource tickets*, when they are reported by the monitoring system and 2) *service tickets*, when they are opened by an end-user, representing the users' perceived experience of a service.

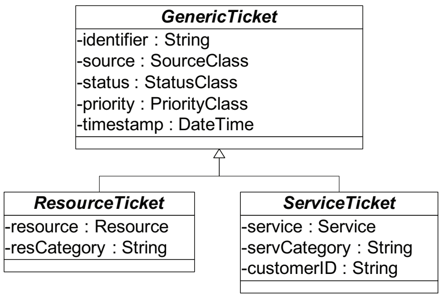

Fig. 3. Ticket Class Hierarchy

In Figure 3 the class hierarchy of tickets is given. The two classes **ResourceTicket** and **ServiceTicket**, representing the two kinds of tickets mentioned before are subclasses of the **GenericTicket** class.

The GenericTicket class has the following attributes:

- ***identifier***, which typically is a string, representing the unique reference (or case) number for the incident report;
- ***source***, with possible values *resource* and *service*, identifies the origin of the ticket as resource-based monitoring or end-user service, respectively.

The attributes ***status***, ***priority*** and ***timestamp*** are irrelevant to this discussion.

Additional to these attributes, the classes **ResourceTicket** and **ServiceTicket** have their specific attributes. The class **ResourceTicket** has two more attributes:

- ***resource***, which is a unique identifier for the affected resource; and
- ***resCategory***, which is a unique identifier for the resource category (see details in the section related to service model).

The class **ServiceTicket** has three specific attributes:

- ***service***, which is the service unique identifier for the service that the end user has a problem with;
- ***servCategory***, which is the unique identifier for the service category; and
- ***customerID***, which identifies the end-user that is experiencing service problems.

Table I shows a sample of a resource ticket and one of a service ticket, with all the related attributes defined.

	Resource Ticket	Service Ticket
Attribute	Value	Value
identifier	320054D	453999
source	resource	service
status	pending	new
priority	medium	high
timestamp	081320081245	08132008928
resource	indexdb	shopCatalog
resCategory	*HW/Server/WAS/indexdb*	
service		*shopping cart*
servCategory		*SW/webAppl/searchCatalog*
customerID		A2816AB

TABLE I
EXAMPLE OF RESOURCE AND SERVICE TICKETS

Service definition: The service definition is specified by the provider in a service catalog. The service definition has two important parts: *service category* and *similarity (matching) rules*. Typically categories are defined from the point of view of the customer. The similarity rules relate the service to an abstract representation of the infrastructure (resource) component. These would be provided as part of the service definition in the service product offering and developed during service design. The similarity rule is one innovation in our approach. As the name suggests, it is a rule that matches the classification the user chose at the creation of the service ticket to one that is done by the monitoring system at the resource ticket creation. This has a very simple form: *if* ***servCategory*** *then* ***resCategory***. For one service category more similarity rules may exist. We assume each ticketing system has his own system of ticket classification.

Example: A user creates a ticket for the shopping cart and search catalog e-commerce service (see Figure 1). The service category represented in this case as a "classification path" could be *SW/webAppl/searchCatalog/cannotSaveSearch*. The principal part of this classification path is *SW/webAppl/searchCatalog* which denotes there is a ticket for a web application search catalog. The possible rules in the service definition are:
if *SW/webAppl/searchCatalog* **then** *HW/Server/WAS* or
if *SW/webAppl/searchCatalog HW/Server/WAS/indexdb* or
if *SW/webAppl/searchCatalog HW/Storage/Database/CatalogDB* or
if *SW/webAppl/searchCatalog HW/Storage/Database/TransactionDB*

Dependency Tree: The *dependency tree* is a representation of a network with all its components and the relationships between them. These components are the *related* CIs.

Example: Figure 2 in Section III shows a part of the dependency tree for the realization of the service *catalog shopping*. In e.g. *httpServer1a* is depending on *dispacher2a*. The *dispacher2a* depends on *WASServer3a*, *ComputerSystem3a* and *OSNLN3a*. The *WASServer3a* depends on data base server *DBServer5a* and *DBServer5b*. *DBServer5a* depends also on *WASServer4a* which depends on the *indexdb*. The other data bases which are controlled from the database servers *DBServer5a*: *catdb1*, *catdb2* are dependents of the *shopping.ear*

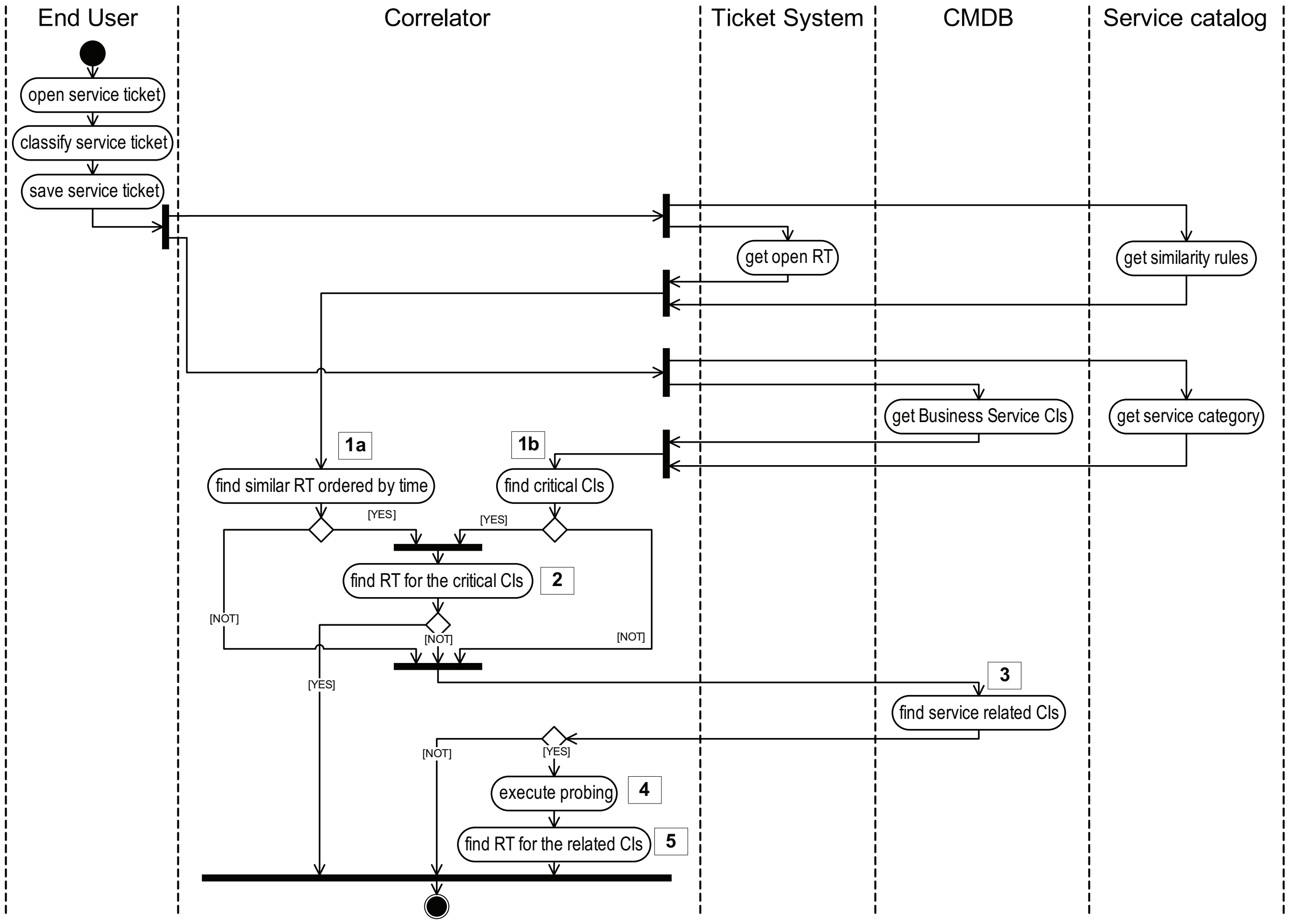

Fig. 4. Process Workflow for Trouble Ticket Correlation

contained in *WASServer3a*.

Business Service CI: The *Business Service CI* is an instantiation of the service definition. It is of great importance in our approach that Business Service CI contains information on the CI instances that are *critical* for support of the service instance for a specific customer. Critical CIs are also included in the dependency tree, as a subset of related CIs.

Example: The *Business Service CI*, as an instance of the catalog shopping service for the customer **A2816AB**. Critical CIs for this instance are the *dispacher2a*, *WASServer3a*, *WASServer3a* and *DBServer5a*.

Constraint Adaptive Probing (CAP): is a technique for finding the most effective way of probing CIs in an often large dependency tree within a given duration of time and without overloading the network.

B. Optimized Correlation Model

In this section, we propose a novel approach for service to resource ticket correlation, building on the notions introduced in the previous section. Our goal is to reduce the computational overhead and increase the accuracy of determining correlated service and resource tickets. Namely, given a service ticket and a pool of resource tickets, the proposed approach is based on three components:

- category-based correlation, which filters the resource tickets based on the *similarity rules*.
- *critical-CI*-based correlation, which filters the resource tickets based on their reference to the CIs that are critical for the failed service. By looking only at the critical CIs, we aim to minimize the overhead of dependency tree search, also called topological comparison.
- temporal correlation, which uses CAP to ensure the best way to probe the CIs in the dependency tree in order to trigger a creation of resource tickets by monitoring system.

The activity diagram in Figure 4 describes the steps of the correlation process, starting from the creation of the service ticket by the end-user.

The column on the left side of the diagram shows the activities of the **End User**. On the right side we have the provider domain. Activities of the provider are shown in the right part of the diagram, separated into four columns: **The Correlator** realizes correlation activities. **The Ticket System** includes open tickets which we use in the correlation. The

CMDB includes the dependency tree (topology with related CIs) and the Business Service CIs (with their critical CIs) as defined before. Last but not least the **Service Catalog** represents the interface to the customer. In the Service Catalog we find the service definition.

The activity starts with the End User opening and classifying a new incident ticket, through selection of classification path. E.g. user **B** of customer **A2816AB** opens a ticket regarding the service *catalog shopping* with the service category *SW/webAppl/searchCatalog/cannotSubmitRequest*. From here on the activity is driven from the correlator domain. Before real correlation can occur additional data is retrieved from the above named domains. Existing open resource tickets (RT) are retrieved from the ITS. Through the service category (classification path) in the service ticket, all relevant similarity rules are pulled from the service definition. Concurrently the service category and the customer identification (from the service ticket) are retrieved from the Business Service CIs. Using the data above, the two activities 1a and 1b are processed in parallel

1a Finding similar tickets from the open resource tickets (of the Ticket System) ordered by time. First a comparison between different service categories is done. This helps to reduce the number of searched tickets to only those tickets which somehow refer to this service. This information is stored as values of the attributes *resCategory* and *servCategory* of the ticket and can be matched with the similarity rules in the service definition. The depth of the classification path in our example is 4 but typically the deeper this is the more refined the search is, and the higher the precision in rule matching. This classification path is the first part of the similarity rule. The second part will be as well a classification path but for a resource ticket, in most cases indicating on a faulty resource.

1b Finding critical CIs as, defined in the Business Service CIs combined with the service category from the service definition. This information will be retrieved through the value of the attribute *customerID* contained in the service ticket and the service category of the service definition. If both these activities succeed, activity 2. follows otherwise activity 3.

2 Critical CIs found in **1b** are used to search the similar RT from **1a** ordered by time for corresponding RTs. If we find matches then the correlation is complete. This is the best use case scenario of the proposed search.

3 If there were no RTs found for the critical CIs the input is needed from the dependency tree (topology) for the Business Service CIs. If no related service CIs for this service have been identified, than the correlation concludes as there are no RTs that can be correlated. Otherwise follows the next step. The related CIs (in the dependency tree) by definition include the critical CIs too, so we cannot miss any candidates in this search (CIs that were not found in **1b** are surely found in **3**).

4 In our approach we anticipate that in some cases no tickets for the last time slot are available after activity 3 was done, because of the monitoring configuration. In this situation we propose to use CAP on the resources in the dependency tree. The probing of the faulty resources generates resource tickets. The role of CAP is to find the most effective way of probing for a given set of CIs within a dependency tree that could be completed during a given time duration with the restriction on number of probes that could be executed in parallel. The feasibility of this solution and the algorithm are further described in Section V

5 Again the Ticket System is searched on RTs for the related CIs. The correlations ends after this step either with a list of correlated RTs or with the result that there are no correlated RTs.

```
procedure IncidentTicketCorrelation
    $initializeAllList$    ▷ all lists needed are initialized
    $newServiceTicket = getNewServiceTicket()$
    $listOfOpenRT = getListOfRT()$
    $listOfSimilRules = getSimilRules()$
    $servCategory = getservCategory()$
    for each $SimilRule$ in $listOfSimilRules$ do
        $ResIdentTree = getResourceIdentTree()$
        $listOfResIdentTree.add(ResIdentTree)$
    end for
    for each $RT$ in $listOfOpenRT$ do
        $RTcategory = getResIdnetifier()$
        for each $resCategory$ in $listOfResIdent$ do
            if $RTcategory = resCategory$ then
                $listOfSimilarRT.add(RT)$
            end if
        end for
    end for
    $BusServCI = getBusServCI(customerID)$
    $listOfCriticalCIs = getCriticalCIs\ (BusServCI, servCategory)$
    if notempty ($listOfSimilarRT$) and notempty ($listOfCriticalCIs$) then
        $listofRTforCritCIs = findRTforCritCIs$
        if notempty ($listofRTforCritCIs$) then
            $listOfCorrelRT = listofRTforCritCIs$
        end if
    end if
    $listOfRelatedCIs = findServiceRelCIs$
    if notempty ($listOfRelatedCIs$) then
        $listOfRTforRelCis = findRTForRelCIs$
        $listOfCorrelRT = listOfRTforRelCis$
    else
        $doCAP(listOfRelatedCIs)$
    end if
return $listOfCorrelRT$
end procedure
```

Fig. 5. Incident Ticket Correlation Algorithm

Figure 5 describes the correlation algorithm in pseudocode with more implementation details. The procedure INCIDENTTICKETCORRELATION implementing the activity diagram in Figure 4 starts with initializing all the list which are used in the algorithm.

At the beginning the list of open RT and the similarity rules will be retrieved (lines 4, 5) from the ticket system respectively from service definition. The service category will be obtained from the service ticket (line 6). The specific resource category of the similarity rules for the identified service will be added to the list of resources (lines 7-10) and the list of similar resource tickets is filled (lines 11-18). These steps corresponds to **1a** in the activity diagram. **1b** is here realized in line 20 and **2** in lines 21-26. Related CIs are found on line 27 (corresponding to **3** in Figure 4). Finally the list of resource tickets for related CIs is filled and the list of correlated CIs will be returned. Constraint adaptive probing is realized on line 32.

V. CONCEPT EVALUATION

A. Constraint Adaptive Probing Formalization

This section is devoted to formal exposition of the Constraint Adaptive Probing Problem. We also provide an algorithm that gives us a feasible solution of the problem.

In the CAP we are looking for a sequence of sets of probes, which guarantees that number of left CIs to be probed manually regarding the incident is minimal.

Suppose that one of the CIs in the dependency tree of the service failed, our goal is to find failed CIs and through probing to trigger the generation of RT. If the dependency tree is large and all probes are executed sequentially the probing exercise could take a long time. Since the ticket system has to be searched after the probing is completed we need to restrict the duration for the probing. This means that the number of sequential tests has to be limited.

Another restriction that we introduce is the limitation on the number of probes that run in parallel. Multiple probes issued simultaneously could negatively affect the network.

Problem (Constrained Adaptive Probing) under the following constraints

a) a number of the sequential tests should not exceed a predefined number L.
b) a number of parallel tests should not exceed a predefined number P.

find a sequence of sets of probes such that the number of CIs that is left to be considered is minimal.

This problem is NP hard, since it contains problem of active diagnosis as a subproblem (see [9]). As an approximate solution to the CAP we give an algorithm based on the greedy approach.

Definitions: Let us assume that a service's dependency tree contains *n* CIs or nodes $N = \{N_1, \ldots, N_n\}$ each of which could be in one of two states OK or $FAILED$. As in example above the CIs in the dependency tree could represent a physical component (server, network, hub etc.) or software components (web application etc). The state of the business service that relies on this dependency tree is denoted by a binary vector $X = \{X_1, \ldots, X_n\}$, where X_i is one of the states OK or $FAILED$ of a CI N_i.

A probe or test T is a method for finding information about service's CIs. We denote by $N(T) = \left\{N_{T_1}, \ldots, N_{T_j}\right\}_{j=1} \subset N$ a set of Cis which are tested by probe T. Probe T fails if one of the CIs is in state $FAILED$ and succeeds if all CIs in N(T) are OK. For simplicity sake we suppose that each test takes time 1.

Dependency between different CIs are expressed in the form of dependency Matrix $D(m \times n)$, where $D_{ij} = 1$ if N_j depends on N_i or $N_j \to N_i$. For the set $\widetilde{N} \subset N$ denote $BD(\widetilde{N}) = \{N_i \in N | \exists N \in \widetilde{N}$ such that $N \to N_i\}$.

Dependency matrix and probes are related as follows: if probe T is OK than $BD(N(T))$ is OK.

In our exposition we follow Khinchin's [13] approach to the information theory. Let $\mathcal{P}$ denote a partition to the set of CIs N and $\mathcal{P}(\{A_1, ..., A_m\})$ denote partition generated by the sets $\{A_1, ..., A_m\}$ from N.

```
Greedy Constraint Adaptive Probing Algorithm
Input:
A set of probes T, dependency Matrix $D$, resource constraint $P$,
time constraint $L$, prior partition$\mathcal{P}$ of the service nodes N
Output:
A sequence $S = S_1, S_2, ..., S_k$ of subsets
of probes $S_i \subset T|$ with $|S_i| \leq P$ & $k \leq L$
Initialize:
$S \leftarrow \emptyset$
do
1. Select p most informative probes $T_{i_j}, \ldots, T_{i_P}$ s.t. $T_{i_j}, \ldots, T_{i_P}$
are $argmax H(\mathcal{P}(S \cup \mathcal{P}(T_{i_j}, \ldots, T_{i_P})|S))$
2. if probe $l_i$ returns $FAILED$ then
$N \leftarrow N \cap BD(N(T_{l_1}))\ and\ \mathcal{P} \leftarrow \mathcal{P}|_{BD(N(T_{l_1}))}$
3. $S \leftarrow S \cup \left\{T_{i_j}, \ldots, T_{i_p}\right\}$
while
$i \leq L$ & $\exists T \in T \setminus \bigcup_l S_l$ s.t.
$H(\mathcal{P}(\vee_{l=1}^{i}\{BD(N(T_{l_1})), ..., BD(N(T_{l_p}))\}))|$
$\mathcal{P}(\vee_{l=1}^{i-1}\{BD(N(T_{l_1})), ..., BD(N(T_{l_p}))\}) > 0$
return $S$
```

Fig. 6. Greedy Constraint Adaptive Probing Algorithm

Information of the partition is defined as

$$I(\mathcal{P}) = \sum_A \chi_A \log p(A) \qquad (1)$$

where χ_A is a characteristic function of A and sum is taken over all atoms of the partition $\mathcal{P}$. Relative information of partition $\mathcal{P}_1$ to partition $\mathcal{P}_2$ is defined as

$$I(\mathcal{P}_1|\mathcal{P}_2) = -\sum_A \chi_A \log p(A|P_2) \qquad (2)$$

where $p(A|\mathcal{P}_2)$ is a conditional probability of A relative to $\mathcal{P}_2$. We also consider conditional entropy

$$H(\mathcal{P}_1|\mathcal{P}_2) = \int I(\mathcal{P}_1|\mathcal{P}_2)\mathrm{d}p \qquad (3)$$

where $\mathrm{d}p$ is a normalized counting measure on N.

The Greedy Constraint Adaptive Probing Algorithm in Figure 6 is a polynomial approximation of solution for the Constrained Active Probing Problem.

B. Evaluation through experimentation

In order to illustrate the advantages of our optimized model we ran an evaluation through experimentation. The optimized model allows us to reduce the user's decision choice from a tree of depth 11 to a tree of a residual depth (11 - hightOfBar). Figure 7 shows the depth of the dependency tree on the y-axe. We assume that resource dependency is a binary tree of depth

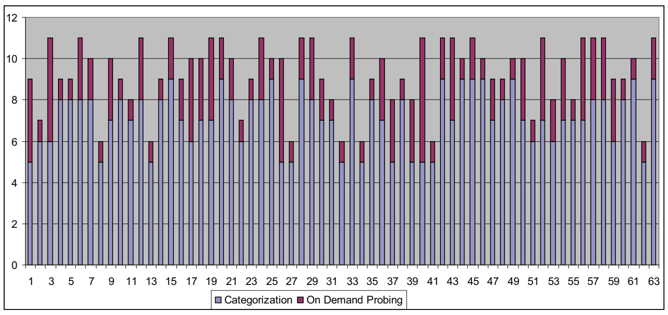

Fig. 7. Results of the simulation

11, which gives a complete system size of 2048 resources. We assume that the classification path varies from 6 to 9 uniformly distributed. We assume that CAP acts in addition of the categorization. We ran the simulation for 4096 times. For a better visualization only the first 64 runs are shown. The simulation illustrates benefits from category-based correlation alone - light dashed colored part of the columns and additional benefit is provided by CAP - dark colored part of the column (Figure 7).

The simulation shows that significant progress could be made towards a correlation of service ticket to resource tickets based on category-based correlation. Furthermore the constraint adaptive probing capability augments the category-based results with additional resolution that further reduces the need for human intervention.

VI. Conclusions and Future Work

In this paper we examine an innovative approach to ticket correlation that improves the accuracy and effectiveness of the incident / problem management process. In particular we provide a correlation capability that leverages insights drawn during both service definition and the description of the deployed infrastructure in configuration management systems. The approach exploits an optimization model based on step-wise correlation in which service categorization is augmented with service/resource similarity rules to facilitate selection of resources that demonstrate correlation between tickets. We further describe how this approach can be augmented by Greedy Constraint Adaptive Probing Algorithm to dynamically identify additional resource details needed for correlation when they are not directly available based on monitoring system limitations. We show through simulation that these combined approaches may provide significant improvements in the ability for analysts to accurately categorize and subsequently correlate a set of tickets.

As future work, we plan to extend this correlation model to handle tickets originated from different Incident Ticket Systems. This applies to heterogeneous service environments with multi-supplier hierarchies or multi-stage services provided by multiple equal partners Also, we plan to evaluate the impact of the suboptimal solution for the CAP Algorithm on the optimality of the ticket correlation results.

Acknowledgments

This paper is the result of the research during a summer internship in 2008 at the IBM T.J. Watson Research Center. The authors are in debt to Dr. David Loewenstern from IBM T. J. Watson Research Center for the insightful suggestions. The authors also want to thank MNM-Team directed by Prof. Dr. Heinz-Gerd Hegering, for helpful discussions and valuable comments.

References

[1] "IT Infrastructure Library, Office of Government Commerce (UK)," http://www.itil.co.uk.

[2] OGC (Office of Government Commerce), Ed., *Service Operation*, ser. IT Infrastructure Library v3 (ITIL v3). Norwich, UK: The Stationary Office, 2007.

[3] G. Dreo, "A Framework for Supporting Fault Diagnosis in Integrated Network and Systems Management: Methodologies for the Correlation of Trouble Tickets and Access to Problem–Solving Expertise," Ph.D. dissertation, Ludwig–Maximilians–Universität München, Jul. 1995.

[4] A. Hanemann, "Automated IT Service Fault Management Based on Event Correlation Techniques," PhD thesis, University of Munich, Department of Computer Science, Munich, Germany, May 2007.

[5] A. Hanemann and P.Marcu, "Algorithm Design and Application of ServiceOriented Event Correlation," in *Proceedings of the 3rd IFIP/IEEE International Workshop on BusinessDriven IT Management (BDIM 2008)*, Salvador Bahia, Brazil, April 2008.

[6] K. Chang, H. Carlisle, J.Cross, and P. Raman, "A self-improvement helpdesk service system using case-based reasoning techniques," in *Computers in Industry*, New York, March 1996, pp. 113–125.

[7] E. Liddy, S. Rowe, and S. Symonenko, "Illuminating Trouble Tickets with Sublanguage Theory," in *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, New York, June 2006, pp. 165–172.

[8] R. Gupta, K. Prasad, and M. Mohania, "Automating ITSM Incident Management Process," in *Proceedings of the 5th IEEE International Conference on Autonomic Computing*, Chicago, June 2008, pp. 141–150.

[9] I. Rish, M. Brodie, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, and K. Hernandez, "Adaptive Diagnosis in Distributed Systemss," *IEEE Transactions on Neural Networks (special issue on Adaptive Learning Systems in Communication Networks)*, vol. 16, no. 5, pp. 1088–1109, 2005.

[10] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik, "Real-time Problem Determination in Distributed Systems Using Active Probing," in *Proceedings of the 9th IFIP/IEEE International Network Management and Operations Symposium (NOMS 2004)*, Seoul, Korea, April 2004, pp. 133–146.

[11] B. Gruschke, "Integrated Event Management: Event Correlation Using Dependency Graphs," in *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, Delaware, USA, October 1998, pp. 130–141.

[12] J. Stanley, R. Mills, R. Raines, and R. Baldwin, "Correlating network services with operational mission impact," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, Chicago, October 2005, pp. 162– 168.

[13] A. L. Khinchin, *Mathematical foundations of information theory*. New York: Dover, 1956.