Reiki: Serviceability Architecture and Approach for Reduction and Management of Product Service Incidents

Chris Connelly, Brian Cox, Tim Forell, Rui Liu, Dejan Milojicic, Alan Nemeth, Peter Piet, Suhas Shivanna, Wei-Hong Wang

HP Laboratories
HPL-2009-88

Keyword(s):

serviceability, support, incidents, failure reduction

Abstract:

There is a significant number of IT failures per year because parts fail, products are used in ways they were not designed for, and humans make errors in using products. These failures result in incidents that product vendors service as a part of the warranty or contracts. Incidents incur significant costs for servicing them, including call centers, parts, and field engineers. Some of the major problems include lack of coherent incident information, leading to inaccurate service diagnosis and inability to forecast failures. At the same time, technology has evolved. Hardware is generally more reliable, failures are moving from hardware to firmware, software, and applications. The scale effect limits human operator engagement, prevents centralized approaches, and expands automation. Traditional ways of handling incidents are not appropriate any more. In this paper we present a set of tools and approaches that enable unified serviceability with selfhealing, automated learning, and an analysis engine. Unified serviceability with self-healing results in clean incident data and it reduces criticality of incidents into deferred maintenance. Automated learning produces empirically proven actionable knowledge enabling cost reduction of automated incident resolution. Using clean data and actionable knowledge, the analysis engine helps predict failures and determine trends, resulting in preventive maintenance. Collectively, preventive and deferred maintenance and automated incident service significantly reduce the costs. This way we have aligned incidents cost with the technology trends.

External Posting Date: April 21, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: April 21, 2009 [Fulltext]

To be published in ICWS 2009, July 2009,

Reiki: Serviceability Architecture and Approach for Reduction and Management of Product Service Incidents

Chris Connelly,[1] Brian Cox,[1] Tim Forell,[4] Rui Liu, [2] Dejan Milojicic,[2]
Alan Nemeth,[3] Peter Piet,[4] Suhas Shivanna,[4] and Wei-Hong Wang[2]
[1]*HP-IT,* [2]*HP Labs,* [3]*HP-EDS,* [4]*HP Enterprise Storage and Servers*
[firstname.lastname]@hp.com

Abstract

There is a significant number of IT failures per year because parts fail, products are used in ways they were not designed for, and humans make errors in using products. These failures result in incidents that product vendors service as a part of the warranty or contracts. Incidents incur significant costs for servicing them, including call centers, parts, and field engineers. Some of the major problems include lack of coherent incident information, leading to inaccurate service diagnosis and inability to forecast failures. At the same time, technology has evolved. Hardware is generally more reliable, failures are moving from hardware to firmware, software, and applications. The scale effect limits human operator engagement, prevents centralized approaches, and expands automation. Traditional ways of handling incidents are not appropriate any more.

In this paper we present a set of tools and approaches that enable unified serviceability with self-healing, automated learning, and an analysis engine. Unified serviceability with self-healing results in clean incident data and it reduces criticality of incidents into deferred maintenance. Automated learning produces empirically proven actionable knowledge enabling cost reduction of automated incident resolution. Using clean data and actionable knowledge, the analysis engine helps predict failures and determine trends, resulting in preventive maintenance. Collectively, preventive and deferred maintenance and automated incident service significantly reduce the costs. This way we have aligned incidents cost with the technology trends.

1 Introduction

IT products fail: servers and storage in data centers, laptops, printers. This is inevitable due to the failure rate associated with materials and parts, mis-configuration, software bugs, incompatibilities, etc. In addition, products are used in ways that they were not designed for. A substantial amount of time, money, and effort is invested in design for serviceability, but incidents still happen, and the cost to alleviate them is substantial, in the range of billions of dollars.

Traditionally, the incidents are handled by call centers, which dispatch customer engineers and parts to service the incident. To reduce the costs, various optimizations are introduced: a) products are designed with increased redundancy or resilience to enable self-healing; b) customers are educated to enable self-mitigation; and c) service delivery is automated to reduce the human engagement. (See Figure 1). The "lines of defense" for incidents incrementally grow from products, through customers, to support organization.

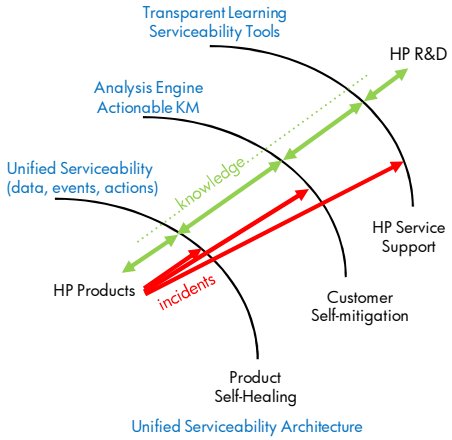

Figure 1. Serviceability Landscape

Incident lifecycle consists of the following four phases: detect, diagnose, mitigate, and restore (see Figure 2). Historically, most automation has taken place in the detect phase, with some automation to mitigate and restore. Diagnose is hardest to automate. Incident service can be reactive, upon incident; preventive, predating incident; and deferred, at later time. Each approach can be automated or not.

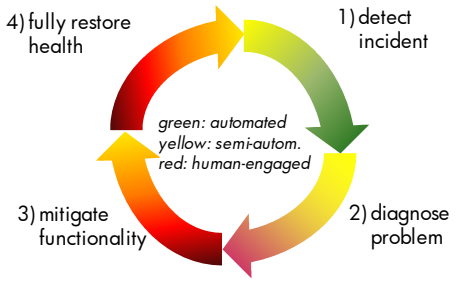

Figure 2. Service Incident Lifecycle.

The trends are changing the landscape of service delivery. Hardware is becoming more reliable and incidents are moving up in the stack: from hardware to

firmware, to software and applications. New data centers are being built with higher levels of redundancy, such as containers, designed for lower criticality of maintenance. New applications are designed with built in resilience, such as Search or Web servers, which can sustain failures of the underlying hardware. Customers are expecting to self-support products much more than in the past.

The hypothesis of our paper is presented in Figure 3, as a hypothetical prediction of the breakdown of preventive v. reactive v. deferred as well as automated v. manual incident service delivery. Our first hypothesis is that reducing human-involved incidents through automation saves costs throughout the service lifecycle. Our second hypothesis is that it is possible to **shift** the incidents into lower cost types:

1. Reduce the proportion of reactive incidents (i.e. incidents that require immediate actions by humans) - which are the most expensive - to incidents deferred automatically by the system which is less expensive
2. Increase the overall proportion of incidents avoided through preventive actions (least expensive)

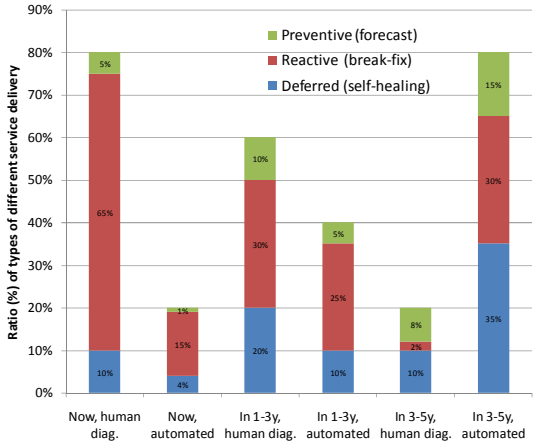

Figure 3. Hypothesis: Hypothetical Breakdown of Preventive/Reactive/Deferred & Automated/Manual

The rest of the paper is organized as follows. Section 2 provides the background of modeling service delivery cost. Section 3 compares current and future serviceability. In Section 4 we present serviceability design and in Section 5 implementation. Section 6 presents specific examples of benefits of our serviceability framework. In Section 7, we provide data analysis. In Section 8 we compare our work to related research. We conclude in Section 9.

2 Minimization of Incident Service Cost

Reiki aims to minimize the total annualized costs involved, by studying costs incurred by incident service delivery, adding spares, and Service Level Agreement (SLA) non-compliance. We follow the decomposition of costs as below:

Total Annualized Cost =
(ProbOfSLNon-Compliance)x(SLAPenaltyPerHour)x8760HoursPerYr
+(PredictNumOfSvcEventsPerYr)x(AvgCostPerSvcEventPerSvcContractType)
+AverageCostPerYrToDeploySpares (if any)

We used continuous time Markov chains to evaluate the probability of not meeting the conditions specified in a Service Level Agreement (SLA). The result is then combined with the SLA non-compliance penalty cost resulting in an annualized SLA non-compliance penalty cost. Next, an annualized service cost is developed by multiplying the number of predicted service events per year times the average service event cost (corresponding to each of the types of service contracts offered). Additionally, the annualized cost of adding spares was calculated. The total annualized cost was then calculated by adding up the three types of costs indicated above. Total annualized cost is then graphically displayed so that various tradeoffs between service contract type (some of which are reactive and some of which are deferred) and sparing level can be studied as a function of SLA non-compliance penalty severity.

As an example, we developed an in depth analysis for a medium sized deployment of server blades where the level of sparing, type of service contract, and service level agreement (SLA) penalty costs are studied. The total annualized cost of service, sparing, and SLA penalties is the objective function that is sought to be minimized. First, a Markov chain model is used to predict the steady state probability of not meeting the conditions specified in the SLA (a minimum number of server blades perfectly healthy) based on predicted server blade failure rates, predicted server blade recovery rates (which depend upon type of service contract), and level of sparing (0,1, or 2 two spare server blades). Next, the total annualized cost as described above is calculated for each combination of service contract type and sparing level. A typical result can be summarized graphically as shown in Figure 4.

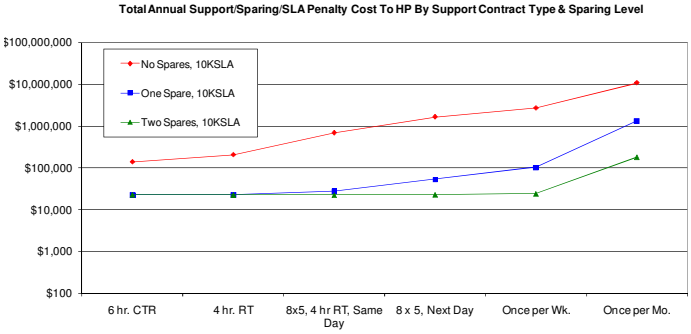

Figure 4. Service Delivery v. Degree of Redundancy

The graphical display then allows for trade-offs to be addressed directly. For example, for the server blade farm under consideration, and for an assumed $10,000 per

hour penalty function for SLA non-compliance, the addition of two spare server blades allows for deferred maintenance type service contracts (next day, once per week), that in turn allows for considerable service cost savings, without giving up customer satisfaction as measured by the degree of SLA penalty costs incurred.

We observe that the incremental cost of introducing spares may be justified since SLA penalty costs may be avoided and less expensive deferred maintenance type service contracts may be utilized, (depending on the severity of the SLA non-conformance penalty costs). Since severity of SLA penalty non-conformance will be dependent on the level of mission criticality of the market segment in question, other scenarios were developed (for example, using a $100,000 per hour penalty function for extremely mission critical applications, or alternatively, using a $1,000 per hour penalty function for much less critical applications.) Interestingly, inclusion of adequate sparing still enables deferred maintenance service contracts (e.g. next day or once per week) to be valid low cost choices.

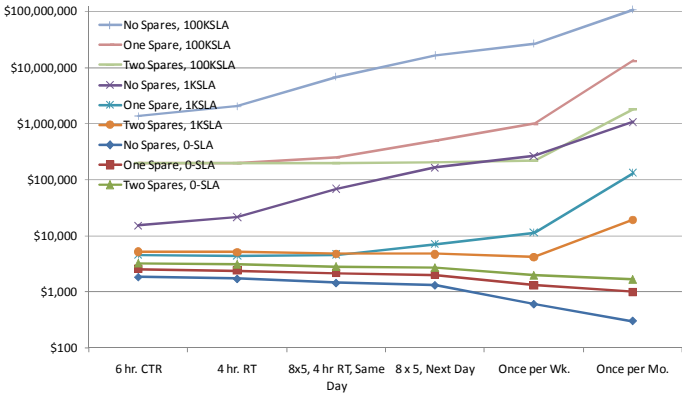

Figure 5. Service Delivery v. Market Segment

3 Use Cases

Today…. When a blade fails in a **datacenter**, an event is created, which gets recorded in an event database and after filtering in the incident database. This further triggers a human operator in a call center to dispatch a part order and a customer engineer to explore the incident and eventually replace the blade if its failure is confirmed. There are too many touches by humans in the process, primarily raising the cost, but also introducing opportunity for misdiagnosis.

When a failure happens to your **laptop** and it stops working, if you are an average user you do not have many options. Sophisticated users can play with the built-in diagnostics and beyond that the only option is to take it to a local shop and hope for the best, at least to retrieve the data out of it (the delta since you made the last backup). If it is more than year and a half old, any intervention will likely cost more than the value of the laptop.

Future vision…. datacenters will be designed as containers, with built in power and cooling. They will be built to last a certified period of time, and replaced with equivalent units. Certain components within the container will continue to fail, but sufficient levels of redundancy will enable the container to provide the agreed upon SLA (more than 80% of products working). While this design was motivated by reducing management costs (installation, management), it has similar implications on servicing incidents. All service is done remotely, and the health of the datacenter is observed for anomalies from the predicted failure rates. If the health trend is as expected, intervention can be performed during regular maintenance, (weekly, monthly), at which time failed parts are replaced and preventive maintenance carried out. Most interventions migrate from critical (reactive) to scheduled (deferred, preventive). Most types of incidents are well known, the few new types are analyzed by operators and entered into the knowledge base, where they are compared to the inflow of new incidents. Both detecting and diagnosing incidents is carried out automatically.

As soon as you have bought your new **laptop**, an icon in the lower right corner has flashed the health of your system and the maintenance schedule. Similarly to the remaining battery time, the lifetime of the laptop makes reminders of how healthy your laptop is, what kind of maintenance if any you want to perform. If any failure is imminent, the laptop has already informed its producer and the part is on the way. If the laptop is completely dead, you can connect it to another using a USB cable and perform some troubleshooting. For complex failures, a certified operator may have to be engaged, but all the knowledge required to service incident is available off of the portal.

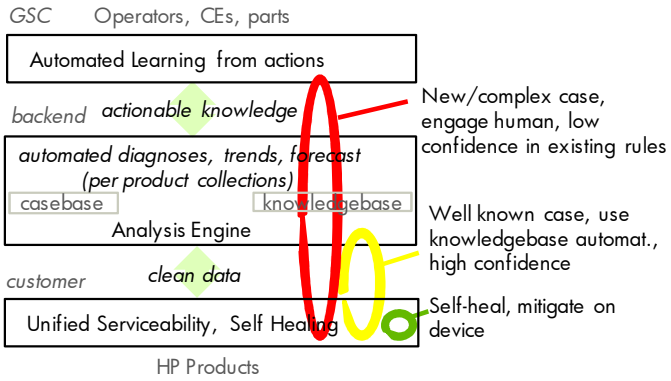

Figure 6. Design

4 Design

The design of our serviceability model is presented in Figure 6. It consists of *unified serviceability* with *self-healing* built into products; *automated learning* from call center operator actions; and an *analysis engine* acting upon data from products and knowledge from automated learning. These three components enable serviceability closed loops: a) on products through self-healing; b) in the back-end data centers through automated rules; and c) in call centers with the help of humans. The first two closed loops can be automated entirely, while the third assumes human engagement. Human engagement is a determining factor for the cost of service delivery.

Unified Serviceability assumes the same set of product data, events, and serviceability interfaces across all products. Product data includes the date the product was manufactured, when it was sold, when it was serviced and how, etc. Similar events consist of the same or similar types of alerts and other information coming from products and into the back end data center. Finally serviceability interfaces enable remotely querying product data, updating firmware, reconfiguring and other serviceability action which could be performed remotely (preferably) or locally. Unified serviceability dramatically reduces service delivery cost because of unification of tools, educating operators, increased automation, etc.

We define **self healing** as an ability of a product (hardware, firmware, OS, middleware, application) to detect, diagnose, and automatically mitigate a product fault. Self healing usually involves automatic repair of localized faults (e.g. ECC in DIMMs) or fail-over to redundant components. The advantages of self healing are in reducing the costs and improving customer experience. Server and storage products have many self healing features built-in as part of the hardware, firmware and OS components. Some examples of these features in HP products are:

1. HP Advanced Memory Partition technology to detect and correct 2 DRAM failures.
2. Dynamic Processor Resiliency to recover from high rate of processor errors with the help of OS
3. Dynamic Page De-allocation to disable the use of pages mapped to failed DIMMs
4. OS initiated recovery for certain types of machine checks caused by uncorrected CPU errors

Automated Learning enables transparently capturing actionable knowledge from the operators in call centers. The biggest challenge is to accomplish automated learning transparently, without slowing down operators, such as requiring additional information, traversing troubleshooting trees or forms to classify incidents or recommend remedies. Significant simplifications for this activity rely on following assumptions:

- Most simple hardware failures will have already been accurately diagnosed by self-healing components.
- A lot of context information will be provided from the product, failure history, surrounding infrastructure, etc.
- Complex failures will be matched against past and offered to operators as candidate diagnosis.
- Building confidence in diagnosis will be accomplished over time, with the fallback to human expertise.

The outcome of the automated learning are: runbook automation-like sets of actionable rules, which can be executed to automatically mitigate incidents and restore health of systems; and accuracy estimates—the confidence level in the diagnosis. Confidence level is built over time. Once it surpasses the predefined confidence threshold, the incident is moved from the new to known category (see Figure 7).

The **Analysis Engine** relies **o**n clean data from products and actionable knowledge from call center operators to make automated diagnosis, evaluate trends and create forecasts. Each of these are valuable only if fairly accurate. The accuracy of complex rules (for a product collection) as well as timeliness of data play a major role in analysis engine effectiveness. Accuracy is verified in real deployments and compared to the predicted levels. The artifacts of the analysis engine comprise knowledge reports for R&D, customer, channel, and partners.

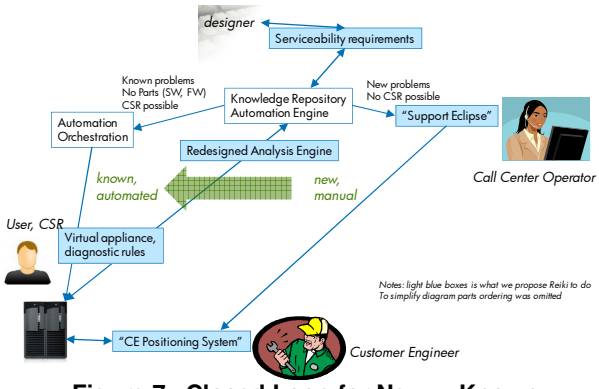

Figure 7. Closed Loop for New v. Known

The classification of service incidents as *new* or *known* originates in Knowledge-Centered Support (KCS) [5], a common methodology for capturing, maintaining and reusing knowledge in support organizations. An incident is said to be *known* when the knowledge base contains a description of the problem that can be used to identify it and an associated solution. An incident is classified as *new* when there is no corresponding problem-solution pair. The ratio of new to known incidents is often used to measure the efficiency of a support organization [6]. Additionally, problem-solution

pairs can be exposed outside of the support organization as a way of enabling users or partners to resolve problems by themselves.

KCS and "New versus Known" were designed for manual support processes (i.e. knowledge is captured, maintained, and located, and reused by call agents, product users and designers); however, the concepts are also applicable to automated serviceability. The consistent data and events from a product and its environment obtained from unified serviceability products allow problems to be characterized and diagnosed by the analysis engine. Solutions manually linked to a problem allow support personnel to create actionable rules, and in some cases, the automated learning feature generates actionable rules without human input (e.g. automated part dispatch for break-fix incidents). Feedback from solution reuse and manual review facilitates the identification of high-confidence solutions and creation of actionable rules. Rules may execute actions in the support provider's environment (e.g. automatically shipping a part to a customer) or in the customer environment (e.g. launching a recovery action). Finally, pattern matching rules for identifying problem diagnoses can be incorporated into knowledge search tools to improve the effectiveness of existing KCS processes.

5 Implementation

There are currently several independent service incident management ecosystems in use – each focused on a particular business segment. Within each domain there are disparate levels of product serviceability, multiple event sources (automated, manual), a plethora of (point) analysis mechanisms and a fragmented approach to knowledge management (See Figure 8).

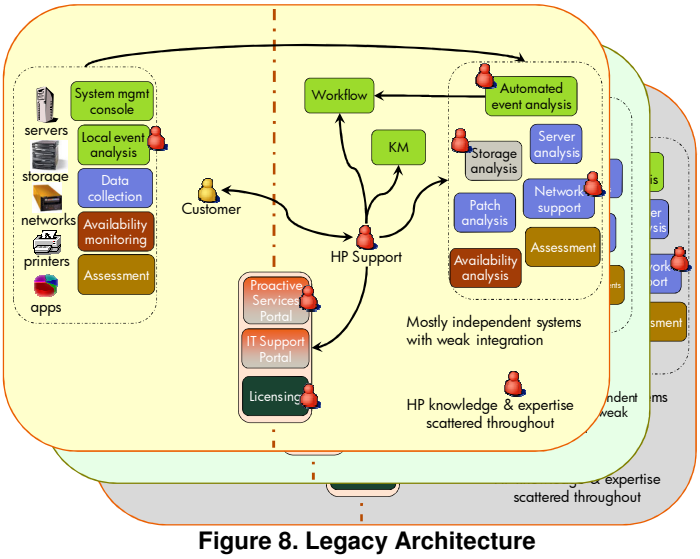

Figure 8. Legacy Architecture

The three main areas of development—Automated Learning, Analysis Engine and Serviceability Analysis—will integrate with the existing incident management fabric and utilize relevant data to help automate and optimize HP's management of incidents (see Figure 9). The Analysis Engine will augment existing event analysis mechanisms for codifying and automating the handling of events while also forecasting potential issues and initiating appropriate action. Automated Learning will strive to leverage the support engineer's expertise and actions in conjunction with active analysis of the available data (structured and unstructured) to indicate meaningful relationships/links and helpful suggestions (root causes, problem solving steps, associated issues, hints, etc.). Unified Serviceability will provide the product development teams a means of learning about existing deployments and identifying serviceability improvements. This will help drive enhanced self-healing capabilities on Services Objects and better event telemetry data feeding into the incident management system.

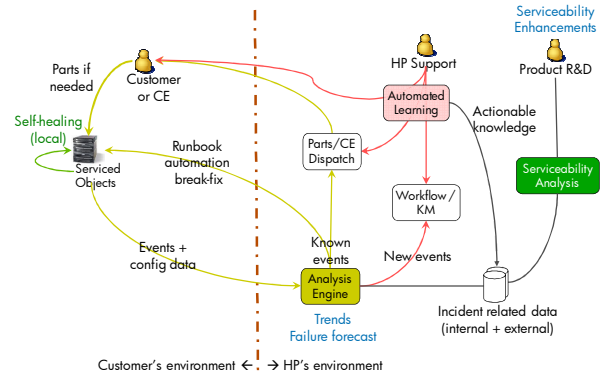

Figure 9. Reiki Architecture

When the various programs were initially conceived and developed proprietary interfaces and protocols were the mainstay. Over the last number of years things evolved to a point where most of the main solutions utilize web technologies and standards. For instance, the remote support event data models have been redesigned around a CIM-compliant data model mapped into an XML document format (XSC schema); web-based solutions have been developed to provide assisted troubleshooting; all user interfaces utilize a range of web technologies with a move towards greater standardization; and most services (internal and external to HP) are now accessible via web-based technologies. That said, the picture is still evolving and there is much to do. Many of the systems and solutions were initially created before the rise of web standards and mindsets (e.g. pre-Web 2.0).

6 Specific Examples of Benefits

Beside general benefit of Serviceability approach we have presented, there are also specific scenarios where it can be of unique help. For example:

Unified Serviceability with self healing can *eliminate user-maintenance inflicted false alarms* (close

the loop between product-site and HP DC to prevent false incident generation). It can also *reduce no faults found in parts* by performing fail-back to failed part just prior to its replacement (after original failover as a part of self-healing). Finally, it can *assist in evaluating end-to-end support cost* with/-out self healing for different regions with support staff of varying ability and, accounting for spares (inventory/built-in), and delivery (4h v. deferred)

Automated Learning facilitates *automation of a new-known incident promotion*, e.g. identifying a frequent, well understood incident and showing how we will move them from new to known. It can also *help manage the lifecycle of rules* (introduction, deployment, adjustment, retirement), including thresholds, criteria for retirement, etc. Finally, it can *identify typical diagnosis errors* and extract learning ("inverted, not-to-do rules"), suggest methodology how to identify them in a general fashion.

Analysis Engine can help *identify epidemics*, such as a patch that caused a failure on multiple sites, or so called "flash crowd" per region. It can also *enable "RSS Feed"* from case and knowledge bases to R&D designer of a product. It can help *identify trends for warranty changes* and showcase how to specify, calculate and insert triggers to re-evaluate/adjust warranty. Finally, it can assist to *overcome incomplete or incongruent data*: search for missing, critical data elements; real time feedback loops to humans; flag and conflicts.

7 Analysis of Potential Reiki Improvements

This analysis will examine the ratio of human-involved versus automated incidents. There are two sources of data to examine: system-generated and human-generated. System-generated data is from our widely deployed remote support software, designed to see system events and transmit them to the support "back-end" for analysis. System-generated data is entered automatically by software event triggers and system calls. The data is stored in our backend system event database. Human generated data is created by human service agents, either locally or remotely by analyzing problem information and creating remedies. The human-generated data is entered into our call-tracking system. Today, analysis is primarily done by humans (remote support agents) with remedies sent back to the client systems in the form of hardware fixes, firmware and software patches, and configuration or tuning changes. Note that call-tracking data base also contains system-generated cases which are distinguished in Table 1.

I. Current data shows a preponderance of human-involved service incidents. The table below shows the most recent 6 months of call-tracking data for one of our storage products that has a highly redundant design and has been deployed for several years. We distinguish between automated, system generated events (labeled Sytem Events) and the human-entered events (labeled HUman Events). Regional differences are yet to be understood but seem most distinct for Region 3. One explanation is that Region 3 customers have less deployment of the remote support software and traditionally rely on human support more so than the other regions. We currently believe that the system generated events are overstated and have an initiative in HP to drive down duplicates and false events from the remote support software.

Region	Call Tracking DB Cases	'08-08	'08-09	'08-10	'08-11	'08-12	'09-01 (26)	Total
1	System Events	43%	41%	37%	40%	41%	42%	41%
	Human Events	57%	59%	63%	60%	59%	58%	59%
	Total	**39%**	**37%**	**34%**	**34%**	**36%**	**41%**	**37%**
2	System Events	54%	48%	40%	48%	48%	44%	47%
	Human Events	46%	52%	60%	52%	52%	56%	53%
	Total	**39%**	**39%**	**44%**	**41%**	**40%**	**37%**	**40%**
3	System Events	21%	17%	17%	17%	16%	16%	17%
	Human Events	79%	83%	83%	83%	84%	84%	83%
	Total	**22%**	**24%**	**21%**	**25%**	**24%**	**22%**	**23%**
Global	System Events	42%	38%	34%	38%	38%	37%	38%
	Human Events	58%	62%	66%	62%	62%	63%	62%

Table 1. Distribution of human-involved and automated incidents by time and region

II. Current data shows a highest weighting of reactive incidents, followed by deferred, followed by preventive. The bar chart below is from the most recent 6 months of data for the same storage product. The remote support software has a tool that has error analysis rules for each failure mode. There are over 700 failure modes of which approximately 200 were seen as events in this time period. All of the incidents described by this data are automatically identified by software agents on the system, analyzed for severity, classified and recommended for an action. The "deferred" category, in almost all cases, represents component errors that have resulted in a failover to a spare part. This is because this product is at near 100% redundancy in design. In each case, a service event will be required to restore full system health, i.e. repair the failed component. The "reactive" category represents failure modes that require further human diagnosis and actions to address the problem. This will include in most cases the dispatch of a service engineer and replacement of parts. The "preventive" category represents information and warnings about conditions that should be addressed to avoid a failure. This may include a dispatch of an engineer to proactively service the system through part replacement or adjustments to hardware. If the preventive action is tuning, configuration, patching or some non disrupting action this may be done remotely.

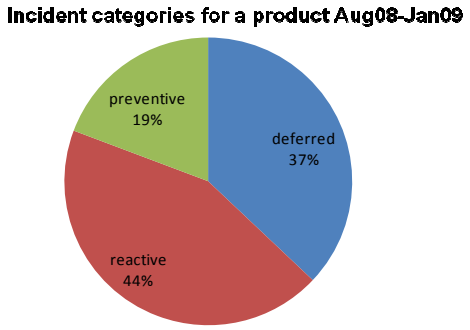

Figure 10. Preventive/Reactive/Deferred Incident Ratio

III. New vs Known. Figure 11 shows the ratio of automatically v. manually reported problems for two thousand servers running remote support software over the course of twelve months. Problems reported by the remote support software are considered to be "known" since detection is triggered by well defined conditions, and the solution is typically the replacement of a hardware component. The figure shows that a large portion of known failures can be detected automatically (although the data overstates automated problem detection due to false alarms). Manually reported problems may be "new" or "known," though a spot check indicates that the majority are known hardware failures, requests for configuration or setup assistance, and software problems not covered by the remote support tools. From this data we conclude that we can successfully detect certain classes of known problems, though opportunity exists to improve detection of failures and expand coverage to other classes of known problems (e.g. software failures).

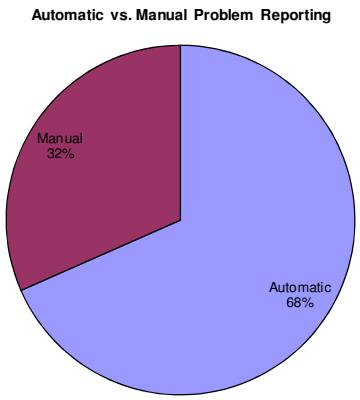

Figure 11. Automatic v manual problems reporting

We can derive the following conclusions

1. Deferred and reactive types together require human actions to address and represent 80.7 % of the total incidents. This becomes the potential for shifting human-involvement to automated. One could also add preventive to this group for automation.
2. Server data shows that a large percentage of problems can be detected and reported with no human involvement, but opportunities exist to extend coverage beyond simple hardware failures. Data from storage devices shows that even with highly redundant products, the potential is very large to convert human-involved incidents (62.31%) into fully automated incidents.
3. Analysis of the incident data by the system is focused on single, known events and does not consider combinations. This is a potential for a more sophisticated analysis engine.

8 Related Work

Self-healing at the hardware level is often achieved through redundancy and hot-swapping; recent work offers increased flexibility, e.g., for SoC, Akoglu et al. [1] proposed to localize and isolate the faulty area and replace the functionality through partial configuration of FPGA. Techniques of software dynamic updating and patching change parts or the entirety of a user-level program or a system program without interruption.

Operating systems provide profound mechanisms for fault detection and resolution in hardware and software. Sun Solaris 10 enables fine-grained response to failures from the lowest levels of hardware/software stack [2].

Web services have become increasingly self-healing. Carzaniga *et al.* [3] proposed to self-heal component-based applications by automatically identifying and executing workarounds. Baresi *et al.* [8] discussed self-healing service compositions based on defensive process design and run-time service monitoring. Mostefaoui *et al.* [9] further studied recovery actions as structured units, with self-healing actions implemented as software aspects, which are triggered and organized based on the type of fault and the context of use.

To address increasingly more complex systems and environments, Cook *et al.*[7] discuss the design for learning-based approaches to identifying the right fixes to errors, failures or faults in multitier services. The authors especially studied different data collection mechanisms, diagnosis- and signature-based synopsis building and querying, combinations of these options and input of human knowledge, and ranking of target synopses. Pranayama [10] is a self-healing solution which uses Bayesian belief network for root cause analysis and fault prediction in adaptive enterprise management.

For large-scale computing systems, IBM's autonomic computing blueprint [11] projects a novel architecture to enable systems to self-manage (-configure, -heal, -

optimize, and -protect) guided by human knowledge. System components must have monitor, analyze, plan, and execute functions, to adapt to changes and anomalies in accordance with business policies and objectives.

A self-healing framework for databases has been proposed in [12]. It aims to proactively detect potential problems in either a learning-based or programmed manner, maintain an adaptable repository of problem patterns or normal behavior patterns, mine knowledge from accumulated usage data, execution patterns and successful resolutions, while minimizing self-healing overhead against risk factors. Similarly, Oracle database has also put key development focus on self-healing capabilities [13], *i.e.*, proactive problem detection, limited damage and interruption, faster diagnosis, simplified resolution and repair, faster solution delivery.

Self-healing mechanisms are also discussed in contexts of automatically re-establishing trust in previously compromised virtual machines [4]. Many other mechanisms can help to improve serviceability. For example, the Boeing 787 aircraft is made 30% less expensive to maintain, because of the extensive use of composite components, integrated system architecture, and carefully scheduled maintenance plan [14].

The service industry has adopted knowledge-centered support (KCS) as a methodology for support teams to service customer problems by searching and updating a knowledgebase [5]. The support technician needs to document or update the very solution at the same time s/he is servicing the customer. Quality of the resulting knowledgebase is guarded by the competency of staff members. As proved by HP Non Stop Support [6], KCS brings consistent and high-quality solution sharing among service staff and end users, capture of knowledge, incremental improvement and reuse of group intelligence.

9 Conclusions

In this paper, we have presented Reiki, a serviceability architecture and approach for managing and reducing incidents from products. We motivated the need for such an approach and architecture by modeling costs and by providing two use cases. We then described the architecture and design, followed by some scenarios of use. We also provided some data analysis of the current levels of automation and costs. Reiki reduces the costs of serviceability by:

- Increased automation of service delivery
- Shifting service delivery to match level of automation
- Understanding costs and balancing product redundancy with the support organization's ability to service

In the future, we shall present more detailed quantified analysis of the results of our work. A comprehensive measurement of serviceability cost before and after deploying Reiki serviceability will be compared for individual products, product families, business, and for the overall services organization.

Acknowledgement

We would like to thank numerous people who helped us with data analysis, in particular: Al Haddix, Jay Harlan, Roy Carlson, Mike Kerr, Don Davis, Rod Naker, and Drew Walton.

References

[1] Ali Akoglu, et al, "FPGA Based Fault Detection, Isolation and Healing for Integrated Vehicle Health", AAAI Fall Symp. on Artificial Intelligence for Prognostics, Nov'07.

[2] Sun Microsystems Technical Report, "Predictive Self Healing in the Solaris 10 Operating System," June 2004.

[3] Antonio Carzaniga, et al., "Self-Healing by Means of Automatic Workaround," Proc of SEAMS'08, May 12–13, 2008, Leipzig, Germany, pp17-24

[4] Julian B. Grizzard, et al., "Towards a Trusted Immutable Kernel Extension (TIKE) for Self-Healing Systems: A Virtual Machine Approach," Proc. of IEEE Workshop on Information Assurance and Security, West Point, NY, pp 444-446, 2004.

[5] Consortium for Service Innovation, "The KCS[sm] Operational Model (Knowledge-Centered Support), Version 3.7".

[6] Consortium for Service Innovation, "HP Non Stop Customer Support, Knowledge-Centered Support [sm] (KCS) at Work".

[7] Brian Cook, et al., "Toward Self-Healing Multitier Services," Proc. of IEEE International Data Engineering Workshop**,** 17-20 April 2007, pp 424-432.

[8] L. Baresi, et al. "Towards Self-Healing Service Compositions", Proc. 1st Conference on the Principles of Software Engineering, Buenos Aires, Argentina, 2004

[9] Ghita Kouadri, et al., "On Modeling and Developing Self-Healing Web Services Using Aspects," Proc. Of the IEEE Conference on Communication Systems Software and Middleware, 2007. COMSWARE 2007, pp 1-8.

[10] Goranka Medhi, et al., "Pranayama*: A predictive self-healing technique for fault-tolerance in Adaptive Enterprise Management, HP Tech Con Asia

[11] IBM White Paper, "An Architectural Blueprint for Autonomic Computing", 2006

[12] Rimma V. Nehme, "Database, Heal Thyself," Proc of IEEE ICDE Workshop 2008, pp 4-10

[13] Richard Sarwal, "The Self-Healing Database: Strategies and Directions"

[14] Boeing News Release, "FAA Approves Boeing 787 Dreamliner Maintenance Program", Dec. 22, 2008, www.boeing.com/commercial/news/2008/q4/081222b_nr.html