A Business-Driven Approach to Closed-loop Management

Mathias Salle, Akhil Sahai, Claudio Bartolini, Sharad Singhal
HP Laboratories Palo Alto
HPL-2004-205
November 9, 2004*

close loop management, MBO, business-driven management, design, redesign, SLA, policy

In this paper we describe a model based approach to making resource allocation decisions driven by the value of those decisions to the business. We believe this enables a generic approach to realizing close-loop management. Our solution is centered on two technologies being developed at HP Labs: Quartermaster and Management by Business Objectives (MBO). Our approach was validated by a demonstrator built using these technologies, and other commercially available HP products.

* Internal Accession Date Only
Approved for External Publication

A Business-Driven Approach to Closed-loop Management

Mathias Sallé, Akhil Sahai, Claudio Bartolini and Sharad Singhal

HP Laboratories
1501 Page Mill Road
Palo Alto, CA94303, USA

Abstract

In this paper we describe a model based approach to making resource allocation decisions driven by the value of those decisions to the business. We believe this enables a generic approach to realizing close-loop management. Our solution is centered on two technologies being developed at HP Labs: Quartermaster and Management by Business Objectives (MBO). Our approach was validated by a demonstrator built using these technologies, and other commercially available HP products.

1 Introduction and Motivation

Businesses are increasingly dependent on their IT environments for critical business functions, where every business "event" triggers corresponding IT events. IT systems therefore have the ability to significantly help (or hinder) business by handling (or not responding to) these events. As business needs change, it is therefore increasingly important that the underlying IT systems also change to allow the business to run smoothly. A critical issue within IT systems is one of resource allocation—how much resource to allocate to which service. Usually, mission critical systems are over-provisioned to ensure that they are always available when needed. However, even elsewhere, enterprises have typically found that resources are underutilized. As enterprises move towards virtualized environments where resources are shared and dynamically allocated between various applications, decisions need to be made about how this resource allocation should be done. The problem is made more complex by the fact that resources may be shared between multiple lines of business, and applications of different business criticality.

It is thus important to understand not only how much resources are required by applications, but also the *value* of those applications to the enterprise when making resource allocation decisions. Similarly, when responding to IT incidents or creating change management plans, it is important to understand the business impact of those incidents or changes.

Thus, shared IT resources must be managed according to criteria that maximize the business value of those resources for the enterprise. As the pace of change increases, rapid and effective decision making becomes part of automated IT operations, and the resource assignment criteria must morph into the decision support capability of IT management solutions.

2 Closed-loop management

To enable rapid changes within IT based on business requirements, it is important that "closed loop" management systems be created. The goal of these management systems

would be to reduce the delay between the time business needs become visible (or business level changes are required) and the time when IT systems are ready to meet those business needs. Figure 1 shows a high level view of these closed-loop systems as a lifecycle. *Design and deployment* decisions are typically required for creating *service delivery systems*. Examples of service delivery systems include data storage systems, servers, and the networks that connect them, as well applications, services or portals that use the IT infrastructure to provide business functions. Once these systems are deployed, service levels are monitored within a *service delivery management* system. As business needs change, so do the requirements on the service delivery. As a result, *decisions* need to be made to adjust service delivery systems *based on business-level criteria*, which may lead to a re-design (or re-adjustment) of the service delivery system.

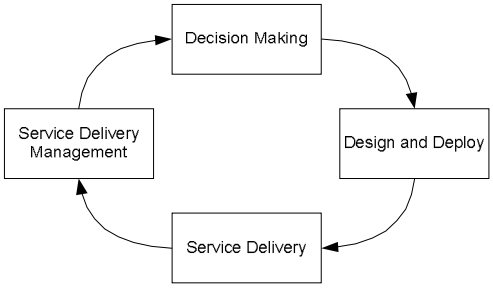

Figure 1: Close loop logical view

In the remainder of this section we discuss the key lifecycle stages in more detail.

2.1 Decision Making

The decision-making process within this loop needs to be tied to business value of the service delivery system. The effectiveness of the support to the decision making process is heavily dependent on modeling the dependencies between measures made at the service delivery level, and indicators that have relevance at the business level. On the other hand, there is a clear tradeoff between the cost of modeling and its effectiveness. For a closed-loop management system to be useful, the cost of modeling has to be kept low. This means that the elicitation of the preferences of the business has to be extracted from knowledge that is readily available. One typical way of capturing business requirements is through Service Level Agreements (SLAs) that define technical requirements on the service delivery systems. These service-level agreements have to be evaluated in the light of business objectives and financial/market and customer data to help make management decisions based on the business-level consequences of meeting or violating the outstanding SLAs. Useful knowledge on business requirements is also present in other forms within the enterprise (such as balanced scorecards [1], business objectives, key performance indicators etc.).

The decision problem that IT managers are faced with is one of assessing the business impact of their available *options*, or courses of action aimed at managing the IT delivery systems. To assess the business impact means to compute a measure of the *alignment* to the business objectives that is expected for each of the possible given course of action.

We define the alignment with a given business objective for an option as **the measure of the likelihood –** given the best knowledge about the current situation **– that the objective will be met**. Then IT managers need to be able to monetize the measure of alignment thus derived and use the monetization value together with other information on the cost of carrying out the respective course of action to rank the available options. The decision problem consists in ranking the options to decide what course of action to take. It is quite easy to see how options can be ranked based on their alignment with respect to one objective. The option's rank is as high as the likelihood of meeting the objective that it guarantees, given the best knowledge about the dynamics of the system available. Things are made more complicated when there is more than one objective to be considered when determining the best course of action. The relative importance of the different objectives is taken into account in determining the monetization value, and therefore the overall rank of the various options.

The generic decision problem just described can be cast into more specific ones depending on the particular domain of IT management that is of interest. For example, in the incident management domain, the problem is to prioritize among concurrent service incidents based on their impact on business objectives [5]. In that context, each option is a possible assignment of a priority value to the incidents. The prioritization that is finally chosen is the one that guarantees the optimal alignment with objectives that were propagated down from the business level, such as maximization of profit, or maximization of total customer experience (TCE), defined as a function of some key performance indicators (KPI). Knowledge about the domain is necessary to assess the impact of the incidents onto the value of the KPIs. Another instance of decision problem in the context of problem management is given in [4].

2.2 Design and deploy

As the business objectives change, the importance of various guarantees of service to be met by the underlying service delivery systems may also change. Thus, the management system may trigger various allocation, design, configuration, and deployment activities on the service deployment systems to enable them to meet the changing business requirements. Currently, many of these processes are manual, and hence lack the agility required for rapid re-configuration. Thus it is also important to automate the design, configuration, and deployment activities to make them more responsive typical resource allocation and design engine may provide the following capabilities:

1. Maintain and manage models of the infrastructure that capture the networking, storage, systems, virtual machines, applications and service details.
2. Create a configuration that meets users requirements keeping in mind the directives and policies specified by the administrators.
3. Undertake capacity allocation where reservations for resources are maintained and concurrent reservations about resource capacities are managed. These reservations could be about resources required in future.
4. Map Resource design to actual infrastructure instances when the reservations become current.

5. Deploy the designs by configuring resources within the resource pools, or re-configure existing systems to meet the new design goals.

Expanding on the second item of the list, resource design depends on multiple requirements:

- § users' requirements (that may be minimally specific),
- § Operator/administrator's constraints,
- § technical capabilities of the systems and the corresponding constraints

In a typical utility environment there will be thousands of components and a similar order of rules that dictate typical resource design. Based on user request a variety of resources, may be used to create a system design, these may be:

1. *Abstract, transient, virtual, polymorphic resources:* The user may request resources that can be realized in multiple ways and the design system determines how to instantiate the resource. For example a user may request a switch, and the design system may choose a CISCO Switch during instantiation. Some virtual resources have to be sometimes instantiated on the fly, e.g. virtual machines. Similarly polymorphic resources are those that can perform multiple functions and so have to be configured to perform a particular function.
2. Composite resources: One can request a resource that is composed from several other resources. Instead of asking for each of the component resources, a requestor could simply ask for the aggregate resource.
3. Constrained resources: One can request a resource that satisfies certain constraints – for e.g., constraints on a resource's properties, or on the associations it has with other resources.
4. Combination of the above: A request can be made for a resource that in turn contains composite resources. In addition, the requestor could specify constraints on top of the requested resource. An operator could specify an additional set of constraints that further restrict the design choices.

In the next section we will describe a prototype closed-loop management system that was built to follow the lifecycle described above.

3 A Prototype of a Closed-loop Management System

Figure 2 grounds the closed-loop management lifecycle into a system architecture view. In this figure, the service delivery system has been segmented into infrastructure services (e.g., networks, resource pools, storage pools etc.), application services (e.g., business applications, portals, etc.) and business services (e.g., revenue generating transactions, or inventory turns within a supply chain). At each level, different metrics that are important at that level are monitored. These metrics then feed into management systems that provide incident, problem and change management capabilities. The goal of the change management system is to control the underlying service delivery systems consistent with information provided by business level objectives as described in the previous section.

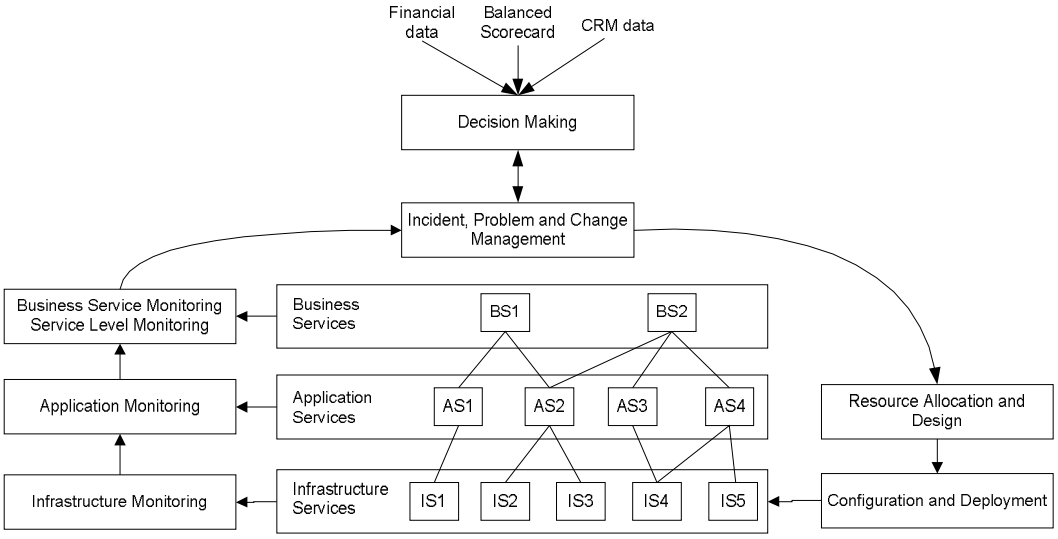

Figure 2: Generic Close Loop Architecture

In order to produce a proof of concept for the architecture discussed above, we have built a prototype system that brings together state-of-art technologies we are currently developing at HP Labs and a large number of commercially available products. The realization of the proposed architecture has been grounded in a specific scenario focused on a hypothetical financial institution, "First Agility Bank". In this scenario, the First Agility Bank achieves a high degree of synchronization between its lines of business and its IT through the adoption of design principles (standardization, simplification, modularity and integration) and close loop control systems that follows the architecture introduced in Section 2. Among the many business services that First Agility Bank runs, we concentrate on their wire transfer service. Figure 3 depicts the business process that underpins that specific service and highlights the set of applications (Check Funds, Check Identity, etc.) used for delivering the service.

The scenario also shows how the various applications are supported by the resources in the Utility Data Center. The IT function and the Line of Business responsible for the Wire Transfer service enter in a service level agreement in which the IT organization commits to running the service with the guarantee that the “average time to completion of wire transfer requests should be less than 30 ms”.

In that setting, the key challenge that the IT function faces is to manage its service delivery systems such that any degradation of service is proactively handled therefore minimizing the risk of violating the SLA and hence reducing the impact on the line of businesses. This challenge is made even more complex by the fact that it is advantageous for the IT function to share resources among many lines of business, in order to better leverage IT investments.

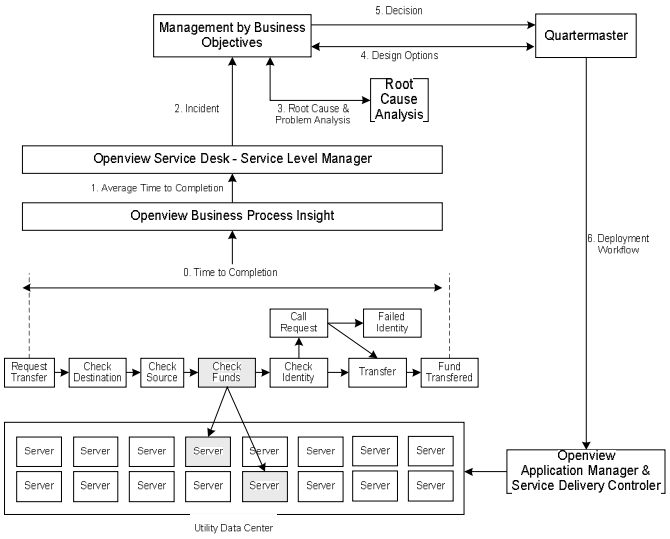

Figure 3: Closed-loop management system implementation

The close loop management illustrated in Figure 3 starts with the modeling and monitoring of the business service using Openview Business Process Insight (OVBPI) [12] as presented in the screenshot in Figure 4.

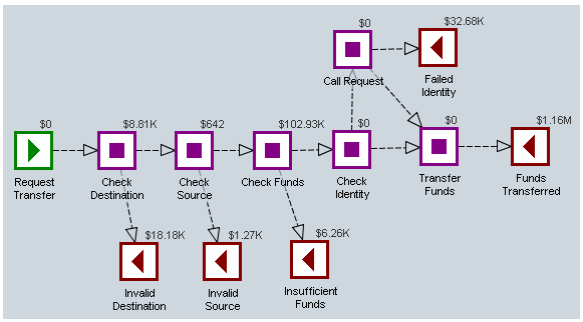

Figure 4: Business Process for Wire Transfer Business Service

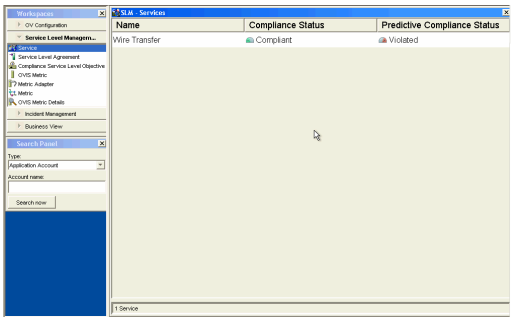

Figure 5: Monitoring the service level agreement using SD-SLM

OVBPI is used here to monitor the "Time to Completion" of each fund transfer request (step 0) and to determine the average time for time to completion over a reviewing period. That metric is then fed (step 1) to the Service Level Management module of Openview Service Desk (SD-SLM). As presented in the screenshot of Figure 5, SD-SLM monitors the metric values and computes the compliance to the SLA. SD-SLM also provides the ability to predict the compliance level at the end of the reviewing period. This allows for the proactive management of the delivery systems.

Let's now imagine that an unexpected load of service requests threatens the SLA compliance. This translates into a "Violated" predictive compliance status as presented in Figure 4. As a result, the proactive capability of the service level management system triggers an incident. That incident (step 2) is sent to the decision making system

(Management by Business Objectives – MBO) and is prioritized based on its relative impact on the business objectives of the IT function.

Figure 6 drills down into MBO to give a high level description of the system. MBO defines a generic information model that is populated through knowledge that is present in other forms within the enterprise (such as Balanced Scorecards, Business Objectives, Key Performance Indicators, etc.). The MBO reasoning engine solves the decision problem described in Section 2.1: it computes the *alignment* to objectives that is expected for each of the possible given *options*, or course of action aimed at managing the IT delivery systems. The engine is then able to monetize the measure of alignment thus derived and use the monetization value together with other information on the cost of carrying out the respective course of action to ranks the available options. On ranking the options, it returns a suggestion on what course of action to take, substantiated by the evidence that it has for assessing the alignment with respect to the business objectives.

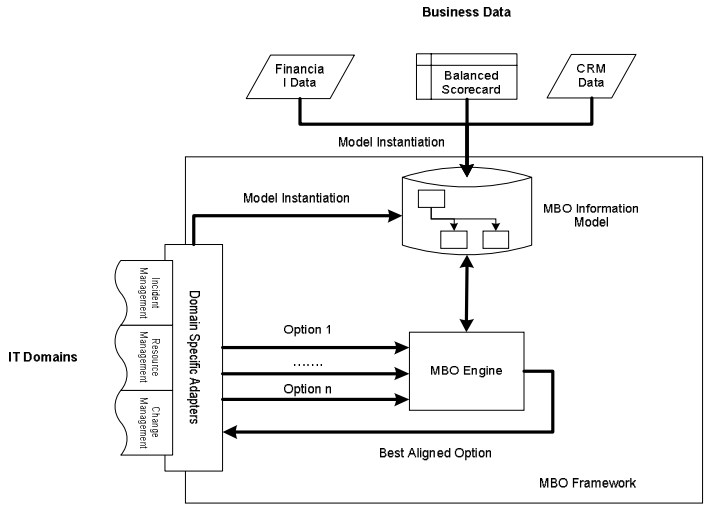

Figure 6: MBO high-level diagram

The MBO information model (Figure 7) is articulated around a pair of key concepts: Objectives and Key Performance Indicators (KPI).

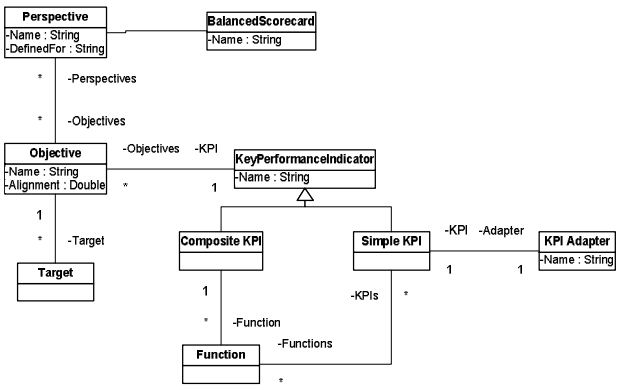

Figure 7: MBO Information Model

Objectives are expressed a target value over a key performance indicator, or KPI. KPIs are measurable indicators of *performance* of the enabling factors of IT processes, indicating how well the process enables the goal to be reached.

Figure 8 shows a screen capture of the business objectives of the IT function. The overall objective is to improve Total Customer Experience by 20%. This objective is based on a composite KPI defined over two supporting objectives. The supporting objectives are the reduction of 10% of the problems associated with the Touch Point Experience of customer, the other one being related to the End to End Transactional Experience. Each of the objectives comes into the definition of the composite KPI through a *weight factor* (a real number between 0 and 1) that in the example is 0.4 for the Touch Point Experience and 0.6 for the End to End Transactional Experience. The weight factor is multiplied by the expected alignment with the supporting objectives for each incident to get to a measure of the alignment with the top level objective. The quantitative measure of the alignment is translated in a qualitative measure by mapping intervals in the value of the alignment with description such as “aligned” (alignment value greater than 0.9), “slightly misaligned” (alignment value between 0.7 and 0.9) and so on.

Figure 9 presents the prioritization screen and shows the incident associated with the Wire Transfer. A relative ranking is established between the various registered incidents and a High priority is associated to that service. The bottom right hand pane shows the impact of that incident on the alignment of each of the supportive objectives. Here, the incident impacts the End to End Transactional Experience, moving it from an aligned state to a slightly misaligned state.

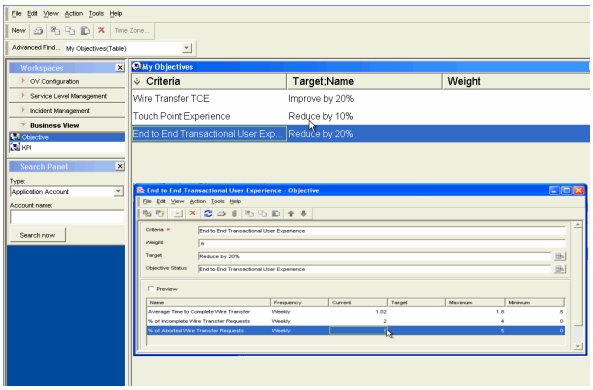

Figure 8: Business Objectives of the IT Function

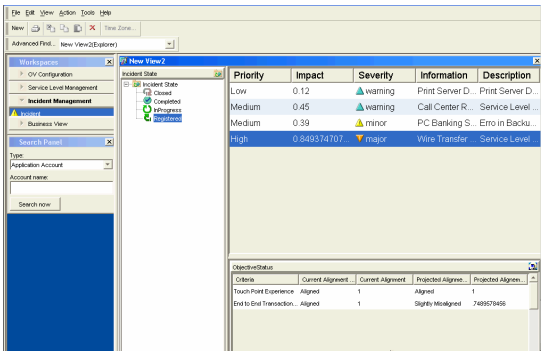

Figure 9: Prioritization of incident using Impact on Business Objectives

Once the root cause has been determined – in this case an under provisioning of Check Funds application - this problem is communicated to Quartermaster (step 4), the resource allocation and design tool, which redesigns the system to overcome this problem.

Quartermaster, is an integrated set of tools and technologies targeted at providing automation capabilities to utility computing environments [15]. Quartermaster tools currently provide the following capabilities to IT users and system operators:

- *Policy-based design and composition*: Quartermaster allows operators to capture system composition rules and best practices in models, and provides users the ability to automatically create custom designs that conform to those policies. This reduces the time to design applications and IT environments and reduces the likelihood of error in system design.
- *Capacity management*: Quartermaster includes scheduling and capacity management algorithms that can track complex patterns of time varying resource demands and react accordingly. This enables operators to manage infrastructure use and permits specific qualities-of-service to be achieved.
- *Resource assignment*: Quartermaster uses mathematical programming techniques to ensure that resource-level requirements (e.g., network bandwidth) of the application are met and bottlenecks are not created in the shared infrastructure when resources are assigned to applications. This enables efficient use of the infrastructure resources while ensuring application-level quality of service

Quartermaster policy based design and composition [3] enables the capability to capture domain knowledge in hierarchical models and constraints that can then be solved through a logic based solver for creating mathematically provable configurations. In order to achieve this, QuarterMaster extends CIM meta-model to incorporate the concept of policies. The current CIM meta-model does not provide the capability to capture such rules. We associate these rules with resource types. These policies capture the technical constraints and choices made by the operators or administrators that need to be obeyed by every instance of the associated class. Below is an example of a resource type declaration in Quartermaster, showing its MOF description as well as an example of policies that could be associated with the type. This is a small part of the actual model and policies

used for the prototype. The example shows how a class QM_Tier can be built by using a number of Logical Servers. These Logical Servers can be AppServers, Web Servers, or Database Servers depending on what kind of Tier is being designed. The Three Tier Site that was configured for the prototype contains three such tiers, namely a web server tier, an appserver tier and a database tier connected to three LANs.

```
[Version ("1.0.0"), IconUrl ("QM_Tier.gif"), Designable]
class QM_Tier : QM_Resource
{
      [Description (
        "The Cost of the Tier ") ]
    real32 Cost;
      [Description (
        "The Number of Servers in the Tier ") ]
    uint16 NumServers;
      [Description (
        "The Minimum Number of Servers in the Tier ") ]
    uint16 MinServers;
      [Description (
        "The Maximum Number of Servers in the Tier ") ]
    uint16 MaxServers;
      [Description (
        "The Number of Subnets the Tier is connected to") ]
    uint16 NumSubnet;
    string Name;
    [Description (
        "The Tier can be connected to two subnets") ]
    String ConnectedTo__0;
    String ConnectedTo__1;
};
[Version ("1.0.0"), Association, Composition]
class LogicalServerInTier
{
      [Key, Composite]
        QM_Tier REF Tier;

      [Key, Component]
        QM_LogicalServer REF LogicalServer;
};
instance of QM_ClassScopedPolicy
{
      Id = "ae-008-05-04";
      AssociatedClasses = {"QM_Tier"};
      Assertions = {
            "MinServers >= 1",
            "NumSubnet ==  1 || NumSubnet ==  2",
            "NumServers >= MinServers",
            "NumServers <= MaxServers",
            "LogicalServer <: QM_WebServer ||
             LogicalServer <: QM_AppServer ||
            LogicalServer <: QM_DatabaseServer”
       };
};
```

The users can request customization of particular resources from the available resource types by specifying additional constraints on their attribute values and on their arrangement in the system. These requests could be for instances of "raw" resources or for composite resources. Quartermaster automatically generates a system configuration by selecting the appropriate resource classes and assigning values to their attributes so that all constraints specified in the underlying resource models are satisfied. We have developed a tool, called Cauldron, that solves these problems. The input to Cauldron is a set of class definitions with attributes and relationships (e.g., composition, association, references, and supertypes) defined. Policy constraints are embedded in the classes as satisfy clauses, expressed by combining the class attributes and relationships using a subset of first-order logic and linear arithmetic. Cauldron, in turn, uses a theorem prover based on a fast SAT solver to assign values to the attributes in the satisfy clauses such that all constraints are satisfied. The requested system is specified to Cauldron as a distinguished class (main), which can have user policies on the requested system embedded in it.

All the classes, their inheritances, associations with other classes and constraints are specified. As these constraints are first-order logic expressions they can be solved using SAT Solvers to create system instances that meet all the constraints. As soon as the design is completed the corresponding resource capacities are allocated so that the design can be deployed at the requested dates. Sometimes if the infrastructure on which the system is being deployed has potential of network bottlenecks, Quartermaster resource assignment tool may be used to assign resources optimally to machines in the infrastructure [15].

Based on the initial Check Funds application model depicted in Figure 10, Quartermaster determines a more complex design based on the new requirements.

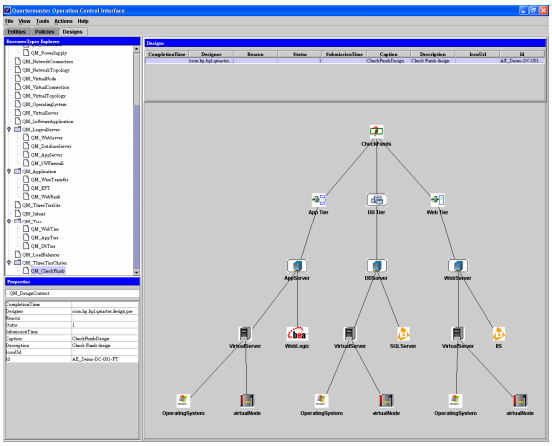

Figure 10: Logical View of the initial design

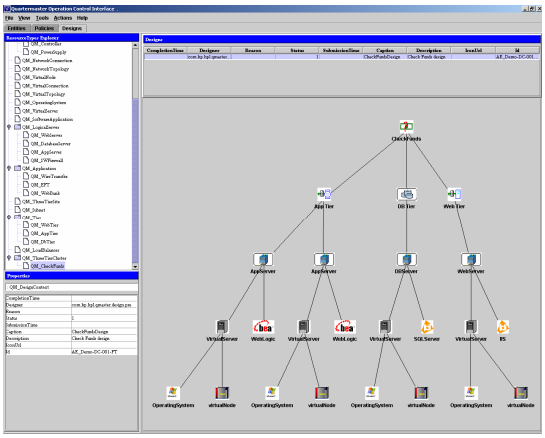

Figure 11: Logical View of the result of the re-design operation

These options are then passed back to the decision system that decides on the best allocation based on its impact on the business objectives of the IT function. Deployment and implementation of the chosen solution is then conducted using Openview Service

Delivery Controller [14] and Application Manager [13], the configuration and deployment systems. The result is the addition of a new application server (Figure 11) to the pool of application server supporting check fund application. Once properly provisioned, the business service returns to normal performance behavior hence conforming to the SLA.

4 Related work

Driving IT management from business objectives is quite a novel proposition. In [11], Buco et. al. present a business-objectives-based utility computing SLA management system. The business objective(s) that they consider is the minimization of the exposed business impact of service level violation, for which we presented a solution in [4]. However, the Management by Business Objective (MBO) technology presented in this paper goes far beyond just using impact of service level violations. It provides a comprehensive method for IT management that can take into account strategic business objectives; thereby, going a long way towards the much needed synchronization of IT and business objectives. For a more detailed discussion of the MBO capability applied to the incident management domain see [5].

There is some work that has been done in the context of design. Introducing constraints in UML specification of systems for configuration purposes is discussed in [16]. They define a set of construction rules at one place termed a domain. In that sense the approach is similar to expert systems. In Quartermaster approach, we embed constraints hierarchically thus distributing constraints on to various resource types, and taking into account these constraints as the construction happens as opposed to creating a large number of constraints (rules) a priori. Our approach enables flexibility and extensibility in specification of constraint and in automatic construction depending on the user requirements. The differing user requirements may result in one construction being different from another. We have also applied the concept to CIM, which is de-facto standard for management of infrastructure. The ClassAds MatchMaking work [17] assumes that the match-maker matches the requestor entity's request against the provider entity's ClassAds (which are specifications in a semi-structured language). The assumption is that all the resources (like machines) exist a-priori and have been advertised. In a resource-utility environment however, some of the resource instances may not even exist a-priori (as is the case with transient/virtual resources) or may be logically constructed resources that have to be instantiated on-demand (e.g. appserver/tier/farm/e-commerce site). This causes a problem for approaches that undertake match-making only on instances. We enable construction on-the-fly by embedding constraints hierarchically in *the resource types* as described in this paper. The same concepts are extensible to resource instances as well. It is also not clear whether the ClassAds language supports first-order logic and linear arithmetic. As we have shown in the examples, it is important to have notions of quantifiers, implications, equivalences and other first order-logic expressions for reasoning.

Closed-loop management is also a well-researched area. Multiple approaches to closed-loop management exist. Control-theoretic approaches to closed-loop management

involves identifying system transfer functions and designing controllers for certain specific products (Web Server [6], [7] Lotus Notes [8]). These approaches are highly focused on undertaking closed-loop management on specific products and do not tackle the closed-loop management in a generic manner for a variety of products involved in a typical design. Other approaches involve using expert systems or pre-specified policies to undertake changes in the system design based on performance/monitoring/failure data [9], [10]. These expert systems are usually case-based systems where possible scenarios are specified as event-action pairs. The problem in such approaches is that if the system reaches an un-previewed state the controller becomes redundant and humans have to be involved in closing the loop.

5 Conclusion and future work

The success reported by the demonstrator described above is very encouraging. We were able to demonstrate an approach to closed-loop management. In the process, we were able to create an end-to-end solution using HP management software products and HP labs prototypes together and to carry out a technology gap analysis. Building the demonstrator resulted therefore in an exercise of a loose-coupled integration of software and systems.

We believe that a greater benefit will derive from a closer coupling of the technologies presented here (namely Quartermaster and MBO). Our approach to bringing Quartermaster and MBO together is based on definition of a common information model that touches on many aspects of the IT resources and services and business objectives through the management lifecycle.

References

[1] Nils-Goran Olve, Roy J, Wetter M. Performance Drivers. John Wiley.

[2] Sahai A, Singhal S, Joshi R, Machiraju V. Automated Configuration Generation through policies. In the proceedings of IEEE Policy 2004.

[3] Sahai A, Singhal S, Joshi R, Machiraju V. Policy-based Automatic Construction of Resources. In the proceedings of IEEE/IFIP NOMS 2004.

[4] Sallé M.; Bartolini C.; "Management by Contract", In the proceedings of IEEE/IFIP NOMS 2004.

[5] Bartolini, C.; Salle, M.; "Business Driven Prioritization of Service Incidents", In the proceedings of DSOM 2004.

[6] Lu C, Abdelzahar, T.F., Stankovic, J.A., Son S.H.: A feedback control architecture and design methodology for service delay guarantees in web servers, Technical Report CS2001-06, University of Virginia, Department of Computer Science, (2001)

[7] Xue Liu, Lui Sha, Yixin Diao, Steve Froehlich, Joseph L. Hellerstein, Sujay Parekh: Online Response Time Optimization of Apache Web Server. IWQoS 2003: 461-478

[8] Joseph L. Hellerstein, Neha Gandhi, Sujay Parekh: Managing the Performance of Lotus Notes: A Control Theoretic Approach. Int. CMG Conference 2001: 397-408

[9] Devarakonda M et al. Policy-based Autonomic Storage Allocation. In the proceedings of IEEE/IFIP DSOM 2003

[10] Lymberopoulos L, Lupu E, Sloman M. PONDER Policy Implementation and Validation in A CIM and Differentiated Services Framework. In the proceedings of NOMS 2004

[11] Buco M. et al., "Managing of eBusiness on Demand SLA Contracts in Business Terms Using the Cross-SLA Execution Manager SAM", IBM, In the proceedings of IEEE ISADS, 2003.

[12] HP Openview Business Process Insight http://www.openview.hp.com/products/bpi/

[13] HP Openview Application Manager http://www.openview.hp.com/products/radia_appm

[14] HP Service Delivery Controller http://www.openview.hp.com/products/sdc
[15] Singhal S, Graupner S, Sahai A et al. Quartermaster-A Resource Utility System. HPL-2004-152.
[16] Felfernig A, Friedrich G. E et al. UML as a domain specific knowledge for the construction of knowledge based configuration systems. In the Proceedings of SEKE'99 Eleventh International Conference on Software Engineering and Knowledge Engineering, 1999.
[17] Raman R, Livny M, Solomon M. MatchMaking: Distributed Resource Management for High Throughput Computing. In the proceedings of HPDC 98.