# Prospects for simplifying ITSM-based management through self-managing resources

D. M. Chess
J. E. Hanson
J. A. Pershing, Jr.
S. R. White

Information technology service management (ITSM) codifies and supports the current best practices in the management and governance of existing IT infrastructures, including the computing infrastructure that underlies service delivery. Once ITSM tools have identified and structured current best practices, there is a significant opportunity to simplify those practices, and thereby ITSM in general, by introducing self-managing resources (SMRs). SMRs and related technologies allow increased delegation of existing ITSM tasks from humans to autonomic managers and the restructuring of ITSM activities through process modification and task replacement or elimination. In particular, the use of SMRs and virtualization can convert many activities that currently require multiple planning, validation, and approval tasks into routine activities that have a simpler task flow, in much the same way that modern file systems have transformed file-layout tasks that formerly required a skilled administrator into tasks that are handled entirely and transparently by the operating system. This paper briefly describes the general principles of SMRs, explores a number of potential impacts that this technology will have on ITSM processes, and illustrates these ideas with an analysis of how selected ITSM flows may be transformed.

## INTRODUCTION

Managers of information technology (IT) systems face a daunting challenge in trying to administer them, as their size and complexity are continuously growing. In order to bring this complexity under control, many large IT organizations are adopting various "best practice" methodologies, herein called *formal processes*, to distinguish them from the all-inclusive term *IT process*, which can mean anything from a running application program to a workflow of any kind. Formal processes such as ITIL** (IT Infrastructure** Library),[1] ETOM** (Enhanced Telecom Operations Map)**,[2] and IBM Service Management,[3] bring order and discipline to the

management of IT systems, thereby reducing the likelihood and mitigating the severity of system failures due to systems management activities.

The need for formal processes is in large part driven by the fact that today's IT systems, and enterprise software systems in particular, are generally brittle,

■ An ideal computing system would take over a great deal of its own management ■

difficult to understand, and dangerous to change. The risks involved in most management tasks are such that they require a formal process to ensure that they are done properly. With the value of formal processes comes a cost, of course—namely, that formal processes tend to be complex themselves, thereby placing significant and sometimes prohibitive burdens on the system administrators charged with carrying them out.

One approach to reducing the burden that formal processes impose on system administrators is to implement an automated management support infrastructure to help in process execution. A longer-term approach is to find and to remove, where possible, the underlying factors that made the adoption of a process necessary in the first place. Both of these approaches should be pursued. Without a management support infrastructure, those management tasks that require governance by a formal process will be prohibitively slow, expensive, and prone to human error. Yet, unless underlying factors such as brittleness are engineered out of the systems being managed, the savings in time and effort afforded by a management support system will be dwarfed by the irreducible costs of human decision making, agreements, and approvals that are built into the formal processes that the infrastructure is supporting.

This paper takes the second of these two approaches, focusing on the question of how certain changes in the architecture of an IT system can permit radical simplification in the formal processes to be applied to it. The following two sections present a sketch of an ideal system from the point of view of its administrators, and, in contrast, some of the significant trouble spots encountered in current practice. Next, the fundamental concept of a *self-managing resource* is introduced, followed by a discussion of how self-managing resources can simplify and transform formal processes for managing IT systems.

## Portrait of an ideal system

From the viewpoint of the ease and flexibility of its management, what would the ideal computing system look like? If nothing else, it would have to take over a great deal of its own management so that its administrators would have little to do, and it would have to be scalable in order to handle its own growth and evolution. In the following, we ask the reader to suspend disbelief and consider what a true self-managing, scalable IT system would be like.

From the customer's view, a self-managing, scalable IT system consists of a number of more or less identical boxes. Customers who want the "boxes" (machines) to be able to communicate quickly with each other can run cables between them, plug them into a switch, or get the rack-mounted versions; other customers can simply place the machines in the same room, and they can communicate wirelessly.

The system is administered through a Web interface (one that has been rigorously tested for usability). Most customers attach one or more network cables from their intranet to one or more of the machines; if a customer decides not to do that, he or she can still communicate with the system wirelessly. The Web interface is mainly used to set a few high-level policies for how the system should manage itself (unless the default policies are satisfactory).

One of the most important functions of the system is to provide information to the customer on its own utilization and performance and indicate when it needs to be expanded. The customer can then either contact the manufacturer and order whatever is needed or whatever his budget currently allows, or if the right policy is enabled, the system itself can place the order automatically, as required.

When the new machines arrive, they need to be placed in the room with the other machines (or perhaps be inserted into a free rack slot if the rack-mounted version is used). The system itself then configures and provisions the machines appropriately.

To deploy a new application into the system, the customer creates a description of the desired characteristics of the application, including functionality, user interface, and performance, and the system then goes to the Web site of any supplier of compatible applications, browses for appropriate applications, and returns data about the applications to the customer. The system informs the customer whether the new application is compatible with its existing capacity, how many more machines it will require if not, and any other consequences of adding the application to the system. If the customer is satisfied with the results, he or she gives the site a credit card or customer number and presses "deploy". After some appropriate amount of time, the application is installed into the local system. If the application requires any new policy decisions that are not yet set, it will ask the customer for those decisions (or accept the defaults).

Over time, changes need to be made to the system: patches need to be applied, and software components need to be upgraded. In our idealized scenario, such changes are handled automatically by the system itself, in much the same way that new applications are deployed, as just described. Periodically, the Web sites of the suppliers are checked for patches and upgrades; these are annotated with indications of their importance (e.g., "critical security patch") and their expected impact on the operation of the system (e.g., "consumes 20 percent more memory and is 15 percent faster"). In some cases (e.g., changes of critical importance with minimal expected impact), the system will download and apply the upgrade automatically. In other cases, the customer will be notified of the pending upgrade, its impact, and its cost (e.g., more boxes or more memory), and will decide whether to apply the upgrade or not. In all cases, there is an "undo" facility available, so that the upgrade can be reversed if the customer changes his or her mind.

Sometimes something goes wrong with one of the boxes. This does not cause any significant externally visible failure (beyond, perhaps, a dropped HTTP connection). The system usually fixes these failures itself, by restarting an aborted application or rebooting a nonresponsive box. Sometimes a component is actually broken, in which case the system stops using it and causes a failure light to become active. Once a month, the customer gathers any boxes with active failure lights and ships them back to the manufacturer for a prorated rebate.

The preceding scenario describes all that is required of the customer for systems management and expansion. In this ideal system, the management tasks are extremely simple, inexpensive, reliable, and nondisruptive. Unfortunately, this system will not be available in the near future. Science fiction though it is, this "thought experiment" highlights some desirable directions for the future of ITSM that might be achievable long before we reach the goal of fully self-managing systems.

There are two key features of the ideal system that we have described.

1. *Changes are safe*—There is no need to construct expensive review, approval, and rehearsal processes before making a change because changes are safe and easily undone. Before a change is made, it is possible to ask the system itself what the likely consequences of the change will be.
2. *Actions are policy-driven*—Rather than requiring human planning and intervention for every action that the system takes, a system made from self-managing components is driven by higher-level policies that can be specified in advance and which guide the automatic actions that the system takes in response to change.

We now return our focus to the real world and examine some of the trouble spots of actual IT systems in use today before going on to examine the movement from present systems toward self-managing policy-driven systems in which change is safe and systems management is simple.

## Systems management trouble spots

Perhaps the largest problem in IT systems management today is the lack of comprehensive information about the IT resources themselves: what is installed, where it is installed, how it is configured, and so on. Information about IT resources is generally kept separately from the resources themselves, if it is kept at all. Centralized databases of information about system components are an advance over poorly organized or disorganized information, but they still suffer from various problems. One prominent problem is that the databases inevitably become inconsistent with the actual states of the resources as they change.

Significant and expensive processes are necessary to minimize the frequency with which this happens, to detect it when it does happen, and to fix it when it is detected.

The lack of information about specific resources is a problem, but the lack of information about how and why the various resources are related to one another

■ Perhaps the largest problem in IT systems management today is the lack of comprehensive information about the IT resources themselves ■

is a still more serious and difficult problem. Problem determination and failure analysis become more difficult, and predicting the impact of a proposed change to a resource is nearly impossible. Information about relationships may not be visible to the resources at all but typically is implicit in the construction of the system or known only to a third party.

Capacity planning is difficult due to this lack of relationship information; specifically, information on the relationship parameters that are related to, or necessary for, the fulfillment of a particular service level agreement (SLA). Not only are the quality of service (QoS) expectations for a real system often unwritten and inchoate, but the particular structure and configuration of the system that are critical to meeting those expectations are often recorded only in the memories of the experts. For this reason, various changes to the system require hours, days, or weeks of reviews and rehearsals.

Because of these factors, it is both difficult to determine the underlying cause when the behavior of the system changes undesirably and to determine the safety or impact of desired changes. We therefore need to establish formal processes, which are often complex and expensive themselves, both for problem determination and for change management.

## Self-managing resources

To a large extent, the trouble spots described earlier can be addressed through the use of *self-managing resources* (SMRs) operating in a service-oriented architecture.[4] SMRs have been described in detail elsewhere, in particular in Reference 5, where they are called *autonomic elements*. An SMR is an IT resource—that is, a component of an IT system—in which the greater part of what is traditionally regarded as management of the resource is performed by the resource itself. SMRs are active entities exhibiting many of the properties of software agents, in particular the capabilities to operate relatively autonomously and to send and receive messages to and from other SMRs. They interact in prescribed ways (as described in the following) to collectively manage the behavior of the IT system as a whole. All this implies that there is a certain minimum level of functionality for a resource to be an SMR, which, in turn, implies a certain minimum size and complexity for a single SMR. Thus, for example, it is not practical for each individual file in a file system, or row in a database table, to be an SMR; rather, the file system or database itself would be an SMR.

Self-management at the resource level is not an all-or-nothing proposition, requiring complete or near-complete adoption before its benefits can be realized. On the contrary, every incremental advance toward full self-management significantly reduces the brittleness and complexity of the task of managing the IT system.

A detailed discussion of SMRs can be found in Reference 5; here, we only provide a brief review of their most important features:

- *Self description*—SMRs provide, in response to queries, detailed information about themselves, including their current operating parameters and status, the services they can supply and need to consume, and their current relationships with other SMRs.
- *Self registration*—SMRs register their existence and a summary of their self-description information with at least one *registry service*.
- *Policies*—SMRs shape their behavior in accordance with declarative policies, which may be given to them from another party in the appropriate circumstances. SMRs only accept policy updates from authorized sources and check all policy updates for consistency before using them. Consistency checking is done to determine the

effect of incorporating the update on the overall set of the SMR's policies and agreements.
- *Explicit management of relationships through agreements*—SMRs form and follow agreements with other SMRs in order to coordinate their collective behavior. Agreements are like policies in that the SMR shapes its behavior in conformity with them; but, unlike policies, agreements are under the explicit control of the SMRs that formed the agreement.
- *Interaction integrity*—SMRs do not permit access to their internal state through low-level interfaces. That is, all interactions with SMRs are done in such a way that the SMR's own policies and agreements are brought into play—for example, in formulating the response to a query or command. This is necessary to support the requirement that SMRs obey their own policies at all times and obey the constraints imposed by the agreements they have entered into.
- *Impact analysis*—SMRs can provide information about what would happen if certain changes were made or certain preconditions were met. This information need not be elaborate and may consist simply of an indication that the proposed change would or would not cause the SMR to violate its policies or agreements. If the change in question is a policy update, the logic required is the same as that which is used in checking the consistency of an update.

Although each of these features is significant for implementing SMRs, incremental adoption of a subset of these features can be beneficial for system management. Simply making all resources self-describing and self-registering would be a major advance over today's typical situation.

To help knit the collection of SMRs together into a functioning IT system, a number of infrastructural services are necessary. These are services (provided, of course, by SMRs) that facilitate the interactions among SMRs; they are closely analogous to the services provided by "middle agents" common in multi-agent systems.[6] Some of these services have analogs in present IT systems; others are necessary only in systems composed of SMRs. A full list of services can be found in Reference 6; here, we mention only four:

- *Registry*—The registry service acts as a network-accessible repository of information about the SMRs in the system. As described previously, SMRs register themselves with the registry service. They also use it to find other SMRs that can provide the services they need to consume.
- *Policy repository*—The policy repository is a network-accessible service that provides policies to SMRs. SMRs are notified by the repository when an update is available and take action to fetch the updates as and when appropriate. This "pull" model of updating policy is a natural consequence of the SMR's requirement to manage itself.
- *Resource factories*—Many kinds of SMR can be instantiated dynamically in response to the needs of the overall system (e.g., to cope with changing workloads). In such cases, the work of creating an SMR is done by a "factory". Once the SMR has been started and given initial default configuration information (such as its base policy), the SMR takes over the responsibility of managing itself.
- *Sentinels*—A sentinel monitors the "health" or life-cycle states of any number of SMRs and can report (to subscribing SMRs) on changes in those states. This enables systems-management software to automatically discover the failure of an SMR in order to restart it, replace it, or take other remedial action. Not all situations require sentinels; often, a problem with one SMR will be detected by other SMRs to which it is (or was) providing service. But when it is necessary or desirable to actively monitor the health of a particular SMR independent of failures in the services it is providing, sentinels can perform a useful function.

## SIMPLIFYING IT SYSTEMS MANAGEMENT

SMRs, or even resources that are somewhat more self-managing than those we have today, have the potential to radically simplify IT service management. The incorporation of SMRs into computing systems will allow for new, greatly simplified, flows that can be automated more easily for many of the high-level tasks of IT systems management. Just as computer chess programs using very different algorithms achieve the same goals as human players, SMRs will naturally enable IT management flows that take advantage of the ability of computers to track large amounts of data and perform repetitive tasks, while deferring to humans when situations arise that are beyond their automatic abilities. In those cases where human intervention is required, the incorporation of SMRs will make the jobs of the system administrators easier because

they will be able to take advantage of the informational and self-management abilities of the computing resources.

An inspection of the formal processes for IT systems management shows that they tend to be very labor intensive. This is largely because, in today's systems, only the administrators have access to information about what is occurring in the system (e.g., how database A is related to application B, what the capacity of server C is, and so on). This information is largely unavailable to the system components themselves, even if they were capable of exploiting it. Thus, only the administrators are in a position to determine the consequences of some contemplated management task. Even if the execution of the task were automated (e.g., by a systems management framework), the context in which the task is to be carried out, and hence the conditions governing its desirability, are still largely inaccessible to systems-management software.

In a world populated with SMRs, each resource would "understand" a great deal of this information. The understanding of its own properties, parameters, and runtime state would have been coded into it, as required by its ability to describe and manage itself. The understanding of its relationship to the overall system (i.e., the context within which it operates) would be captured by the set of agreements to which it would be a party.

The built-in ability of SMRs to protect themselves against damaging management commands (i.e., where executing the command would lead to a violation of the SMR's agreements) greatly reduces the risk involved in sending such commands. The SMR in question would respond to the instruction with a message saying, in effect, "I cannot comply because doing so would violate agreement X." The need for multiple checks and approvals among multiple administrators or stakeholders, as prescribed by the formal process flow, is largely obviated. A formal process would be needed only to override this behavior, in order to force the SMR to comply with the command despite its agreements.

For example, an inexperienced administrator might attempt to shut down a database server for routine maintenance, not knowing that the database was in use by a business-critical application at the time. (We may suppose that an unexpected failure in another database server had necessitated an unplanned switchover to the one in question.) If that database is an SMR, it would reply to the shutdown command with the message, "I cannot shut down now because it would violate my agreement with the business-critical application," thereby preventing a possibly costly error.

This in itself permits a radical simplification in the operation of systems-management software. In many of the ITSM flows there are "fast path" branches in which (for example) a particular change request is classified as pre-approved; if a particular type of change is pre-approved, then much of the complexity of the overall flow is immediately bypassed. The introduction of SMRs in the IT system means that a greater percentage of tasks are error-proof, which has the effect of permitting the fast path to be taken more often, resulting in significant savings in cost and time.

This is not the only type of simplification that SMRs can achieve. Just as SMRs can protect themselves (and therefore the system of which they are a part) from administrator errors, they can also protect themselves from damaging actions by ordinary users, and, even more important, by other SMRs. This permits a fundamental realignment of the relationship between the IT system and the processes charged with managing it. Instead of interacting with a systems-management support framework (e.g., submitting a change request), users as well as administrators will be able to interact directly with the system's components for most of what they need done, just as they do at present for such routine tasks as creating files, changing passwords, and so forth. If the SMR receiving the command determines that the request is safe to execute (as determined by its policies and agreements), it will simply do so; otherwise, it will refuse. Instead of submitting a change request, for example, a user simply tries to make the change he wants, with the understanding that if it cannot be done safely, it will not be done at all. The policies of the SMR that receives the request determine how or whether the request and any subsequent actions are logged, so that changes can be selectively tracked.

A third type of simplification enabled by SMRs takes advantage of their ability to form and modify agreements. This permits sophisticated adaptive algorithms to be built into the SMRs themselves, so

that they can modify their own behavior (including the services they consume) in response to changes in demand for the services they provide. Thus, for example, SMRs permit a simplified process of deploying a new J2EE** (Java** 2 Enterprise Edition) application in which a user places the application code in the data center's deployment repository and then tells the application to deploy itself. In order to do so, the application (itself an SMR) must have been started at least to the extent that it can receive messages and execute its self-deployment task; this can be done automatically by the deployment repository. The application then searches the data center for appropriate hosting containers (e.g., WebSphere* Application Servers) and negotiates with these potential hosts for the necessary conditions (e.g., performance), eventually striking a formal agreement with the best one. This hosting container, in turn, may have to renegotiate the agreements that it already has (e.g., to ask for more storage capacity in a database).

Ultimately, we envision a data center where most of the service management processes are fully automated: guided, of course, by policies which have been set by the administrators. When a circumstance arises that cannot be handled by the automated SMRs or where there is no guiding policy, the task is escalated to the administrators for action. Even then, in many cases, the administrators can interact with the system components in a relatively straightforward way. Only in a small fraction of cases will it be necessary for the administrators to enact a formal process to ensure that the task is performed safely. For example, formal processes are likely to be needed in order to change or override system-level policies that affect the behavior of multiple SMRs. In such cases the possibility of seriously compromising the system's behavior by applying an incorrect policy remains very real.

To illustrate the value that SMRs can provide in simplifying ITSM processes, we consider a specific, admittedly narrow, scenario in which the system to be managed is a homogeneous collection of virtual servers, and show how SMRs can simplify the process of adding or removing a server and, potentially, deploying software. In this scenario, each virtual server is an SMR, as is the "hypervisor" that provides the virtualization layer between the physical hardware and the virtual servers. (In terms of the infrastructural services listed previously, the hypervisor is a resource factory.) The hypervisor running on each physical machine also doubles as a sentinel for the virtual servers running on that machine. Each virtual server is operating under an explicit agreement with the hypervisor, which spells out the levels of base resources (CPU, disk, memory, etc.) allocated to that server by the hypervisor, as well as the lifetime of the agreement itself.

■ Self-managing resources have the potential to radically simplify IT service management ■

In our example, a user of the system wishes to have a new virtual server allocated to him in order to test new software that he has developed. The user sends a request for a new server to the hypervisor, including the base resource levels needed and the lifetime of the agreement. The hypervisor examines the set of agreements currently in place with the extant virtual servers and determines whether it can create a new virtual server with the desired properties without violating the terms of any of those agreements; for example, whether there are sufficient resources (CPU, memory, storage, etc.) that are not already committed to the existing agreements and that can be committed to this new request. If so, it allocates the new virtual server, gives it an initial policy granting administrator access to the requesting user, and sends the user the necessary information about the new machine. Also, an agreement is put in place between the requestor (the user) and the newly created virtual server, indicating that the server has agreed to provide the user with the base resource levels that were requested.

The relevant ITSM process here is change management, which is depicted in ***Figure 1***. The activities that can be eliminated are indicated with red cut circles. The essential feature of SMRs that makes these activities unnecessary is the existence of the agreements between the hypervisor and the servers and the built-in goal of each SMR to obey the agreements it currently has.

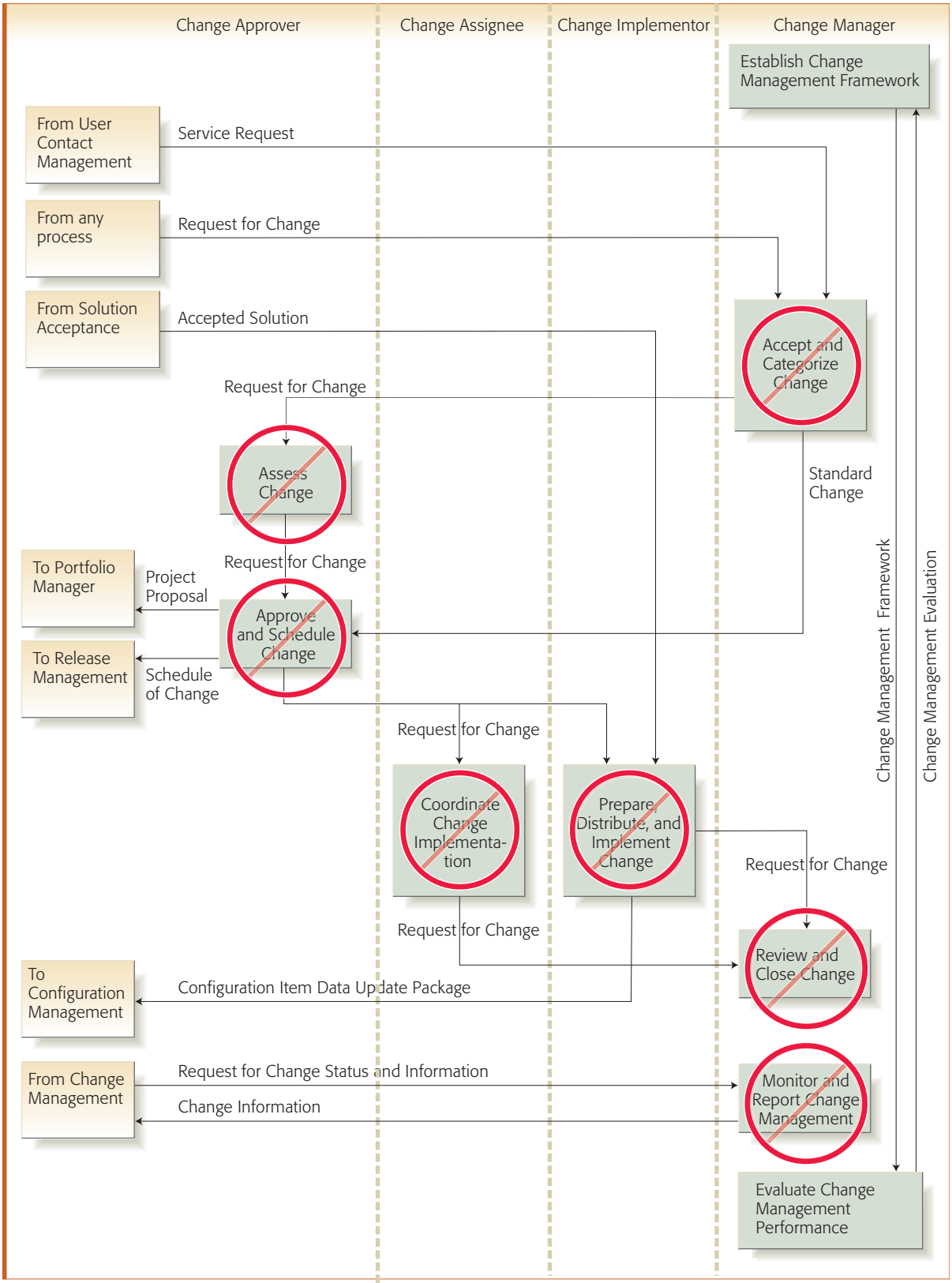


**Figure 1**
Overview of change management

We can modify the scenario somewhat by assuming that the physical machine did not have sufficient free local disk space to accommodate the new virtual machine or, due to some policy, external storage-area-network (SAN) storage is to be used instead of local disk space. In this case, when the request for the new server arrives, the hypervisor needs to allocate storage from the SAN, which is also an SMR. The hypervisor attempts to renegotiate the existing agreement that it has in place with the SAN, asking for an increase in its storage allocation. If the SAN agrees to the new terms of the agreement, then the storage is allocated to the hypervisor's physical machine, permitting the hypervisor to satisfy the user's request. If the SAN will not or cannot agree to the new terms, the hypervisor may choose to search the registry for another SAN controller available that might be willing to agree to provide the storage. If this attempt fails, the hypervisor must reject the user's request.

Finally, we explore the scenario where the software that the user is testing has been developed as an SMR, for instance, an application of some sort. In this case, the application understands its own requirements (namely, a virtual server with a certain amount of memory, CPU, and storage), and is capable of deploying itself. The application, which is running in the user's test environment, sends a request for a new server to the hypervisor. From the point of view of the hypervisor, this scenario is indistinguishable from the previous case, where the request came from the user. After the virtual server is constructed, an agreement is put in place between the requestor (the application in this case) and the newly created virtual server, indicating that the server has agreed to provide the application with the base resource levels that were requested. If the application requires middleware SMRs to be installed in the new server, it uses the registry to find appropriate resource factories and negotiates with them to have this done. At this point, the application can install itself on the new virtual server and begin running there.

## CONCLUSION

We have described a general approach for moving from today's administrator-intensive IT systems management toward a self-managing IT infrastructure. To make this transition, the individual resources of the IT infrastructure must become SMRs. They must "understand" their own capabilities and be able to enter into formal agreements with other parties to provide these capabilities at specified service levels. They must also understand the lower-level resources that they require in order to provide these capabilities at the agreed-upon service levels. If asked to form an agreement to provide service to an additional client, the SMR must determine whether this can be done without adversely affecting its existing agreements, and what (if any) additional lower-level resources must be acquired.

SMRs and related technologies will allow both increased delegation of existing ITSM tasks from humans to the SMRs themselves and restructuring of ITSM activities through process restructuring and task replacement and elimination. In particular, the use of SMRs will convert many activities that currently require multiple planning, validation, and approval tasks into safe activities that can be done routinely, with a simpler task flow.

Of course, this is a very difficult goal to achieve, if it can be achieved at all. There are enormous challenges in encoding sufficient domain knowledge to produce a viable SMR. For relatively low-level resources, such as (virtual) servers, recasting them as SMRs is comparatively straightforward because their service agreements will be expressed in concrete terms that are easily understood: CPU share, memory occupancy, storage allocation, etc. However, for sophisticated middleware, such as relational database management systems (DBMSs), it is not yet at all clear how to express service requirements in a meaningful manner, or how the DBMS, acting as an SMR, can translate these requirements into its own need for CPU, memory, and I/O bandwidth. A request to agree to provide a client with a service level of, say, 150 SQL (Structured Query Language) requests per minute does not begin to tell the DBMS enough information: whether or not the DBMS can accept this agreement depends on the size of the database, its schema, and the characteristics of the SQL requests (e.g., selects on primary keys versus four-way joins). In such cases, it may be necessary to run benchmarks in order to characterize the workload that this request represents, to employ types of adaptive system models that have not yet been developed, and to advance in other ways our methods for automatically predicting and understanding the runtime behavior of complex systems. Much research is

needed to determine the sorts of benchmarking, modeling, and other techniques that will be useful and necessary for SMRs to be able to operate effectively. These will be far from the only research challenges facing the creation of truly self-managing systems; Reference 7 suggests a number of others.

Will SMRs operating in the ways that we suggest here truly revolutionize the way that IT systems are managed and give us a version of ITSM that approaches the ideal system described earlier? Today's prototypes and research efforts suggest both that great progress and improvement are possible, and that they will involve solving extremely difficult (and extremely interesting) problems.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of the United Kingdom Office of Government Commerce, Telemanagement Forum Corporation, or Sun Microsystems, Inc. in the United States, other countries, or both.

## CITED REFERENCES


1. IT Infrastructure Library (ITIL), Office of Government Commerce, http://www.itil.co.uk.
2. eTOM Overview, The TeleManagement Forum, http://www.tmforum.org/browse.aspx?catID=1648.
3. IBM Service Management, IBM Corporation, http://www.ibm.com/software/tivoli/governance/servicemanagement/index.html.
4. H. He, *What is Service-Oriented Architecture?* O'Reilly Media, Inc. (2003), http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html.
5. S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, "An Architectural Approach to Autonomic Computing," *Proceedings of the First International Conference on Autonomic Computing (ICAC '04)*, IEEE Computer Society Press, Washington, DC (2004), pp. 2–9.
6. K. Decker, K. Sycara, and M. Williamson, "Middle-Agents for the Internet," *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Morgan Kaufmann, San Francisco, CA (1997), pp. 578–583.
7. J. O. Kephart, "Research Challenges of Autonomic Computing," *Proceedings of the 27th international Conference on Software Engineering (ICSE '05)*, ACM Press, New York (2005), pp. 15–22.



*Accepted for publication February 14, 2007.*
*Published online June 28, 2007.*



***David M. Chess***
*IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (chess@us.ibm.com).* Mr. Chess is a research staff member in the Internet Infrastructure and Computing Utilities department at the Watson Research Center. He received an A.B. degree in philosophy from Princeton University in 1981 and an M.S. degree in computer science from Pace University. He joined IBM Research in 1981 and has worked on computer-mediated collaborative work, computer security and virus prevention, autonomic computing, and scalable systems management.

***James E. Hanson***
*IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (jehanson@us.ibm.com).* Dr. Hanson is a research staff member in the Internet Infrastructure and Computing Utilities department at the Watson Research Center. He received a Ph.D. degree in physics from the University of California at Berkeley. Since joining IBM Research in 1995, he has worked on emergent phenomena in networked systems, software agents and multi-agent systems, and autonomic computing.

***John A. Pershing, Jr.***
*IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (pershng@us.ibm.com).* Mr. Pershing is a research staff member in the Distributed Infrastructures department at the Watson Research Center. He received B.S. and M.S. degrees in computer science from the Massachusetts Institute of Technology. He has been involved in computer systems work for over 30 years, primarily in the areas of computer networking, large-scale clustering, availability, and systems management.

***Steve R. White***
*IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (srwhite@us.ibm.com).* Dr. White is a research staff member in the Internet Infrastructure and Computing Utilities department at the Watson Research Center. He is currently working on autonomic computing, exploring how to build computing systems that have billions of components, yet are still self-managing. Dr. White has published in the fields of condensed matter physics, optimization by simulated annealing, software protection, computer security, computer viruses, information economies, and autonomic computing and holds over two dozen patents in related fields. He is a member of the IBM Academy and has received IBM's highest technical awards for his work. Dr. White received a Ph.D. from the University of California at San Diego in theoretical physics in 1982. He held a post-doctoral fellowship at IBM Research before becoming a research staff member. ■