Management of the service-oriented-architecture life cycle

D. E. Cox
H. Kreger

Service-oriented architecture (SOA) development and deployment generally builds on a service view of the world in which a set of services are assembled and reused to quickly adapt to new business needs. This flexibility is seen by many IT organizations as the core value of SOA and has been driving some deep transformations in the way software is being built. Although SOA technology addresses many of the traditional problems of integrating disparate business processes and applications, deploying service-based applications introduces new aspects of the information technology (IT) environment that must be managed. These new aspects include developing and testing applications composed of operational services, deploying and provisioning distributed service-based applications across organizational boundaries in a secure, reliable, and repeatable manner, and tracking the business impact of services on the business processes that those services support. This paper describes the management capabilities needed to ensure that an SOA fulfills its promise of increasing integration and improving business adaptability.

INTRODUCTION

The traditional solution life cycle, including requirements analysis, modeling, and architectural design, followed by detailed design and construction in an IDE (integrated development environment) for deployment to a runtime environment, is evolving toward a more integrated process. As management technology expands in scope, management tasks and capabilities (and thus the enablement of a management solution) are introduced earlier in the solution life cycle, into the phases of modeling, development, and testing, not just in the runtime environment.

The solution life cycle can be broadly divided into the preproduction phase and the production phase, as shown in *Figure 1*. The steps of the preproduction phase are typically performed by architecture, design, and development organizations. The product of the preproduction phase is a packaged, tested set of solution artifacts (software components, installation programs, database and interface schema definitions, documentation, etc.). The steps of the production phase are typically performed by deployment and IT organizations. The product of the production phase is a running solution that is optimized for availability, IT resource usage and

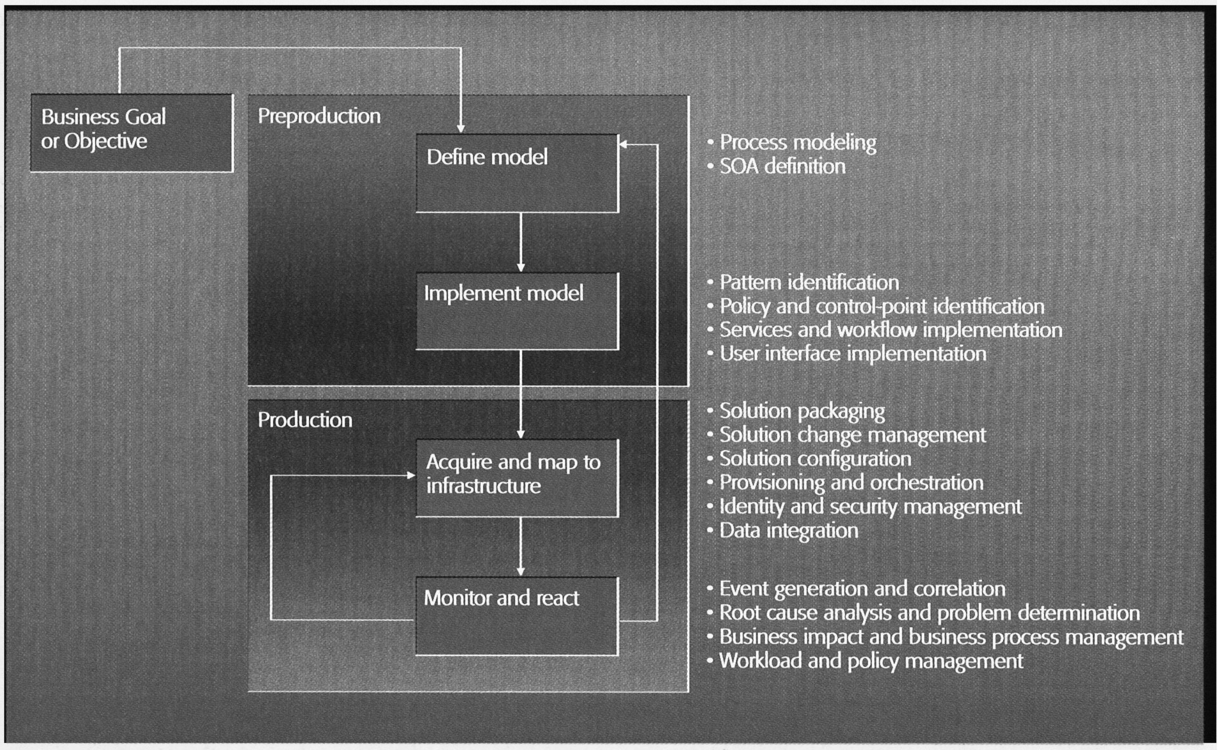

Business Goal or Objective
Preproduction
Define model
Implement model
Production
Acquire and map to infrastructure
Monitor and react
• Process modeling
• SOA definition
• Pattern identification
• Policy and control-point identification
• Services and workflow implementation
• User interface implementation
• Solution packaging
• Solution change management
• Solution configuration
• Provisioning and orchestration
• Identity and security management
• Data integration
• Event generation and correlation
• Root cause analysis and problem determination
• Business impact and business process management
• Workload and policy management

Figure 1
SOA solution life cycle

cost, and meeting business commitments. The production phase also includes steps to introduce maintenance and upgrades, as well as steps to phase out solution components (solution-life-cycle management).

The preproduction environment includes tools and processes for planning, modeling, development, function testing, and load testing. Management requirements in the preproduction environment center around developing, testing and debugging the solution by using tools and techniques traditionally used in the production environment, and preparing the solution for management in the production environment.

The production environment includes deployment and patches, upgrades and rollbacks, control operations, monitoring and optimization, security, and life-cycle management. Management in the production environment centers around the new management issues introduced by the nature of Web Services and service-oriented architecture (SOA). The requirements for the development, testing and debugging, and production environments are listed in *Table 1*.

Some requirements are similar between the different SOA life-cycle phases but may be implemented by using different technology or targeted toward different roles. For example, development requirements might be best implemented in a development environment such as Eclipse,[1] whereas the production requirements might be best implemented by using a standard management server/agent infrastructure. This paper describes the management systems and capabilities needed to manage the full life cycle of an SOA.

SOA CHARACTERISTICS

One of the key distinctions between an SOA and other distributed application architectures is the granularity and formality of the application components. There are many documented best practices about the recommended granularity of services in an SOA. Some are focused on performance, and others are focused on achieving the proper level of reuse and composability.[2,3] Composability is the ability to

Table 1 Requirements for an SOA solution

Development Requirements	Testing and Debugging Requirements	Production Requirements
Ability to visualize SOA components	Ability to visualize the services topology	Ability to deploy a services-based solution into a production environment
Ability to visualize the architecture by means of models that can also feed development, testing tools, and management	Ability to generate the proper instrumentation (static or dynamic) to make the solution and solution artifacts manageable (if it was not done during development)	Ability to dynamically provision an SOA architecture solution for automation and efficient use of capacity
Ability to generate the proper instrumentation (static or dynamic) to make the solution and solution artifacts manageable	Ability to generate test traffic for both unit testing and performance testing	Ability to manage multiple versions of a service to ensure compatibility and co-existence
Ability to capture relationship information	Ability to generate stubs for existing services to imitate production services based on recordings of production data flows	Ability to secure services interactions
Ability to advertise to a management system the management interfaces and capabilities that are available to manage the SOA components, such as operational status or configuration	Ability to specify a service version as a dependency and manage multiple versions of a service	Ability to provide a single definition of users and a single user sign-on for heterogeneous service architectures
Ability to define policy (such as security, quality of service, or privacy) and associate it with services, components, and users	Ability to monitor and report availability and performance of components of the solution during testing	Ability to visualize the services relationships (e.g., transaction paths, installation dependencies, runtime dependencies) to support tasks like visualization of processes and failure analysis for cause and impact
Ability to capture version compatibility	Ability to view events or faults generated by components of the solution or by the runtime infrastructure	Ability to visualize the state of the business processes, based on the state of the underlying services, to provide quick understanding of current business performance
	Ability to validate that the events produced in the preproduction environment can be correctly correlated to produce the KPIs (key performance indicators) and metrics to detect business situations as specified by the business analyst	Ability to monitor and report on the availability, security, and performance of the solution and solution artifacts
	Ability to visualize the KPIs and metrics in scorecards, reports, and dashboards	Ability to monitor and report the status of the business commitments (SLAs [Service Level Agreements]) for the solution's availability and performance
	Ability to visualize the business process impact based on simulated failures or problems to test this capability before the solution is deployed into production	Ability to isolate and diagnose a problem in the SOA environment
		Ability to view and take manual or automatic action on events or faults generated by the production environment
		Ability to feed production, trace, and debug data to test models and development tools in real time

build new things from existing, reusable interfaces and components, either independently or in combination, to satisfy system-specific and user requirements.

The following critical aspects of an SOA affect management. Services are often designed for one or a small number of closely related functions. They have well-defined interfaces and relationships. SOAs use standards-based communications and are more closely aligned with business processes.

In the following discussion, we present an example that illustrates the differences between a distributed application architecture and an SOA from a management perspective. First, we describe the distributed

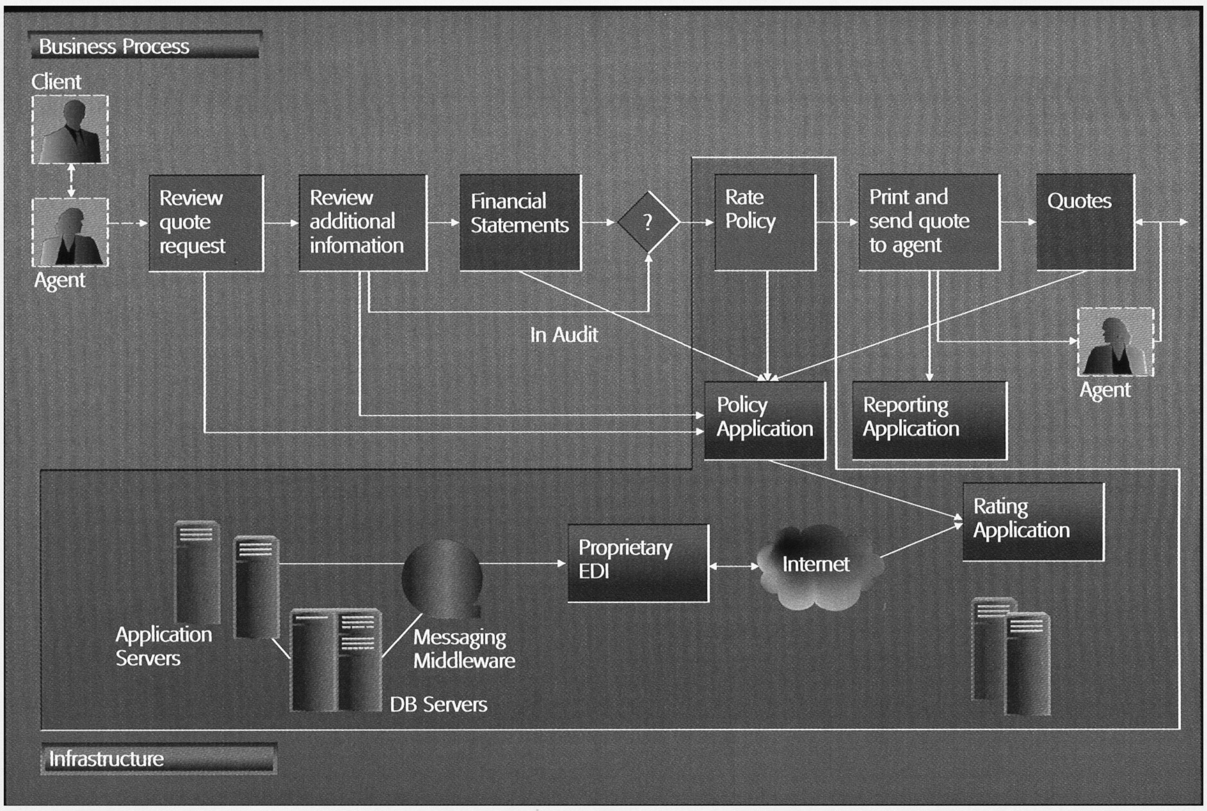


Figure 2
Example of business process mapped to distributed application architecture

application architecture shown in Figure 2; next, we describe the SOA shown in Figure 3.

Figure 2 shows a typical business process (involving an insurance quote) and how it maps to distributed application artifacts. The distributed solution is composed of two applications running in the local IT environment, the *policy* application and the *reporting* application. The insurance-quote business process also requires use of a third-party application, the *rating* application, which runs in another company's IT environment.

The highlighted area in Figure 2 shows how a business process step, "rate policy," maps to the application artifacts and the IT infrastructure. The rate-policy step invokes a task in the policy application, which is a distributed application that runs on multiple application servers and depends on certain database tables hosted by a relational database management system (RDBMS). In order to complete the rate-policy task, the policy application uses Electronic Data Interchange (EDI)[4,5] to invoke a request to an external rating application. The relationship mapping is very vague and difficult to quantify. This is because the policy application supports many business processes, and there is no way to determine which business processes use the policy application. The application supports many tasks related to insurance-policy manipulation, and there is no way to discover the list of tasks. The application exposes several capabilities through a user interface only, and there is no way to describe or discover (make known) the interfaces to the capabilities.

Further difficulties are due to the fact that the tasks provided by the policy application are lumped together into a single management view of status and availability, so that there is no management interface to determine the status or performance of individual tasks. The policy application is distributed among several application servers, and there is no way to determine which of those application servers are necessary to the processing of a specific

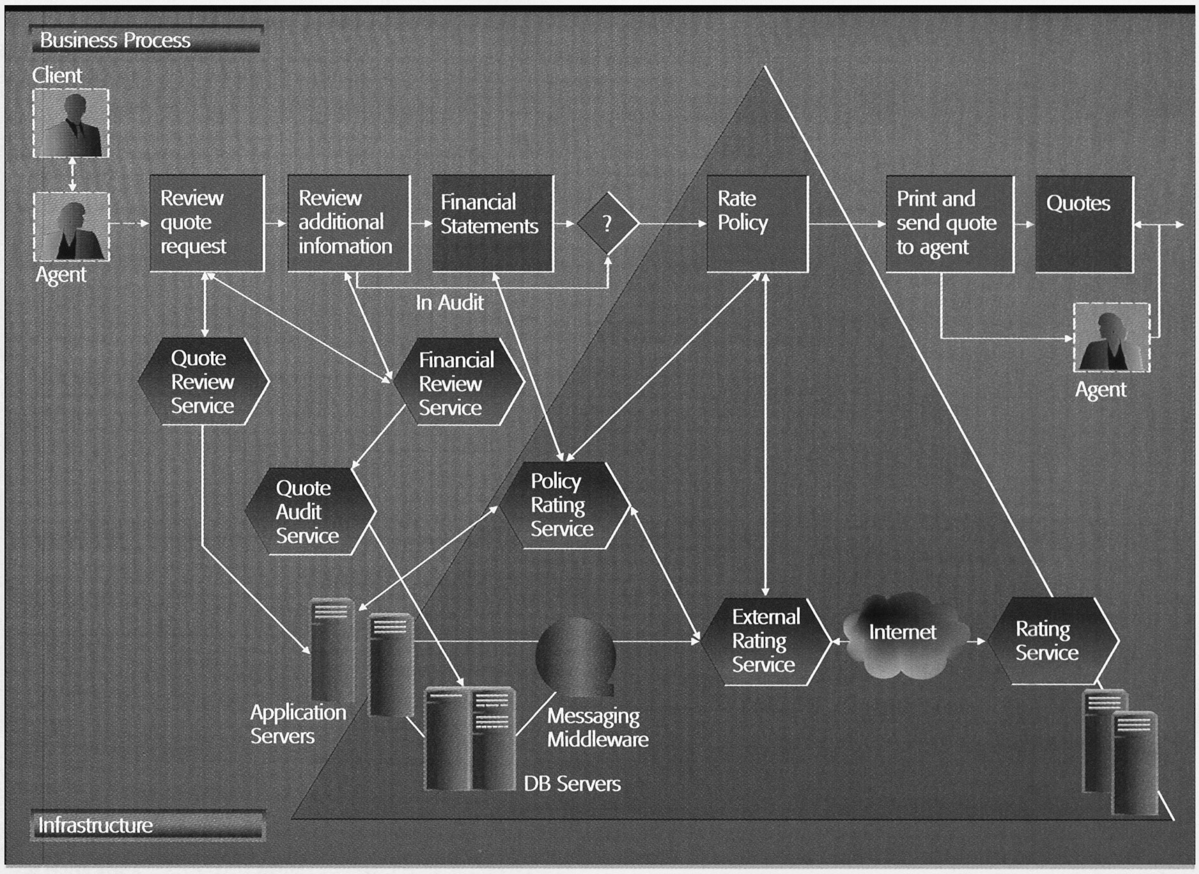


Figure 3
Example of SOA business process

task such as rate policy. The policy application has dependencies on a RDBMS and a set of database tables, but it is impossible to determine which tasks or capabilities of the application depend on the database system or specific database tables.

The instances and locations of the policy application components are not defined or recorded in a standard, distributed repository. The EDI interface to the external application does not have any management capabilities. The dependency relationship between the policy application and the external rating application is not defined in a way that a management tool can discover.

The management of a distributed application is limited in many ways due to the ambiguous relationships described previously. The result is that IT operators tend to learn typical relationships and failure patterns by trial and error. The process of managing such a system is consequently error-prone, manually intensive, and very sensitive to changes in the system.

As an example of a process that avoids these difficulties, ***Figure 3*** shows the same business process mapped to an SOA. The services in this architecture implement specific and granular tasks and have well-defined WSDL (Web Services Description Language)[6] interfaces. The relationships between a step in the business process, the services, and the IT infrastructure as shown in the highlighted area of Figure 3. Due to the nature of services,[7] the relationships in this figure are specific and easily discovered. This is because the services have standards-based WSDL interfaces, which can be choreographed by a flow engine such as Business Process Execution Language (BPEL);[8] the service interfaces can be described and advertised in a distributed standards-based repository (i.e., Univer-

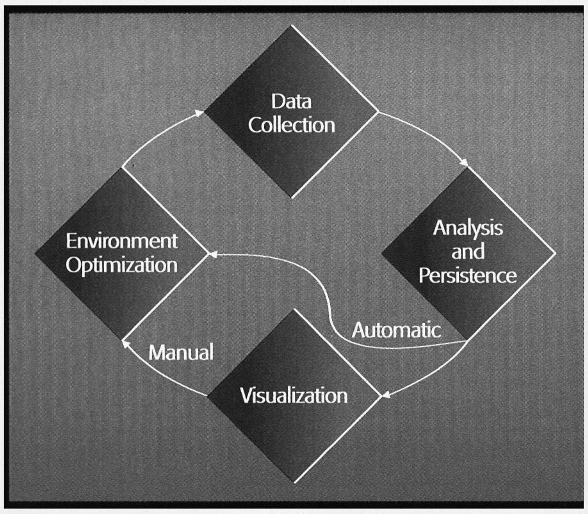


Figure 4
SOA management system interactions

sal Description, Discovery, and Integration [UDDI][9,10]); the services run on specific application servers (known to the management system, but not necessarily known to the client application); and they have specific and well-defined dependencies on other services.

Management interfaces (Web Services Distributed Management [WSDM][11]) allow discovery of dependencies on other types of resources, such as IT resources, and management capabilities of the remote rating service can be exposed through WSDM. The service instances are easily discovered through WSDM management interfaces.

MANAGEMENT SYSTEM INTERACTIONS

This section describes the high-level interaction between different aspects of the management system. It shows how some of the tasks and capabilities introduced earlier relate to the four major components of a management system, as shown in *Figure 4*. Although many of these tasks and capabilities apply equally to non-SOA environments, this section highlights key points that are especially important for SOA environments.

Data collection

The first task that a management system performs is the collection of data about the managed environment. For traditional IT management, the management system would typically collect data about the performance and availability metrics for servers, operating systems, and applications. For management of an SOA, the management system must recognize services in the SOA first as class objects. The management system must also collect data at the proper level of granularity; for example, to manage an SLA (service-level agreement) for a specific user, data must be collected to calculate the average response time by the user. Similarly, the management system must collect or discover data about the relationships between a service and the business processes that use the service, in order to be able to measure business impact.

The specific management data collected for an SOA includes the identity of services and related infrastructure, configuration information, metrics, status, activity trace entries, and topology diagrams depicting the relationships among services. This information is used in many different management scenarios, including setting runtime performance thresholds, customer-focused SLA monitoring, and business impact analysis with prediction of impending failures.

Management systems get data directly from the data collection service or harvest it from the environment of the service. Services can provide data through (1) asynchronous events, (2) "heartbeats" sent to the management system (i.e., regular messages sent periodically when the resource is functioning properly), (3) metric and configuration properties that can be queried, or (4) audit information and errors saved in a log accessible by the management system. Data can be harvested from the services environment by interacting with similar management data providers for service containers, service runtimes, and service buses. For example, a management system would poll a Web service that is part of an SOA for operation invocation counts, but it would poll the application server hosting the Web service for total CPU time and memory usage.

Data can be collected by using a number of different interaction patterns: polling of values for the resource on a periodic basis, using asynchronous events when abnormal situations occur, using heartbeat messages when the resource is healthy, and batch searching of new log messages. In addition to these collection patterns, management systems can use intermediaries, or management agents, to provide communication to, or a proxy for, the service instrumentation. Management systems can interact directly with a Web service that offers a management interface, or they may interact with a printing service through a Web-service bridge that communicates with the printer by means of SNMP (Simple Network Management Protocol).[12] The management agents may also provide additional services for the management systems to support scalability and additional management protocols. As an illustration, a management system might have metric analysis services run locally on the systems where application servers are deployed, reducing networked polling traffic.

In addition to providing instrumentation for the provision of data to management systems, services should be instrumented for control operations to allow automated optimization and control. The kinds of operations and how they are used is discussed in more detail later in this section.

As mentioned, management systems may harvest data from the service environment. The service environment consists of the network systems, the service bus, application servers, brokers, and other middleware components that allow the SOA to function. Management systems might regularly scan systems and service environments for new services, new relationships indicating a change in dependencies or processes, or new log messages. Some data is only collected when requested by administrators. For example, collecting, normalizing, and correlating logged information from services, service environments, and service buses can provide data used for sophisticated trend analysis, resource requirement prediction, and SLA analysis.

Analysis and persistence

After data is collected by the management system, it is analyzed and made persistent (maintained across session boundaries in a database system). The following section describes some basic management abilities (such as calculating status) that are achieved by analyzing data. Data analysis enables the provision of a wide range of management capabilities, including complex functions such as problem isolation, business impact analysis, and management of SLAs.

Typical analysis generates service performance statistics, including request counters, transaction performance, processing times, and failure rates, both for the service as a whole and for each user's perspective on his or her experience with the service. The management system compares the analyzed data with expected or committed values. If the SOA environment is not performing as expected, the management system uses a policy to determine corrective actions, such as alerting an operator or invoking automatic corrective processes.

Relationship discovery is a key goal in the analysis because SOA environment models are aptly represented as a set of relationships on the same plane (or layer) and between planes. The planes analyzed for relationships include the service plane, the service environment plane, the IT infrastructure plane, the enterprise application plane, and the cross-business application plane. Three of these planes are described in ***Figure 5***, and their interactions are shown in ***Figure 6***. Understanding relationships between services is as crucial as understanding relationships between planes. Spanning service-to-business planes is a requirement unique to SOA management as compared with typical resource and application management. Services in these cases can include a combination of programmatic or human services. Relationships are discovered through analysis of relationship registries, relationship information provided by resources, or deployment relationships. Relationships are used to augment other management data to facilitate problem determination and business impact analysis.

Service status is determined by analyzing metrics, state models, and failure messages. In addition, the status of other services and resources that are in dependency relationships should be considered. The same is done for the service environment and service bus. Service status must also be understood from each service user's experience. Detecting an inappropriate status for a service may depend on determining and predicting its ability to continue to function and satisfy important SLAs. For example, if an application server running a service fails, the management system must know if a backup service exists and if the failover mechanism is automatic or not. If there is a backup service, the management system may change the configuration or policy to insert the backup service into the SOA environment; if no backup service is defined, the management system may alert an operator that no automatic corrective action is available and that the failed service will impact certain users or business processes.

Problem detection and determination are two steps in one of the oldest management processes. Man-

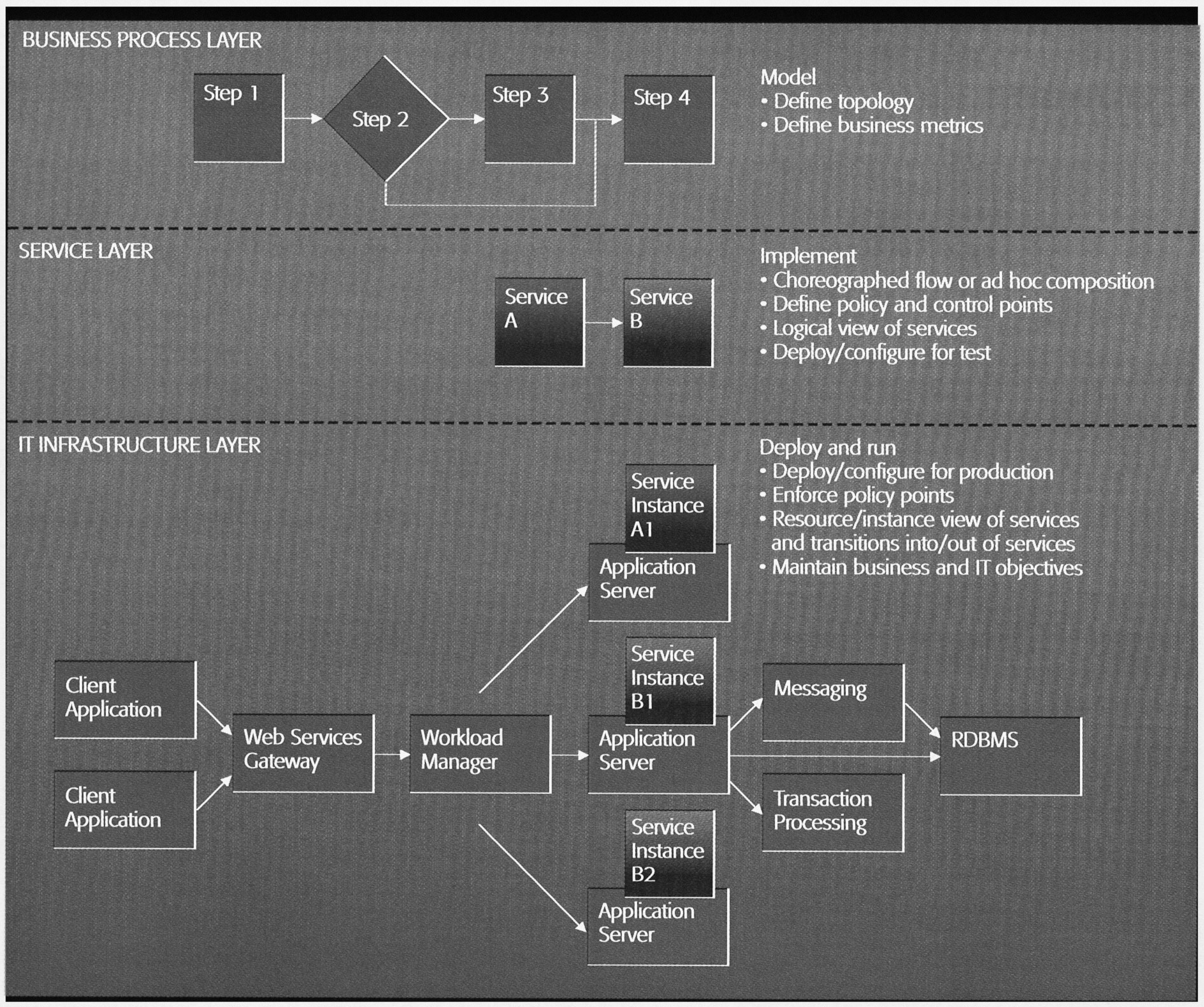


Figure 5
SOA planes

agement systems must be able to detect a service failure or degradation in a timely manner. Detection is supported by receiving analysis of failure events from or about services, polling metric information and detecting threshold exceptions, monitoring for operational status changes, or discovering invalid configurations or relationships. For SOAs, this detection and analysis must, at times, be done across service, application, and business planes. Once the problem is detected, management systems use analysis of data and dependency relationships to determine what the root cause of the failure is. Sometimes management systems can only narrow the root cause down to a few likely options for the operations staff to investigate further, but this programmatic analysis is much faster and more accurate for most common failures, relieving operations staff of tedious and error-prone work and leaving them more time to resolve more complex problems.

Business impact detection and determination are two final steps in the management process. Like problem detection and determination, they rely on receiving or retrieving indicators of business system problems. In contrast with typical resources, relationship information is crucial for SOAs for traversing business, service, and IT resource planes to find the source of the business system problem. Business impact understanding is used to prioritize the resolution of multiple problems, either with automation or by operators. This understanding may also enable a management system to detect that a business commitment or SLA is about to be violated, allowing corrective action to be taken before it is too late.

Service-level monitoring and enforcement is slightly different from resource performance and problem determination analysis in that instead of focusing on the resource, service, or system as a whole, the focus is on a particular client's experience with a service. Raw metrics and events must be sorted and decomposed, based on which clients were interacting with the service. Thus, instead of the number of transactions processed by a service, the number of transactions per client per service is tracked. Using the client-specific data and events along with SLAs, management systems detect exceptions to the agreements. This can be done in real time by the service environment, in which case imminent agreement exceptions should be detected so that real-time corrections can be made to ensure that actual exceptions do not occur. Service-level exception detection can also be done on a more coarse time basis, such as daily, weekly or monthly, using persistent data.

Raw and analyzed data is made persistent in a normalized format by using the management system's common data model. The process of converting raw information into the common data model requires mapping semantic information from many different management models into a common set of fields. Performing this process once rather than each time analysis is performed reduces inconsistent data mapping and skewed results. Maintaining the information in a normalized data model enables the efficient processing of high volumes of data and provides analysis results of higher fidelity. The normalized data model also allows easier correlation of data collected from different sources. This normalized historical information is used in future management tasks, such as historical trend analysis of service performance, resource usage, usage analysis for billing, and building and maintaining accurate relationships. The normalized historical data may also be used for data-mining techniques, such as identifying common conditions leading up to a failure, so that processes can be created to detect the condition and prevent the failure.

Visualization

The visualization capabilities of the SOA management system allow users to view and interact with

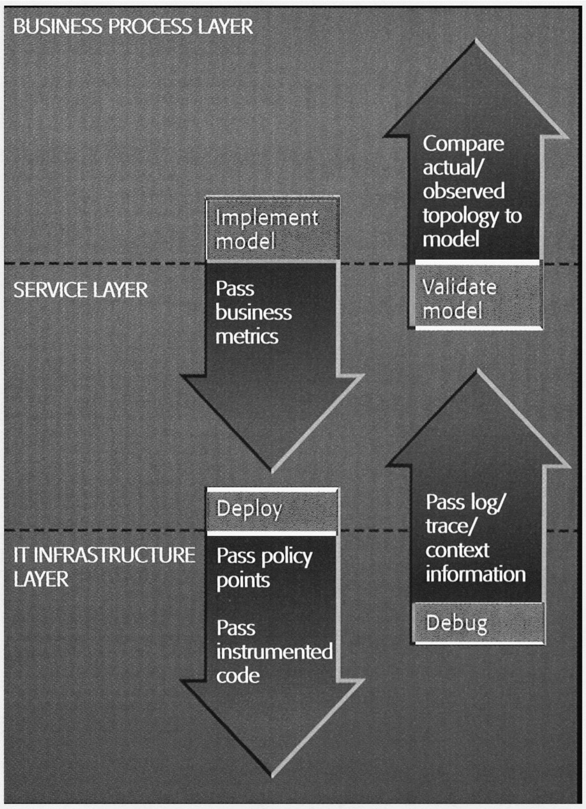


Figure 6
Interaction between SOA planes

the services, SOA topology, and SOA environment. The visualization capabilities span the SOA management life cycle from architecture and development, through deployment and configuration, to monitoring and optimization, and finally to maintenance and end of life. Visualization as a tool for SOA management is especially important because of the infrastructure, service, and business planes it must span and relate. Visualization tools can thus be complicated to maintain.

Visualization in the preproduction life-cycle phases should be integrated in an IDE tooling environment such as Eclipse.[1] The management system should provide data for preproduction visualization that describes the observed behavior of services in the SOA. The observed views can be compared to the expected or intended topology to detect problems. The observed data from the management system can also be used as context to help resolve problems. For example, log files or actual message content can be passed from the management system to the visualization tool in the IDE environment to give the support person more information to diagnose a problem. See the paper on Web Services Navigator by De Pauw,[13] elsewhere in this issue, for details on SOA management capabilities that can be provided in the preproduction environment.

Preproduction tooling should also provide user interface capabilities to define control points for the management system across the infrastructure, service, and business planes. The visualization tool should allow the architect or designer to define points in the service flow where authorization, prioritization, data integrity checking, business metric analysis, and other tasks can be performed. The runtime management system can then use the control points to monitor the SOA and can modify policy at the control points to control the SOA. The management control points may be a formal, designed part of the solution[14] (most likely as intermediary services) or may be implemented in the infrastructure and thus be transparent to the services.

The production management system should have visualization capabilities that allow monitoring and control of the SOA. The management system continuously monitors the system for indications of problems, such as events being generated or thresholds being exceeded. When problems in the SOA are detected, the management system shows visually that a new alert needs attention and also shows the affected systems and business processes in a table or topology view.

The SOA management aspects of the management system are used to diagnose and correct the problem. The SOA visualization tool includes specialized topology and message information that helps identify problems in service-to-service interaction. The SOA visualization tool also helps the operator understand where the control points are in the service topology and what policy and configuration parameters are available to control the environment.

The management system also provides user interfaces to take action on the SOA and the SOA environment. The user interfaces allow the user to control the services and the runtime infrastructure by providing commands such as “configure,” “start,” “stop,” and “restart.” The user interfaces also allow the user to control the environment by changing policy, which may result in configuration changes or commands being issued.

Environment optimization

When business-, resource-, or service-level problems are detected through events or monitoring, automated corrections can be made to the SOA environment, services, or Web services. Again, the cross-plane impact of the policies and optimization that is so important for keeping resource, service, and business goals optimized is unique to SOA management. Automated reactions occur based on policies defined in the management system. These policies can be pushed down into the resource level and complied with locally by the services. Automated corrections can manifest themselves in several ways, such as policy changes, automated recovery, load balancing, and dynamic provisioning.

Policy or configuration changes can be made by management systems to influence the ongoing behavior of the resource. For example, thread pools can be made larger, or a policy to allocate new resources, an expensive operation, can be changed to occur less frequently. In addition, policies can be used to control the availability of the management information itself; that is, metrics collection, log monitoring, and message monitoring, where the status of transactions is determined and the amount of information logged from a message is tuned according to the contents of the message, its business impact, and policies.

Automated recovery occurs when a management system automatically restores a resource that is down, has failed, or is degraded to fully operational status, functioning within the expectations of policies and SLAs.

Load balancing allocates service requests across a set, or cluster, of services. This usually maximizes system resource usage, provides better performance for services, and provides better availability, because it prevents the failure of one service from causing new service requests to fail. Load balancing can also support virtualization of resources, where one resource or service can appear to belong to many clients, without the clients being aware that they are sharing the service.

Dynamic provisioning corrects situations where there are insufficient resources available to maintain

SLAs. New resources can be brought online to support the services. In addition, services are provisioned and instantiated in times of high demand and then destroyed, freeing resources when they are no longer needed. Dynamic provisioning of a new instance of a service can vary widely, from adding an existing but idle instance of the service to a pool to requesting a new machine from a pool and installing the operating system, middleware, and service software. For Web Services SOAs, routing service invocations to a convenient service instance is transparent to the client or business process because Web Services support transparent address redirection.

Automated recovery and dynamic provisioning are usually sophisticated multistep conditional tasks with compensation for failure and the capability of rolling back activities. BPEL documents can effectively describe these conditions as business processes, also called *management processes*. Management processes, like those defined by the ITIL (Information Technology Infrastructure Library)[15] standards, can be built by using the same high-quality tools and workflow engines that drive production. Applying SOA concepts like BPEL to the management processes themselves is the primary enabler behind IT process automation. This not only makes building and customizing management systems cheaper and easier for management system vendors; it also makes it easier for businesses because they are reusing existing skills in business process development and execution. This builds an unprecedented synergy between business and management processes.

OVERVIEW OF THE MANAGEMENT SYSTEM ARCHITECTURE

This section describes the architectural layers of the management system that is used to manage an SOA. It provides a more detailed architectural, layered view of the four major capabilities described in the previous section (data collection, analysis and persistence, visualization, and environment optimization). The management-system architecture is shown in *Figure 7*.

The layers of the management-system architecture work together to provide the four major capabilities. The *data collection* capability of the management system is manifested in the management instrumentation provided by the managed resource layer, and in the management-system agent layer, which interacts with the instrumentation. The *analysis and persistence* capability is provided on a local, single-machine basis by the management agent and on a system-wide basis by the management server layer. The *visualization* capability is provided by the user interface layer of the management system architecture, which works with the raw and analyzed data stored by the management server. The *environment optimization* capability is provided by the user interface layer (for manual manipulation of the environment) and by automation capabilities in the management server layer. Both manual and automatic environment optimization can be put into effect directly by issuing commands to the management system or the managed resources in the SOA environment, or indirectly by changing policy in the management system or the managed SOA environment.

MANAGEMENT SYSTEM COMPONENT DESCRIPTIONS

In general, management architectures are very similar in that they have managed resources that interact through an agent with a management server, which has a user interface for operators. The management system topology usually consists of a *management agent* on each machine that contains managed resources or systems. The *management server* is a distributed application that runs on one or more systems in a small number of central locations. The *management user interface controller* is also a distributed application that runs in a central location. The users of the management system may access the user interface locally through a browser or installed client, or remotely over the Internet with a browser. Secure access to management operations and secure communications among the components of the management system are clearly important parts of the management system.

This section describes in more detail the components of the management architecture, layer by layer. In this architecture, the service in an SOA is a managed resource, as are the middleware and hardware components that provide the SOA runtime environment.

Managed resources

The managed resource layer provides instrumentation that allows a management system to interact

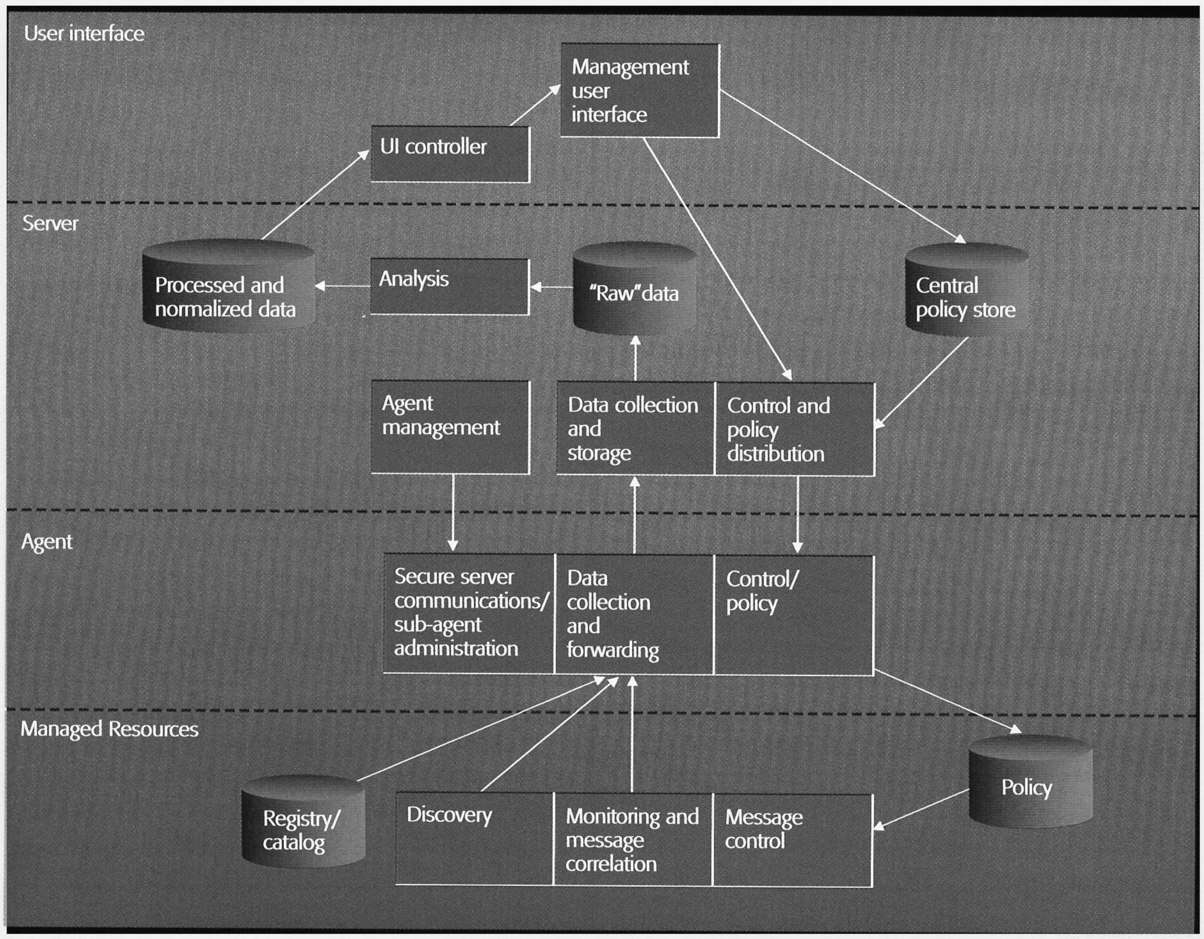

User interface
Management user interface
UI controller
Server
Processed and normalized data
Analysis
"Raw"data
Central policy store
Agent management
Data collection and storage
Control and policy distribution
Agent
Secure server communications/ sub-agent administration
Data collection and forwarding
Control/ policy
Managed Resources
Policy
Registry/ catalog
Discovery
Monitoring and message correlation
Message control

Figure 7
Management system architecture

with its environment. The instrumentation must provide sufficient information to enable some of the management capabilities described in the section "Overview of the management system architecture." Typically, this includes creating events to signal health or failure situations and providing properties and metrics that indicate the current health, performance, and configuration of the system. The instrumentation must also allow for polling, configuration, and control of the environment, often by direct operations as well as by using a set of management policies.

When a managed resource becomes available, a management server should perform discovery on several levels and integrate the resource into the managed environment. The management server discovers the existence of the resource and its manageability capabilities. To support the discovery of its existence, the resource should register or broadcast its presence to the data collection component so that the management system can begin to bring this new resource into the managed environment. To support the discovery of its capabilities, the resource should advertise a description of the management capabilities it supports. If the data collector processes that description and understands this type of resource, it will add the resource to its managed-resources configuration. If this resource is not one that the data collector has been configured to collect from, the data collector notifies operators and administrators who use tools in the user interface to configure data collection, monitoring, and policy for the resource. After the data collector is aware of and able to interact with the managed resource, the system should use policy to control which system messages should be monitored, filtered, correlated, and sent to the data collector.

Instrumentation should be provided by using a combination of standards including CIM (Common Information Model)[16] models, JMX** (Java Management Extensions),[17] and WSDM. Instrumentation should be provided for the services themselves in an SOA, as well as for the infrastructure components that host the services and handle the messages.

CIM provides the basic, standard, extensible models for the management of resources that are used to create the service environment and services. The creators of instrumentation for services should extend these models with management information necessary to manage a particular service or resource. For Web Services, WSDM provides the basic management model, and service developers should extend the basic management capabilities with any capabilities specific to the service being developed. For example, CIM provides the basic model for a service that includes the fields `Name` and `Started Boolean` and operations for the methods `startService` and `stopService`. It can be extended to provide additional information for management of a rating service that might include the methods `updateRatingPolicy` and `updateRatingTables`.

WSDM defines how to represent management interfaces by using Web Services. This standard management-interface definition creates an integration layer between the different management protocols used to instrument service-environment resources and management systems. WSDM defines a basic set of manageability capabilities that can be composed, like selecting from a "buffet," to express the capability of the management instrumentation. WSDM defines how to express resource identity, metrics, configuration, and relationships by using Web Services. WSDM depends on the following standards; WS-I (Web Services – Interoperability)[18] for the basic profile, WS-Resource Framework[19] for property collections, and WS-Notification[20] for management-event transport. WSDM also defines a standard management-event format to improve interoperability and correlation.

The customized models, like the CIM example described earlier, provide the actual properties and operations for a WSDM-manageable Web service interface for the service. Building on the CIM example described earlier, the properties of the rating-service-manageability Web service would include `Name`, `Started`, `CurrentRatingPolicy`, and `CurrentRatingTable`. WSDL operations would include `StartService`, `StopService`, `UpdateRatingPolicy`, and `UpdateRatingTables`.

Tools supporting the development and testing phases of the SOA and service life cycle can facilitate management instrumentation during development. Development tools provide "wizards" that are triggered during development of manageable components, services, and business processes. These wizards should display log and error messages as candidates for management events, allowing developers to tag properties in components such as identity, metrics, and configuration-management data. These wizards can generate the instrumentation by using JMX MBeans (management beans) and Web Services to reflect this management information as a WSDM-manageable resource.

Agent

The use of agents increases the scalability of the management system by reducing the volume of data communications and offloading some management work from the machine running the management server to the machine running the managed resources. This is accomplished by reducing the number of communications paths from the management server to the individual managed resources on a target system, running some management server tasks (i.e., discovery, monitoring, and setting thresholds) locally, and by filtering and aggregating data before forwarding it to a management server. Agents can also be used as nodes in a distributed or federated data-storage topology. The advantages of an agent architecture layer are more apparent when there are a large number of remote managed resources in the environment, such as in a bank branch topology.

The agent works in conjunction with the management system to provide the data collection capability for the managed resource. The agent performs the task of invoking the instrumentation in the managed resource and then forwards relevant information to the management server. The agent may perform some filtering and correlation of data as well. Not only do policies from the management server dictate what data is collected from which resources and how often; they also specify filters for discarding duplicate or irrelevant data and rules for performing correlation.

The agent provides operational control of the managed resource, for example, sending `startService` operations from management servers to a rating service. The agent also acts as a policy distribution point, sending validated policies from the management server to the policy-capable managed resources. When the managed resources are not aware of policy, the agent acts as a policy enforcement point for the resource, translating management and business policies into operations or configuration changes in the managed resource.

In addition to enabling scalable management, agents themselves must be manageable, allowing fixes, upgrades, and policy changes just like other software. The subagent `admin` function provides for agent management as well as a secure communications pipe to the management server.

Server

The server has three distinct tasks. The first task supports traditional management of resources, collecting raw information from numerous types of resources and environments by using instrumentation, logs, and other sensors. The server filters, correlates, and saves management data in a standard format as dictated by a set of policies. This information is fed into the analysis step discussed previously. This analysis is used to maintain a federated model representing the overall managed environment that is the basis for visualization of the system by the user interface.

The second task of a management server is to send command and control information to managed resources through the management agent. Command and control messages may be initiated by an operator or by automation through policies, scripting, or management processes. The management system must ensure that all command and control messages, as well as access to collected data, are controlled through secure authorization mechanisms. Finally the policies themselves must be maintained, either by operations, higher-level policies, or automatic reactions of the management system to service and resource changes in the system. One of the operations includes distributing the policies to agents and resources that perform policy enforcement. Additionally, resources and agents may request a policy relevant to some condition in the system that needs to be automatically addressed. The command and control features of the management server manage operations as well as changes in policies and configuration of resources to support optimizing the system in honoring SLAs and high-level business policies.

The third, and often overlooked, task is the management of the agents themselves as discussed in the section "Agent." The agent management task includes keeping a list of the agents and the managed resources that they have access to, managing the agent software, ensuring that the agent is available, and managing the policy that controls the agent's behavior.

User interface

The user interface gives operators a window into the system and tools for failure analysis and recovery. The interface allows an operator to efficiently visualize the topology and operational status of the business systems and SOA, including individual services and underlying resources.

This visualization is a cached reflection of the system model created by the analysis step, which is fed from discovery, status, and performance events. Resource existence and relationships and their status, which may have to be aggregated from the status of other related or dependent resources, must be synchronized with the environment model.

The user interface must also support policy configuration to guide the overall management of the environment in a consistent way. The policy management interface should make it easy to develop high-level rules that create resource-level policies. The business and management policy can also be expressed as a set of BPEL processes, for example, scripts identifying a series of conditions, tasks, and compensations for performing sophisticated, tailored management scenarios and recoveries. These BPEL processes drive operations onto agents and resources. These policies and processes are stored in a central repository. The new policies may be pushed immediately to agents and resources or kept locally until conditions are met to activate them in the management server.

In addition, the user interface should provide operators a tool to assist with problem determination for the business system or SOA as well as tools or connections to tools for correcting problems. The problem determination tools use resource models and historical analysis data. Using a common management-event format, like that in WSDM, is key to enabling consistent correlation of events from resources and their environments.

USAGE SCENARIO

This section provides examples at a high level of how a preproduction task and a production task would be accomplished using the SOA management capabilities described earlier in this paper. This section shows the interaction between the administrator or user of the management system, the major management system components, and the managed SOA environment.

Problem determination scenario

A solution based on an SOA was developed and deployed as a proof of concept running in the operational environment but has not yet been used in production. The appropriate management instrumentation and control points were designed into the system using the preproduction techniques described earlier in this paper. During the solution testing, several configuration parameters of the application were adjusted to improve performance. Suddenly, the solution began to experience failures. All of the transactions sent to the Web Services interface of the solution were failing.

The IT operator was the first to notice the problem. Alerts appeared on the event console indicating that the "number of failed requests" threshold had been crossed. The IT operator began to diagnose the problem by launching the service topology display; the services were available, and the systems running the services were all operational and performing according to expectations. The IT operator deduced that a logical or business-level problem existed and passed control of the problem to the level-3 support team in Development.

The level-3 support person brought up the service topology display and viewed the recent transactions. All transactions since 1:02 p.m. on that day had failed. Selecting one of the transactions, the level-3 support person viewed the message content. The transaction failure was due to a security negotiation failure. The support person brought up the code for the service and saw that it generated a fault message if the incoming security request had a different policy from the policy for the service being invoked. The support person brought up the policy configuration tool and looked at the configuration of the client and the service. The client was configured to use a nonsecure connection; the service was configured to require a secure connection. The support person brought up the audit log for the policy and saw that at 1:01 p.m. an operator changed the security policy on the client to "no security," presumably to improve performance.

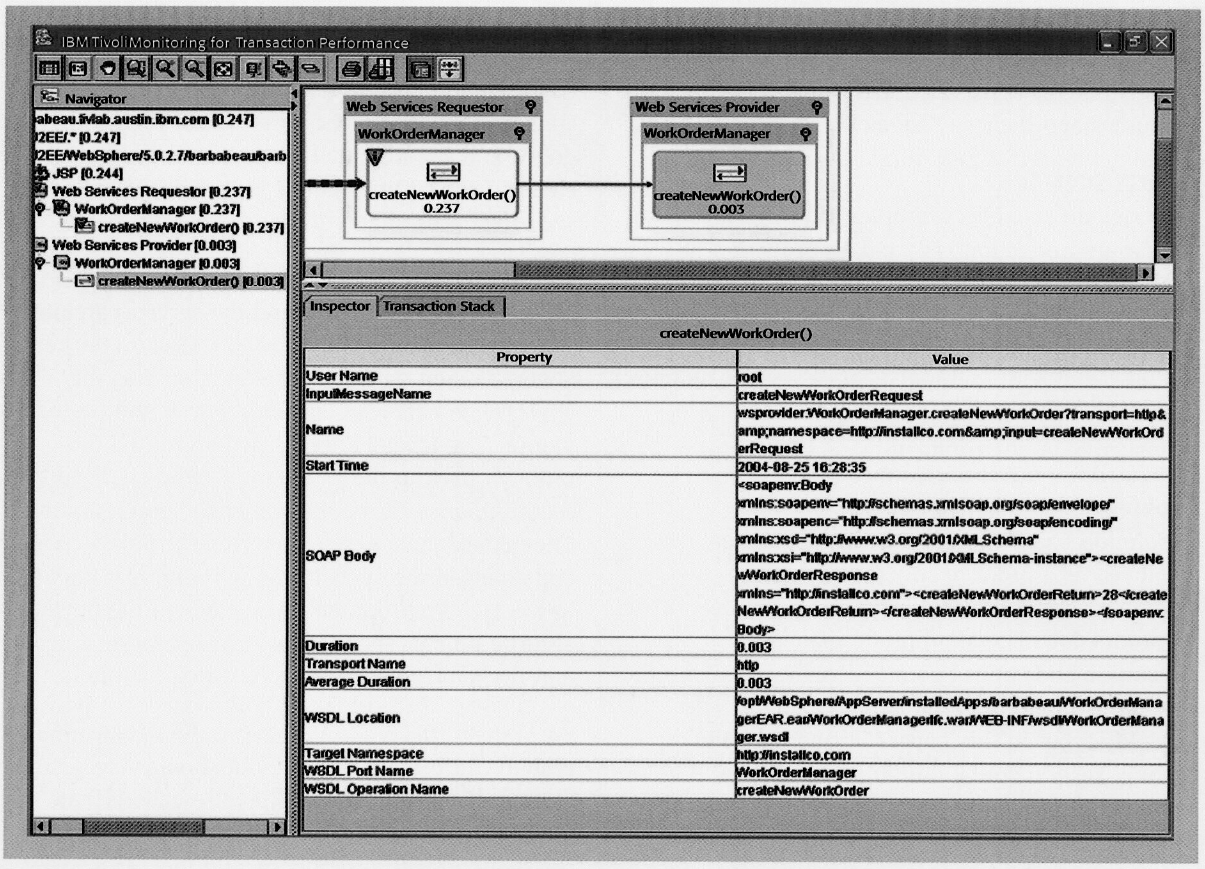
The level-3 support person initiated a conference call between the IT team and the services architect. The attendees agreed that an SSL (Secure Sockets Layer) session with 56-bit encryption was the appropriate balance between security and performance. The level-3 support person routed the problem back to the IT operator, who changed the security policy to the recommended configuration. The management system detected the policy change and changed the specific configuration parameters in the SOA environment to implement the new security policy. The SOA became operational again, and the management system cleared the alerts.

To prevent this type of problem, the management system can provide a hierarchical policy system. A high-level policy can be defined to ensure that lower-level policies are consistent.[21] For example, the scenario just described would include a high-level policy that all Web service traffic should have at least 56-bit SSL encryption. Any attempts to configure a specific service for a different encryption would be rejected before the system could be configured correctly.

Business impact scenario

In this scenario, a company had developed an SOA-based solution to integrate internal applications within its enterprise. The SOA solution was deployed in the data center and was being used for production work. The data center experienced a problem with power distribution, and five machines running application servers went down. Many of the services in the SOA implementation were running on the failed application servers. The IT team had configured the application servers to fail over to a backup machine, and all five application servers did so. The management system configured the traffic to route all failed services to the backup instances running on the backup machine.

The IT operations team notified the business analyst of the failure and recovery actions. The business analyst looked at the recent trends for the most important business processes. Because the solution


IBM TivoliMonitoring for Transaction Performance
Navigator
abeau.tivlab.austin.ibm.com [0.247]
J2EE/.* [0.247]
J2EE/WebSphere/5.0.2.7/barbabeau/barb
JSP [0.244]
Web Services Requestor [0.237]
WorkOrderManager [0.237]
createNewWorkOrder() [0.237]
Web Services Provider [0.003]
WorkOrderManager [0.003]
createNewWorkOrder() [0.003]
Web Services Requestor
WorkOrderManager
createNewWorkOrder() 0.237
Web Services Provider
WorkOrderManager
createNewWorkOrder() 0.003
Inspector
Transaction Stack

createNewWorkOrder()

Property	Value
User Name	root
InputMessageName	createNewWorkOrderRequest
Name	wsprovider:WorkOrderManager.createNewWorkOrder?transport=http&amp;namespace=http://installco.com&amp;input=createNewWorkOrderRequest
Start Time	2004-08-25 16:28:35
SOAP Body	<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><createNewWorkOrderResponse xmlns="http://installco.com"><createNewWorkOrderReturn>28</createNewWorkOrderReturn></createNewWorkOrderResponse></soapenv:Body>
Duration	0.003
Transport Name	http
Average Duration	0.003
WSDL Location	/opt/WebSphere/AppServer/installedApps/barbabeau/WorkOrderManagerEAR.ear/WorkOrderManagerIfc.war/WEB-INF/wsdl/WorkOrderManager.wsdl
Target Namespace	http://installco.com
WSDL Port Name	WorkOrderManager
WSDL Operation Name	createNewWorkOrder

Figure 8
Diagnosis using IBM Tivoli Monitor for Transaction Performance

architecture team defined business performance metrics in the application, the management system was able to calculate the status of the business processes in real time based on the status of the underlying managed resources and by monitoring the transaction flows. The business analyst discovered that nine of the top ten business processes were performing adequately, but one of the business processes was running more slowly than normal. The management system reported that at the current performance level, the IT SLA would be violated.

Based on the business analyst's recommendation, the IT team was instructed to solve the performance problem with the slow business process first. The IT team used the relationship information provided by the management system to identify which services supported the slow business process. The IT team then used the relationship information to determine where the instances of the dependent services were running. The IT team brought up a performance monitoring tool (e.g., IBM Tivoli Monitor for Transaction Performance[22]) to locate the service instance that was performing poorly and affecting the business process, as shown in ***Figure 8***. The IT team also use the management system's monitoring tools (not shown in Figure 8) to discover that the machine hosting the slow service was running at 100% of processing capacity.

The IT team had now identified that too many services had failed over to the same machine, and one service in particular was running slowly enough to affect a committed SLA. After one of the failed five machines was restored, the IT team provisioned an instance of the critical service on the restored machine and configured the service bus to route traffic to both services. The business process performance was restored to normal, and the IT team restored the remaining services to their proper machines over the next few hours as the other machines were repaired.

CONCLUSION

SOA introduces new requirements for management as well as new opportunities to provide better systems management. The services in an SOA and the infrastructure supporting it need appropriate instrumentation and control interfaces to allow management of the entire IT environment. The modeling and development phases of an SOA can be used to ensure that the deployed SOA is manageable. Management systems must treat the services and other components of an SOA as first-class participants in the managed system and must recognize that services contribute to business processes and SLAs.

This paper has shown that the nature of SOA allows many management tasks to be implemented in a more reliable and integrated fashion. The concepts of SOA can also be applied to the management system itself to automate the management processes, to standardize the interfaces among different management system components, and to standardize the instrumentation exposed by the managed environment.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Mark Anderson, Rosalind Radcliffe, John Whitfield, John Harter, Sudhakar Chellam, and Phil Fritz. We also thank the WSDM technical committee for "catching the vision" and laying the foundation for the building bricks of the future. We also acknowledge the reviewers of this paper.

CITED REFERENCES AND NOTES

1. Eclipse is an open, extensible tool platform. See http://www.eclipse.org.
2. K. Brown and R. Reinitz, *Web Services Architectures and Best Practices*, IBM WebSphere Developer Technical Journal (2003), http://www-128.ibm.com/developerworks/websphere/techjournal/0310_brown/brown.html.
3. *Using Web Services Effectively*, Sun Microsystems (2002), http://java.sun.com/blueprints/webservices/using/webservbp.html.
4. For the implementation of EDI, see The Accredited Standards Committee (ASC) X12, http://www.x12.org/.
5. For EDI Internet integration, see http://www.ietf.org/proceedings/96dec/charters/ediint-charter.html.
6. W3C Web Services Description Working Group, http://www.w3c.org/2002/ws/desc/.
7. D. Cox, "How to Simplify IT Infrastructure Management: Using SOAs is Key," *WebSphere Journal* (November 2004), http://www.sys-con.com/story/?storyid=47216&DE=1.
8. *Business Process Execution Language for Web Services Version 1.1*, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.
9. Universal Description, Discovery, and Integration, UDDI.org Consortium, http://www.uddi.org.
10. *Understanding UDDI*, IBM developerWorks (2002), http://www.ibm.com/developerworks/webservices/library/ws-featuddi/?n-ws-7252.
11. OASIS Web Services Distributed Management Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
12. *Introduction and Applicability Statements for Internet Standard Management Framework* (2002), http://www.ietf.org/rfc/rfc3410.txt?number=3410.
13. W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. F. Morar, "Web Services Navigator: Visualizing the Execution of Web Services," *IBM Systems Journal* **44**, No. 4, 821–846 (2005, this issue).
14. M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, "The Enterprise Service Bus: Making Service-Oriented Architecture Real," *IBM Systems Journal* **44**, No. 4, 781–798 (2005, this issue).
15. Information Technology Infrastructure Library, Office of Government Commerce, http://www.itil.co.uk/.
16. *Common Information Model Standards*, Distributed Management Task Force (1999–2005), http://www.dmtf.org/standards/cim.
17. *Java Management Extensions Specification* (1998–2002), http://www.jcp.org/en/jsr/detail?id=3.
18. Web Services Interoperability Organization, http://www.ws-i.org.
19. Web Services Resource Framework Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
20. Web Services Notification Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.
21. N. Nagaratnam, A. Nadalin, M. Hondo, M. McIntosh, and P. Austel, "Business-Driven Application Security: From Modeling to Managing Secure Applications," *IBM Systems Journal* **44**, No. 4, 847–868 (2005, this issue).
22. *IBM Tivoli Monitoring for Transaction Performance*, http://www.ibm.com/software/tivoli/products/monitor-transaction/.

Accepted for publication June 3, 2005.

David E. Cox
IBM Software Group, 4205 South Miami Blvd, Research Triangle Park, North Carolina 27709 (decox@us.ibm.com).
Mr. Cox is a Senior Technical Staff Member in the Tivoli Technical Strategy and Architecture group in IBM. He is currently the lead architect for Web Services and service-oriented architecture management at Tivoli. He is a member of the Tivoli Architecture Board and a core member of the IBM Software Group Architecture Board. He is also a voting

member of the OASIS Web Services Distributed Management Technical Committee. Mr. Cox has 20 years of technical experience in systems and network management, communications software, and operating systems. He has written numerous technical papers and holds five United States patents. He received a B.S. degree from North Carolina State University and an M.S. degree from the University of North Carolina at Chapel Hill.

Heather Kreger
IBM Software Group, PO Box 12195, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (kreger@us.ibm.com). Ms. Kreger is a lead architect for Web Services and Management in the Standards and Emerging Technologies area. She is currently co-leader of the OASIS Web Services Distributed Management Technical Committee and member of several related DMTF (Distributed Management Task Force) work groups. Ms. Kreger was IBM's representative to the W3C® Web Services Architecture work group as well as co-lead of JSR 109, which specifies Web Services deployment in J2EE environments, and a contributor to the Java Management Extensions (JMX™) specification. She is also the author of numerous articles on Web Services and management in the *IBM Systems Journal, Communications of ACM, Web Services Journal,* and other public technical work, including the "Web Services Conceptual Architecture" and "WS-Manageability Standards." She is also the author of *Java and JMX, Building Manageable Systems.* ■