A configuration management database architecture in support of IBM Service Management

H. Madduri
S. S. B. Shi
R. Baker
N. Ayachitula
L. Shwartz
M. Surendra
C. Corley
M. Benantar
S. Patel

In this paper, we present the architecture of the IBM Tivoli® Change and Configuration Management Database. Its main features include a rich data model, automatic discovery of data for configuration items, visualization of application dependencies on configuration items, and multicustomer support. We discuss implementation topics, such as relationship management, composite configuration items, data federation, reconciliation of data from different sources, a security model for multicustomer support, and integration of change-management and configuration-management processes.

INTRODUCTION

The Information Technology Infrastructure Library** (ITIL**)[1,2] is a well-known set of best-practice guidelines for the delivery of quality information technology (IT) services. Configuration management, problem management, incident management, change management, service help desk, and release management make up the basic disciplines of the service support group of best practices.

Configuration management involves placing all configuration-related data on a single logical repository known as the configuration management database, or CMDB. The CMDB can be either a unified database or a federated database, a collection of databases that presents a single user interface. It stores configuration items (CIs) and their attributes and details about the relationships between CIs.

A successful CMDB implementation should satisfy the following requirements:

1. *A rich data model*—The data model should support all IT entities and their relationships. These IT entities can be physical, such as computers and devices, or logical, such as software applications and business processes.
2. *Automatic discovering of CI information and tracking of changes as they happen*—For the sake of productivity, it is critical to avoid the need for manual data entry. Large enterprises typically

rely on many tools and applications whose configuration data resides in a variety of repositories. As a result, a CMDB also must be able to handle reconciliation of data originating in different sources.

3. *Visualization of application dependencies on CIs*—In order to deal with the complexity of the modern IT environment, it is required that the customer be aware of the ways in which the applications that support the business depend on various CIs. The challenge can be overcome if the CMDB is able to visualize and display those dependencies. We refer to this property as "application visibility."
4. *Support for multicustomer environments*—In many deployments IT organizations have to support multiple customers, such as customers of a service provider or internal customers, when, for example, lines of business are supported by the IT organization. In such environments there are implications for security and process control because of the need to segregate data by customer account.

In this paper, we present the architecture and the main features of the IBM Tivoli* Change and Configuration Management Database (CCMDB), a CMDB that satisfies the preceding requirements and supports the IBM Service Management initiative.[3,4] IBM Service Management (ISM) is an approach designed to automate and simplify the management of business services.[5] We discuss implementation topics, such as relationship management, composite CIs, data federation, data reconciliation, a security model for multicustomer support, and integration of change-management and configuration-management processes. A number of products that cover some of these functions are offered by other vendors.[6–9]

The rest of this paper is organized as follows. In the next section, we present the CCMDB data model and describe how relations are managed. Next we present the CCMDB mechanisms for populating the database with data from different data sources and for reconciling inconsistencies in the obtained data. In the following section we present the CCMDB approach to visualizing application dependencies on CIs. Next, we describe the multicustomer support feature of CCMDB. In the next-to-last section, we present the CCMDB approach to integrating the change-management process and configuration-management process. The last section contains the conclusion.

CCMDB DATA MODEL

The data model of a database can be viewed as a conceptual representation of the structure of the data stored in the database. In addition to a variety of resource types, the model should be able to handle: (1) formal definitions of resources; these enable the import of resource data into the CMDB; (2) virtualized resources, both hardware and software; (3) complex resources and their structures made up from component parts; and (4) aggregation of entities into composite units.

Making use of our years of experience in systems management in various industries, we developed the CCMDB data model, whose primary role was to integrate data imported from Tivoli management products. It is defined using Unified Modeling Language** (UML**) diagrams, which enable us to partition large models into submodels. All objects, classes, and relationships are implemented with Java** persistent objects. Java programmers are thus able to manipulate objects and relationships, which are then stored in a database designed to support that model. The following types of constructs are defined in the CCMDB data model:

1. *Models*—Attributes, classes, and relationships belong to models. Models define name spaces for these entities and serve as the units of versioning. An example of a model is Computer System.
2. *Classes*—Classes represent IT entities. A class may have attributes and may participate in relationships. Examples of classes are `ComputerSystem` and `OperatingSystem`.
3. *Attributes*—Attributes are defined separately from classes and may be reused in any number of class definitions. An example of an attribute is `OSName` (operating system name).
4. *Relationships*—Relationships are defined separately from classes, and are strongly typed. Only relationships may have references. Furthermore, all relationships are binary. An example of a relationship is `InstalledOn` (relationship between classes `OperatingSystem` and `ComputerSystem`).
5. *Data types*—These are similar to data types in programming languages. A data type defines the set of values that can be associated with an

Table 1 Relationship `InstalledOn` in Javadoc format

OperatingSystem [ ]	***OSInstalled*** ***contained*** *: true* ***no-compare*** *: true* ***relationship-type*** *:* *name="com.collation.platform.model.topology.relation.InstalledOn"* *reverse="true"*

entity. An example of a data type is String (attribute `OSName` is of type String).

The data model built by IBM Tivoli contains representations for most IT entities found in a modern IT installation and is referred to as the Common Data Model (CDM). Currently, there are about 1000 classes, subclasses, and relationships in CDM, one of the richest data models for systems management in the industry. The model Computer System, which is the primary model for systems management products, is a combination of hardware (e.g., a machine) and software (e.g., an operating system). In the model Computer System both `ComputerSystem` and `OperatingSystem` objects are considered collections of interesting entities, so both are derived from class `System`. Included in this model is the representation of the Operating System as a hosting environment (a place where software can run). This concept is also applied later to other hosting environments, such as a Web application server (hosting environment for applications).

Relationship management

Relationships specify not only a connection between two entities—they also provide meaning (semantics) for the connection. Without relationships between data objects in a CMDB, the data store would simply be a repository for discovered configuration information and would not provide a complete view of the IT environment.

Examples of relationships are: `AccessedVia (AppServerCluster AccessedVia IpAddress)`, `Contains (ComputerSystem Contains FileSystem)`, and `RunsOn (OperatingSystem RunsOn ComputerSystem)`. A more comprehensive list can be found in References 4 and 10.

Relationships can be represented either implicitly or explicitly. Implicit relationships are built into aggregate or composite object groups; they connect objects of the same type, and queries involving such relationships are more efficient. Explicit relationships are separate objects, defined between the roots of those aggregates; they can connect objects of different types. CCMDB provides APIs (application programming interfaces) to assist users in traversing these relationships; most of the configuration queries can be expressed by a single filter expression. Explicit relationships can be handled dynamically during runtime and can support use cases in which the type of the target object is unknown or is different from the source.

Queries with implicit relationships

A computer system may have a number of operating systems installed on it. ***Table 1*** shows the relationships defined for the `ComputerSystem` class in Javadoc** format.

The `InstalledOn` relationship is defined as a dynamic array of unknown size, named `OSInstalled`. Each member of the dynamic array is an `OperatingSystem` object. With this definition, the CCMDB persistent objects layer creates an intermediary table with the key attributes from both `ComputerSystem` and `OperatingSystem` objects.

The API consists of a single method (named `find`) used to traverse and search data; it can be viewed as a filter expressed in an SQL-like language, the Model Query Language (MQL).[4] Most of the configuration queries can be performed by using this single filter expression. Under the covers, CCMDB translates the filter into SQL (Structured Query Language) statements, which take full advantage of the query optimization capabilities provided by the database management system.

The following are three MQL queries of varying complexity.

1. *Low complexity*—What are the operating systems installed on machine X?

```
SELECT OperatingSystem.*
FROM OperatingSystem, ComputerSystem
```

```
WHERE OperatingSystem.parent.guid ==
  ComputerSystem.guid
  AND ComputerSystem.fqdn == 'X'
```

2. *Moderate complexity*—What computer systems support a test environment with operating system Microsoft Windows** XP and database product IBM DB2 Universal Database* 8.2?

```
SELECT *
FROM ComputerSystem
WHERE OSRunning.OSName == 'Windows XP'
  AND exists(OSRunning.installedSoftware.
  productName == 'DB2 8.2')
```

3. *High complexity*—What computer systems use IBM-manufactured hardware, are running Windows XP, and have Norton Antivirus 1.6 installed?

```
SELECT *
FROM ComputerSystem
WHERE ComputerSystem.physicalPackage.
  manufacturer == 'IBM'
  AND OSRunning.OSName == 'Windows XP'
  AND exists(OSRunning.installedSoftware.
  productName == 'Norton Antivirus 1.6')
```

Queries with explicit relationships

Every explicit relationship contains two unique IDs: one for the source CI and one for the target CI. To traverse explicit relationships, CCMDB provides method `FindRelationship( )`. This method retrieves the relationship graph of a given relationship type, starting from a specified object. The result is an array of Relationship objects each of which consists of a source object, a target object, and a relationship type. The relationship graph can be traversed forward or backward, starting from any source, by relationship type, and terminate at any specified level without loops.

Explicit relationships are especially useful when either the type of the target object is not known or more than one type is involved. For example, if we have to determine what CIs are affected by an RFC (request for change), it is difficult to retrieve that information using implicit relationships. Because the type of the CI is not known, we have to use a query that searches each type of CI and determines if the source of an `Affects` relationship is an RFC object. The query searches iteratively until all links are found, a resource-intensive task. The task is much simpler if instead we use method `findRelationship()`.

```
Relationship [ ] affectsRelns =
  api.findRelationships(rfcGuid,
  api.CDB_DIRECTION_FORWARD,
  "com.collation.platform.model.topology.
    relation.Affects",
  2, null);
```

Here, `rfcGuid` is the globally unique identifier of the RFC object, the relationship traversal follows the defined direction (forward), the relationship type is `Affects`, and the search will stop at the second level of indirection.

Composite configuration items

In real-world enterprise systems, there are likely to be thousands of resources with complex relationships between them, and managing that complexity becomes difficult without grouping them effectively. Fortunately, there exist several natural grouping patterns, such as physical machine clusters, connected software systems, and collections of services. To address this complexity issue, CCMDB provides a means to manage a graph of closely related CIs as a *composite CI* (or simply *composite*).[11,12]

The composite is defined by a template-based filter that acts on the entire relationship graph to produce a specific subgraph. The template mechanism allows customers to change the definition of a composite based on their particular business needs. Composites are treated the same as any other configuration item, and they exist as distinct elements in the CMDB database. The composite graph is constructed by navigating relationships based on the template definition. For example, `Server` has a root `ComputerSystem` object, which then includes related hardware, operating system, software, and networking elements.

The identity of a composite is determined by the identity of its root element, which means it must contain all the attributes necessary, even if some of those attributes are derived from subcomponents. The subcomponents can be added, updated, or removed over time. For example, a `Server` can be created just with a `ComputerSystem` object, and an object `OperatingSystem` can be added later.

The rules for handling composites can get quite complex, depending on the relationship semantics; for example, "A composite can contain other composites"; or "A composite can be partially defined (does not contain all the subcomponent parts yet)."

Figure 1 shows the composite graph of a WebSphere* Application Server. The figure shows that the WebSphere Application Server composite contains the Server composite. The advantage of the composite concept becomes apparent when one observes that dealing with three composites is much simpler than handling dozens of subcomponents.

A composite instance can be created automatically after a discovery scan or explicitly from a user interface. A composite is created with a name, set of attributes, and associated naming rules to uniquely identify the composite instance. The user interface provides the list of valid atomic CI types and relationships that can be instantiated through navigation of the composite containment tree.

The search methodology is a depth first traversal of the containment tree of the root element. This algorithm visits the node of an atomic CI v after it has visited all other CI nodes in the subtree rooted at v. The traversal of a tree with n nodes takes $O(n)$ time (the algorithm has an order of n time complexity).

When a composite is deleted, it must be determined which of the supporting elements are also deleted and which are preserved but "disconnected" from the now defunct composite. This is determined case by case and depends on the semantics of the relationships involved.

LOADING AND MAINTENANCE

In this section we discuss the processes related to loading and maintaining the CCMDB: automated sensor-based discovery, discovery through application descriptors and component templates, data federation, data import, and reconciliation.

Automatic data and relationship discovery

Continuous and automatic discovery of configuration items is extremely important not only to avoid data entry costs but also to validate the current state of the database. CCMDB uses sensor-based discovery, be it credential-less "sniffing" of the network environment (i.e., collecting high-level configuration information) or credential-based discovery. The sensor is a new variation on the agent-less methodology that achieves nearly the depth of a local agent's discovery without incurring the deployment cost. A sensor basically emulates a user running locally on the monitored host, just like an agent—but only for a brief time. The sensor functions by using secure network connections, encrypted access credentials, and host native utilities.

The CCMDB discovery engine provides a framework in which to schedule, distribute, coordinate, and manage the various discovery sensors. Upon discovery initiation, the sensor performs a multistep process:

- It uses a standard protocol (SNMP, or Simple Network Management Protocol) to determine the IP (Internet Protocol) addresses of devices.
- If no credential is provided, it performs a shallow discovery, which extracts the operating-system-related information and provides a basic idea of the active assets in the environment.
- If the operating-system credential is provided, a secure connection is made to the host by using a secure protocol such as SSH (Secure Shell).
- When the session is attached, it captures open ports and the listening processes.
- Using native utilities, it discovers software and patches installed, applications hosted, and running processes.
- Based on the collected information, it invokes (using application credentials) other sensors to discover application configurations, such as DB2* or WebSphere.
- It sends the data collected to the discovery server, terminates the SSH session, and closes the connection.

Several instances of this process may be activated in parallel in order to handle a large workload. After collected data are stored in the database, a heuristics-based background process determines the implied relationships between CIs. For example, a WebSphere Application Server application communicating through a specific host/port might match a DB2 server listening to that port. By using these implied relationships, a complete dependencies graph is built, a vast improvement over the current state of customer tools.

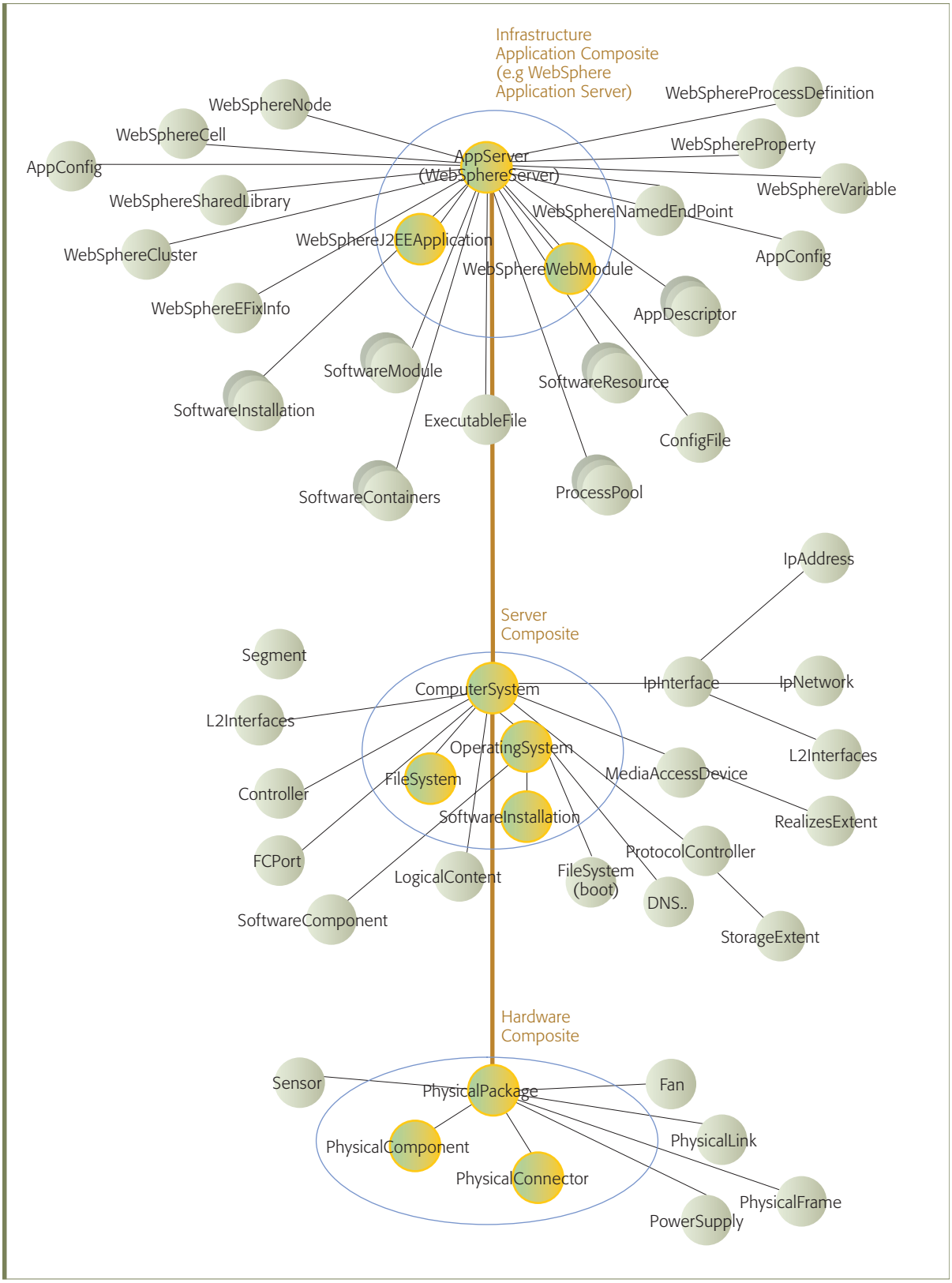


Figure 1
Composite CIs: template based tree pruning/filtering

Application descriptors are simple XML (Extensible Markup Language) files that are added to application modules at the time of packaging. These application descriptors allow the discovery engine to automatically create and maintain *business service* groupings (a grouping used to form a higher-level aggregated object).

The discovery engine can also discover a business service by identifying component signatures and designating matching customer components as "belonging to" a given business service. The *component templates* specify the unique signature of the components by using combinations of items such as program names, ports, and environment variables for classification.

The following are the main features of sensor-based discovery:

- Sensors can retrieve as much data about applications as is necessary.
- Sensors can work in conjunction with existing legacy agents, using the agents as just another data source.
- Sensors leverage remote management protocols to gather application- and platform-specific data.
- Sensors are easy to deploy and manage, with or without credentials.
- Sensors have limited impact on the target machine, consuming typically less than 1 percent of CPU processing when active.
- Sensors provide a flexible end-to-end view of applications and configurations and their dependencies.

Data federation

Consolidating data into a single physical data store has been the most commonly used way to achieve fast, highly available, and integrated access to related information.[13,14] Consolidation is often justified by performance or by the consistency achievable with a single master copy. Federation, on the other hand, makes distributed data appear as if it were a single source, regardless of the location, format, and access language. Different from the data consolidation approach, federation does not physically bring data into the central repository. CCMDB combines the strengths of both data consolidation and data federation. The underlying federation technology is the IBM Information Integrator technology using DB2 as the federation engine.[13–16]

After the external data sources are configured, they can be queried and processed through DB2 SQL. These external sources can be relational data sources, such as DB2, Microsoft** SQL Server, Oracle Database 10g, and Sybase Open Server, and nonrelational data sources, such as XML documents, Excel** files, Web Services, IBM WebSphere MQ, VSAM* (virtual storage access method) data sets, and IBM IMS* (information management system) databases.

Figure 2 illustrates the CCMDB data integration architecture. The CI and relationship information are retrieved from their location in storage, either the physical data store or the logical data store. For data federation, we leverage the IBM Information Integrator technology to integrate data from heterogeneous data sources without requiring all the data to be copied centrally. CCMDB also provides data consolidation services to move data physically into the database. The data consolidation services include IDML import and discovery sensors.

Data import with IDML

To enable data import into CCMDB from virtually any source, IBM developed the Identity Markup Language, or IDML.[4] An adapter program extracts data from a source application such as Tivoli Provisioning Manager (TPM) or Tivoli Configuration Manager (TCM) and produces IDML files. These files are bulk loaded into CCMDB. In many cases, a source might already have an export format that can be used with a simple XML translation.

Reconciliation

Because CI information may originate in more than one source and because CI records from different sources are usually not identical, a process of reconciliation is necessary. The CCMDB reconciliation for a CI first requires the one-time task of developing naming rules in which a set of attributes that could identify the CI is specified, along with a priority for each attribute set. The major parts of a naming rule are:

- *Class*—the class used to name instances; for example, `ComputerSystem`.
- *Superior class*—a class larger than the given class (if it exists); for example, an `OperatingSystem` class named as a subordinate to the `ComputerSystem` class; its name includes that superior name.
- *Naming attribute*—a character string that denotes an attribute which could identify the CI.

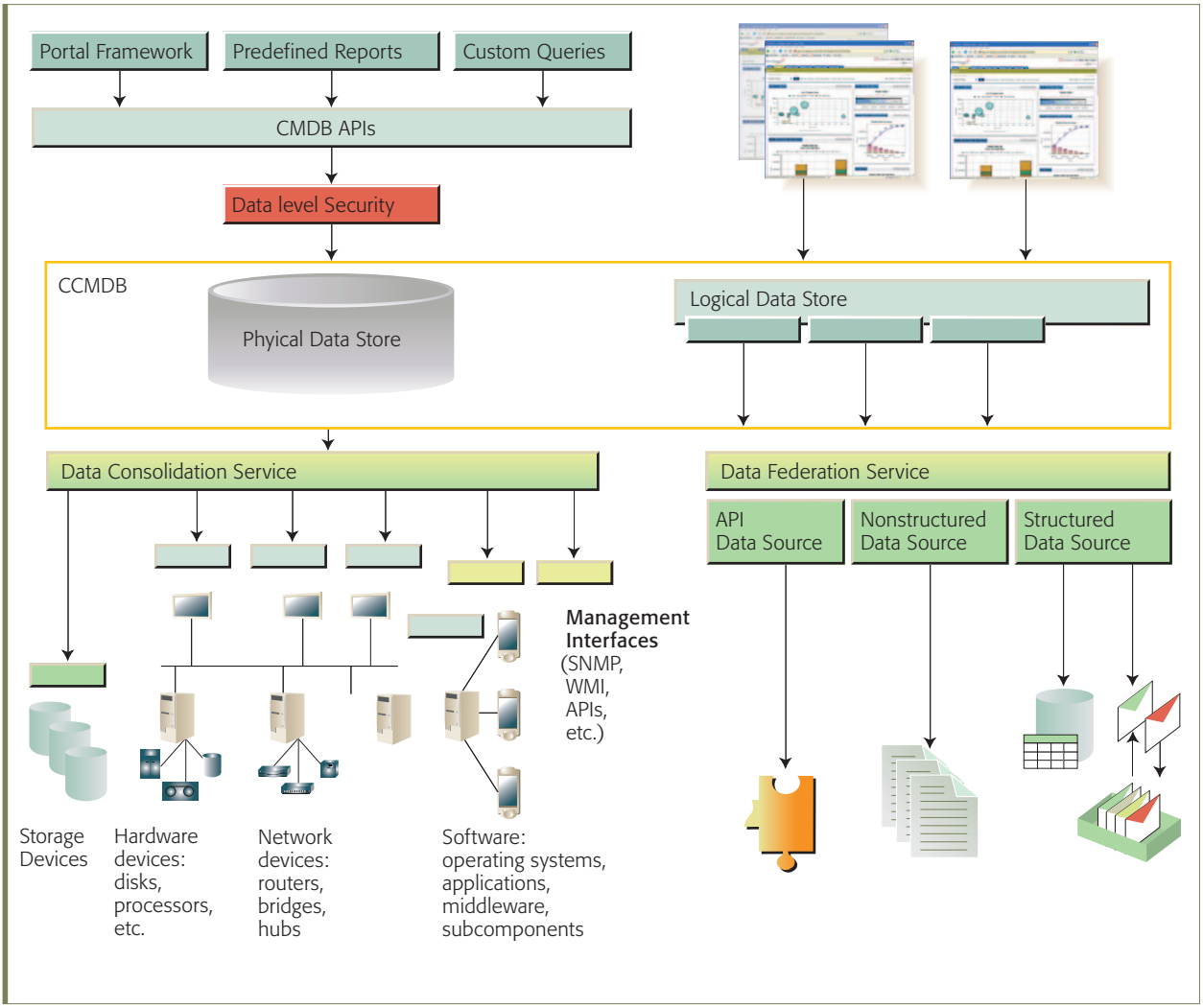


Figure 2
CCMDB data integration architecture

- *Priority*—a non-negative integer that signifies the priority in which naming attributes will be used to identify a CI.

The following are the naming rules for the `ComputerSystem` class:

0='signature'
1='manufacturer,model,serialNumber'
2='systemBoardUUID'
3='primaryMACAddress'
4='hostSystem,VMID'
5='managedSystemName'

The second rule, for example, is associated with priority 1 and naming attribute 'manufacturer,model,serialNumber'.

When creating an instance of a `ComputerSystem` class, the following steps are performed:

- Check that the request includes at least one naming attribute. The request is rejected if an insufficient number of attributes is provided or if the parent object of that instance does not exist.
- Generate the Type 3 GUID (globally unique identifier, based on IETF [Internet Engineering Task Force] RFC 4122),[17] according to the priority and the name string generated from the rules. If the parent is specified in the naming rule, obtain the naming attributes from the parent.
- Match any existing CI instance with the same GUID or alias.

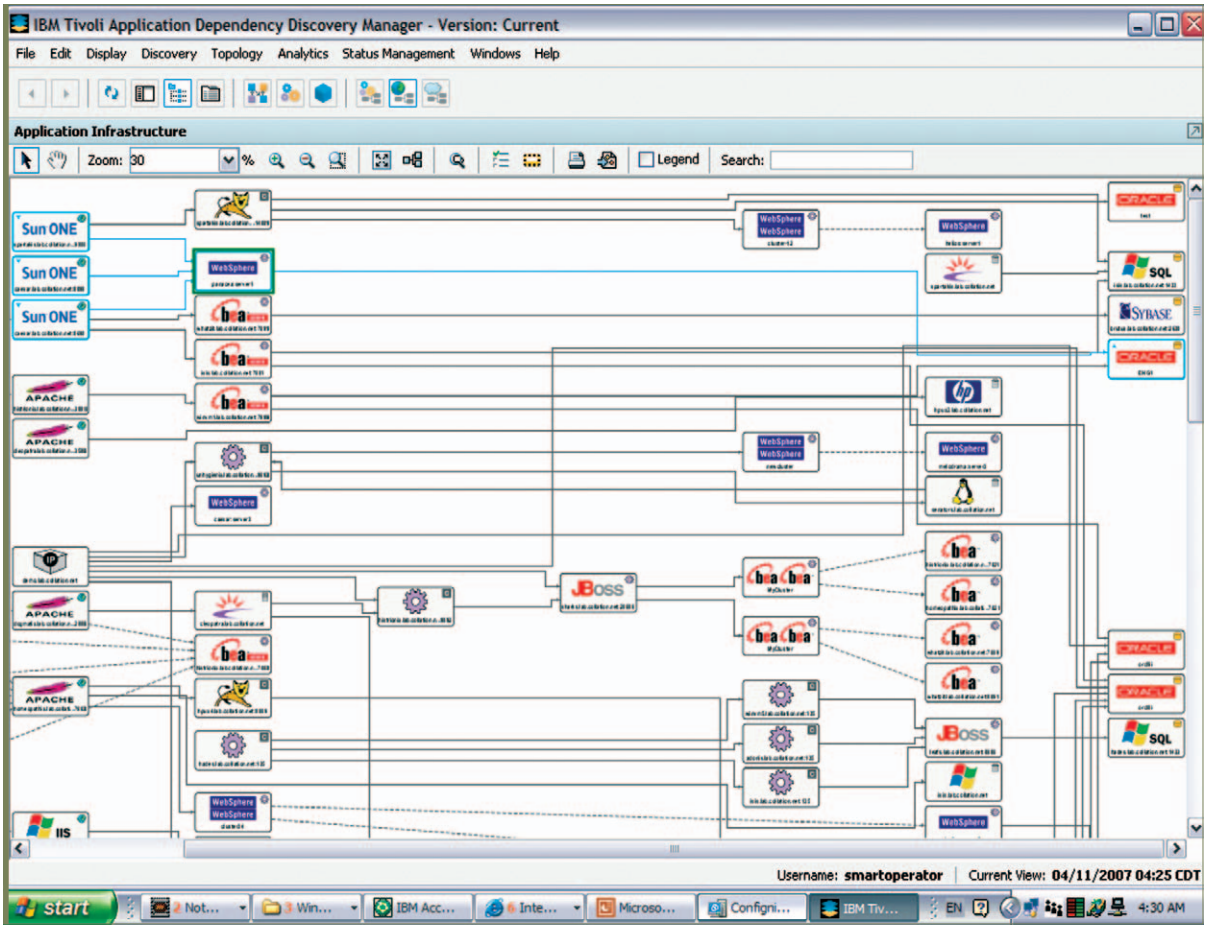


Figure 3
Visualization of application dependencies

- It is still possible that multiple copies of a CI are not matched until more information is provided. At that point, duplicates are merged and aliases are updated.

VISUALIZATION OF APPLICATION DEPENDENCIES

Many of today's IT services and applications are built on an infrastructure that consists of thousands of components. Most customers in the process of implementing an IT service management initiative are looking for a solution that addresses this complexity. It is imperative that the customer be able to react quickly in case of problems, for which it is required that the customer be aware of the ways in which the applications that support the business depend on various CIs. Consequently, it is necessary that the CCMDB provide visualizations of such application dependencies. In addition, these visualizations should be integrated with other business-related transformations, such as implementations of service-oriented architectures or business system dashboards.

The CCMDB visualization of application dependencies is illustrated in *Figure 3*. The display contains a wide variety of software components running on various application server platforms such as IBM WebSphere, Apache** HTTP Server, and BEA WebLogic Server**. All the discovered CIs and the relationships between them can be viewed as a graph. Each node in this graph represents an infrastructure software component (middleware) in the enterprise server environment. The infrastructure software includes application servers, Web servers, databases, and various system services such as the Domain Name Service (DNS) and the Lightweight Directory Access Protocol (LDAP). The edges (connecting lines) of the graph represent the

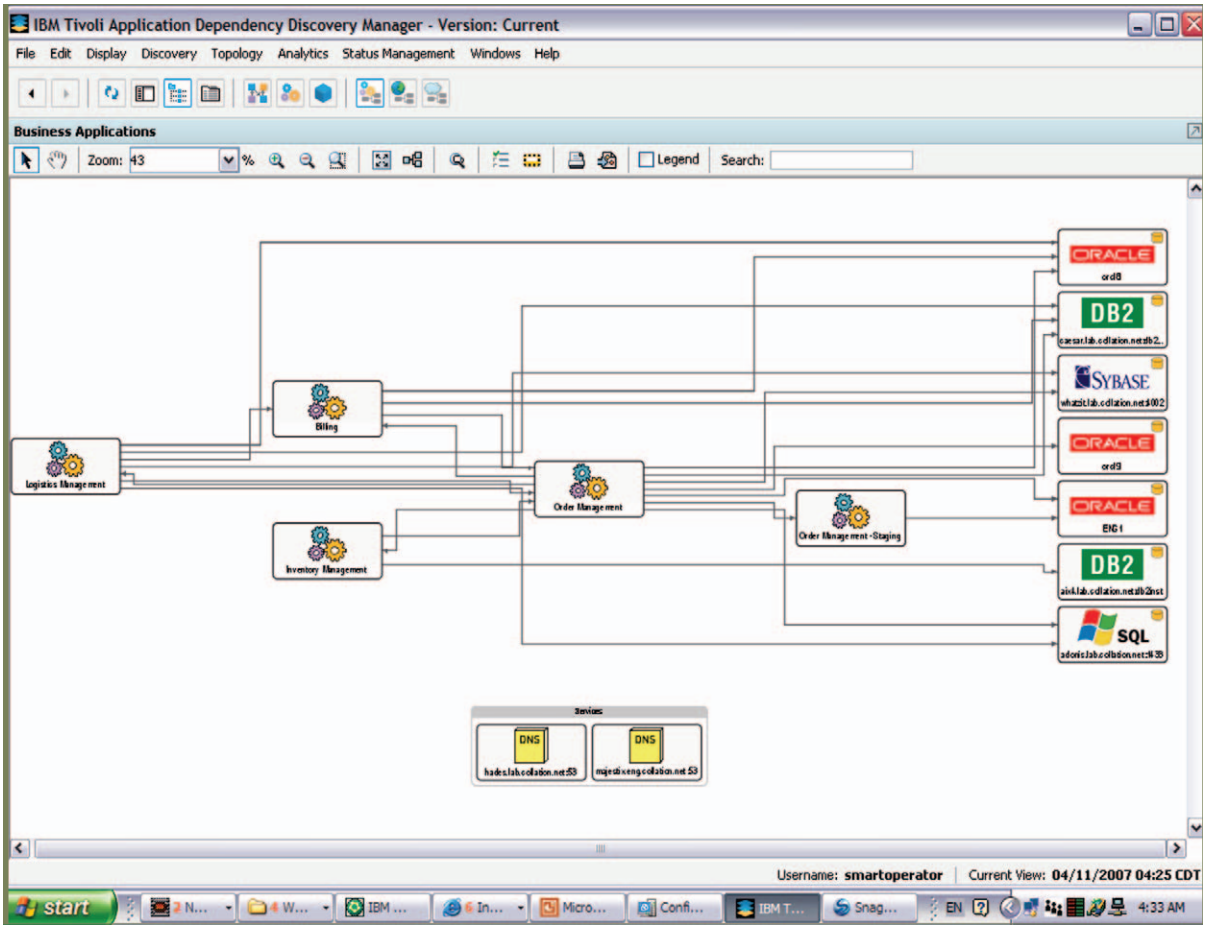


Figure 4
Visualization of dependencies of business services on business applications

relationships between the components. In this example, there are three Sun ONE Web Servers** that depend on a WebSphere Application Server. The WebSphere Application Server depends in turn on an Oracle database for its database services.

CCMDB also provides visualizations of business services and their dependencies on business applications (*Figure 4*). For example, Order Entry is a business service that can be delivered by integrating application components from the Order Management, Inventory Management, and Billing business applications. Users can create business services to simplify the infrastructure by combining large collections of individual components into groups.

MULTICUSTOMER SUPPORT

IT service providers rely on the CMDB as a major component in support of IT service management processes. These providers often face the challenge of integrating multiple worldwide operations onto the same physical infrastructure. To accomplish this they have to implement data segregation and access control, so that an application can be deployed once and shared among many customers. When the same application is configured to serve more than one customer, we distinguish between multicustomer support and multitenant support.

Multicustomer—A service management application with multicustomer functionality is one that allows a service provider to manage data and services on behalf of multiple customers with the appropriate data segregation between customers. In an application implementing this functionality, only the service provider has access and control over managed data and services, and only the people employed by the service provider have access to it.

Multitenant—Multiple customers access a single instance of the service management solution with full access to the data and services that they manage and the appropriate data segregation in place to keep them from accessing data and services that they do not manage. In an application implementing this functionality, there is still a service provider, but its role is primarily to mediate the service requests between customers and maintain the core application functionality.

Indications are that customers would prefer a multitenant solution if configurability and security could be ensured at the same level as in the single-tenant solution. To provide multicustomer support a CMDB must have built-in configurability in order to allow the tailoring of the solution to meet the business needs of each customer. With well-designed multicustomer support, the service provider can

- leverage a common pool of human resources,
- leverage common libraries (software portfolios, etc.),
- segregate customer data for improved security,
- generate reports that aggregate data for groups or all customers, and
- provide management personnel real-time key performance indicators when these are obtained by aggregating data for a group of customers.

The CCMDB multicustomer approach reduces the cost and management burden on the supplier, and the savings can be passed on to the customer. *Figure 5* illustrates the CCMDB multicustomer security model.[18] The solution is built upon the base ITIL CMDB security architecture with an option to create additional access roles and permissions. The roles and permissions could be added and assigned to a user at runtime.

Roles and permissions

To support a service provider in multicustomer environments, we must adapt to the service provider organization and its unique processes. This translates into a different collection of `roles and permissions` defined for each `Customer` or `Account`. In addition, other processes can have `roles and permissions` beyond those in configuration management. Configuration manager is one of the roles identified by ITIL. In a multicustomer CMDB the role of customer configuration manager has a scope of `Customer` with responsibility for the customer's configuration-management process and the ability to manage other roles, permissions, and CIs for the customer. The customer configuration librarian is an owner of the `Account` CIs and manager of all master copies of the `Account` CIs.

Data segregation in configuration management uses the notion of `AccessCollection`s, which are secured containers of groups or individual CIs. Any administrative unit (customer, organization, division) could be associated with an `AccessCollection`, which contains the CIs related to the unit. Allocating a CI to any administrative unit is done by adding the reference of the CI to the `AccessCollection` for this unit. To represent the notion of account ownership of resources, for example, the relationship type `Owns` has been created. The relationship type `Uses` represents a potentially shared usage of a CI.

In Figure 5, `Customer`, `Account`, `ServiceProvider`, `Organization`, `workGroup`, `Person`, `Role`, and `AccessCollection` are all administrative units. They are interrelated, with a predefined set of relationships to provide this support. Although in general a person may have many roles and each role may have access to certain `AccessCollection`s (of CIs), sometimes the access to a collection by a person has to be restricted to only when he or she acts in a specific role. This is accomplished by a `Person-In-Role` object. The assignment of `Person-in-Role` to support a set of CIs is done through the association of `Person-in-Role` to the `AccessCollection` that has those CIs as members. The `AccessCollection`s are protected by the Security Manager, which is a pluggable component in the CCMDB. The assignment of `AccessCollection`s to user-role pairs, as well as roles to permissions, is stored in the security policy. At any time the CMDB database and the security policy can be updated to include new roles, permissions, users, and access collections. The approach is designed to balance centralized security management with performance of the query results.

To configure a limited multitenant solution, the role of super configuration manager should be created with the permissions to create `Customer/Account` and set up CCMDB security. The person assigned to this role has the responsibility to create `Customer/Account`, allocate CIs for these units, and assign

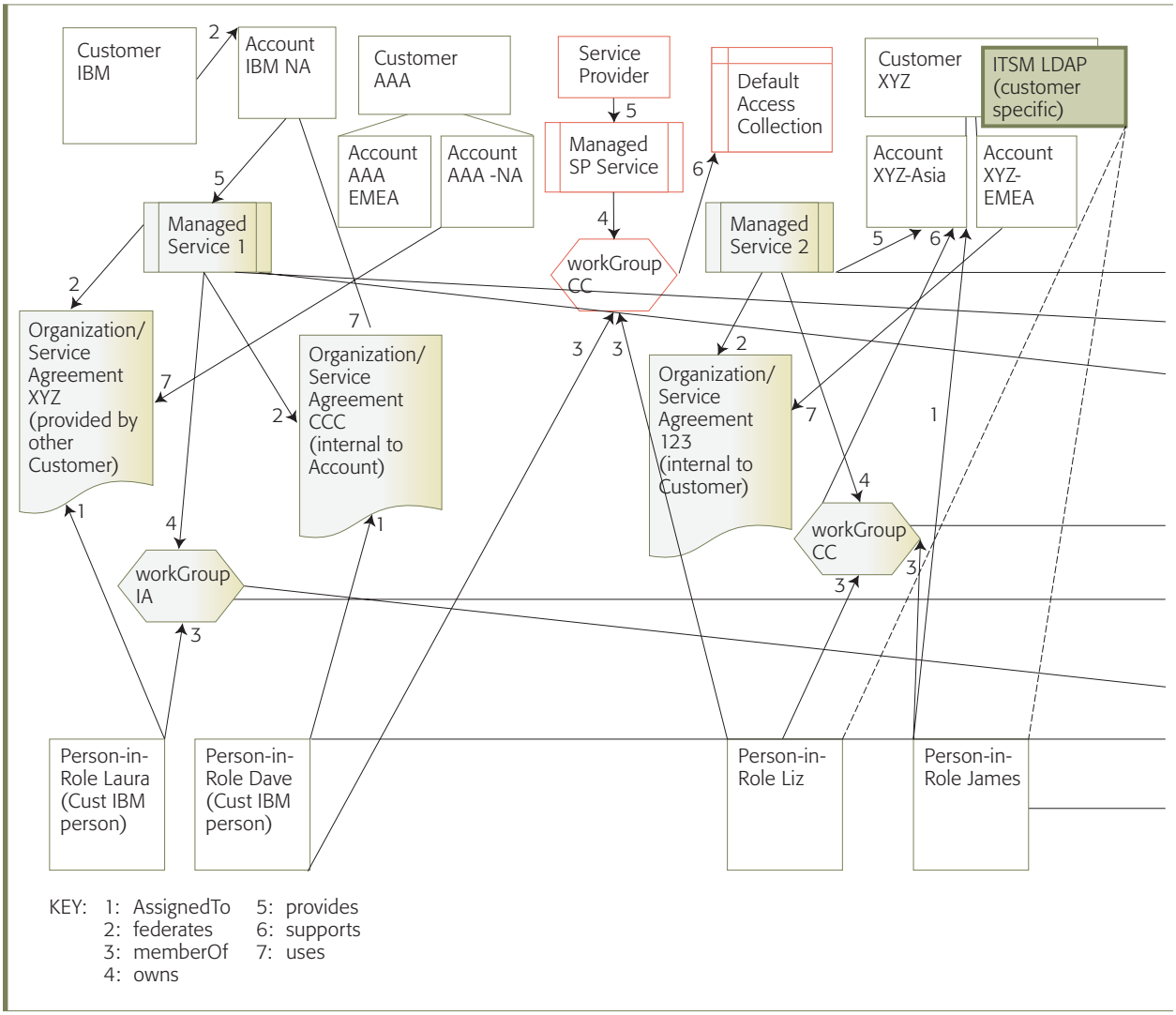
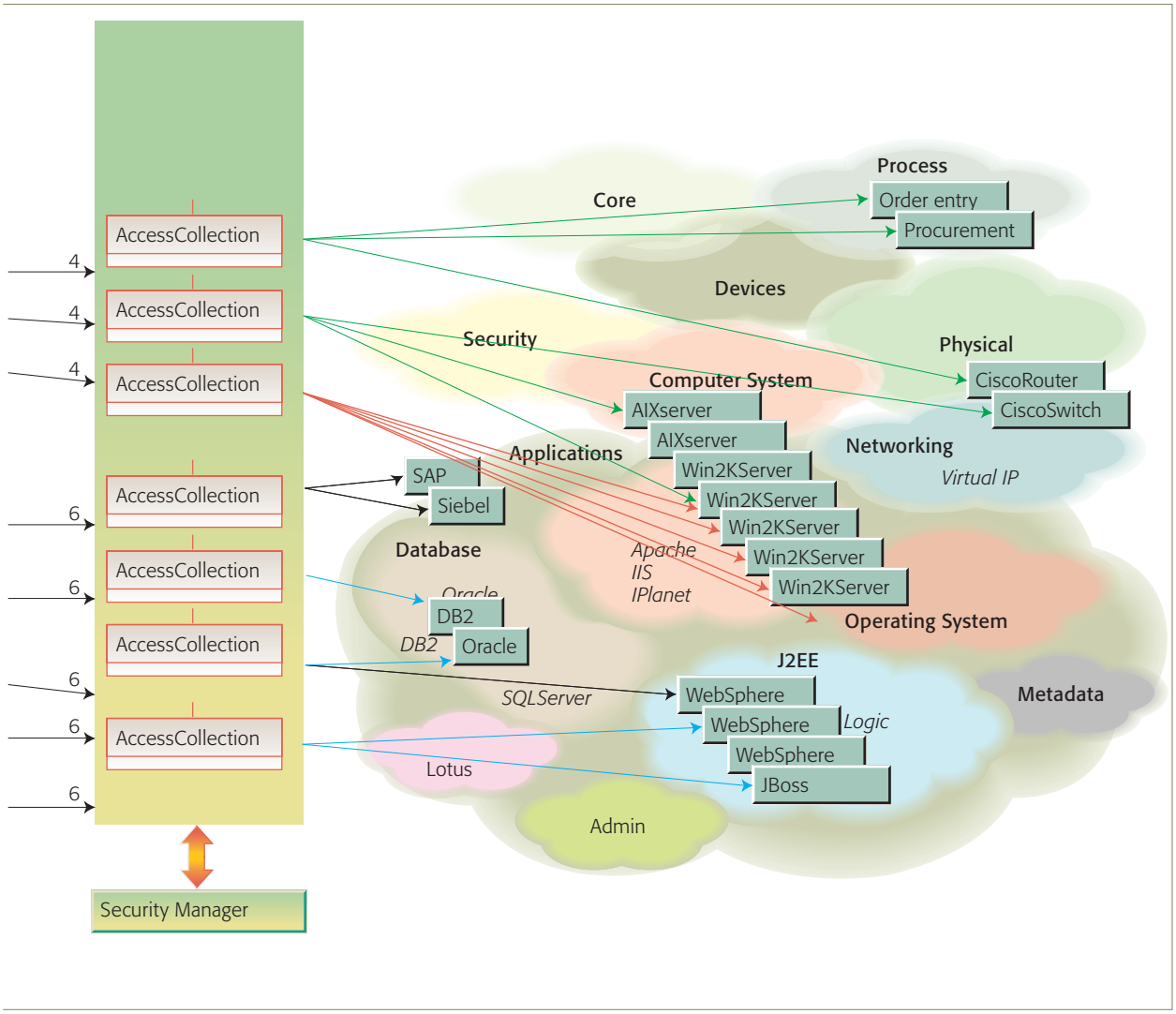


Figure 5
CCMDB multicustomer security model

`People` to `Customers`, `Accounts`, and `Organizations`.

INTEGRATED CHANGE AND CONFIGURATION PROCESSES

Deriving full benefits from a CMDB requires that the change-management process and the configuration-management process be tightly integrated. In today's IT environment, having a standard process for managing changes to the IT infrastructure and IT services is essential. It is hard to imagine how one could manage the adverse effects of change without maintaining an accurate representation of what the configuration should be (we refer to it as the authorized environment). Conversely, it is hard to imagine how one could maintain an accurate representation of the authorized environment without a process to manage the changes in the environment. This is why ITIL states, "Ideally, Change Management should be regarded as an integral part of a Configuration Management system."[1] In fact, configuration management and change management are treated separately in ITIL only because historically some organizations have implemented change management without a full configuration-management process to support it. However, ITIL recommends that these two processes be planned and implemented together.[1]



These processes must handle the complexities of a large multicustomer environment, which include a large infrastructure and multicustomer support. Change management and configuration management represent the main control processes for the IT environment. Change management relies on configuration management for accurate information on the authorized view of the environment. Configuration management relies on change management for information on planned and completed changes to the environment so that an accurate representation of the environment is maintained. According to ITIL, all changes to the environment, other than standard service requests, should be under the control of change management. CCMDB is one of the first products in the industry that fully integrates change and configuration management.

The main features of the CCMDB integration of configuration management and change management are listed next, and we discuss them in the rest of this section.

- Manage relationships between an RFC and each of the affected CIs
- Represent CIs and their protected life-cycle states and transitions by using templates for customization

- Create and maintain different versions of CI (authorized, actual, gold standard, baseline, etc.)
- Enforce change control on CI by the *Control CI* subprocess (explained later)
- Remediate variances between actual and authorized CI records
- View authorized changes to CIs on the attributes under change control

The goal of change management is to minimize the adverse effects of changes to the environment. A change is defined as any installation or alteration of hardware, system and application software, procedures, and environmental facilities that adds to, removes from, or modifies the service delivery environment. Change requests can be initiated by an administrator or by an automated service support process.

The change-management process maintains information in the CMDB on each RFC throughout its life cycle. The information includes relationships between the change and the affected CIs. It is essential that this information be updated by using the configuration-management process.

Configuration management is the process for identifying, defining, and maintaining information on the IT components and services of an IT system. Configuration management also maintains information on how those components relate to one another and to service-support (part of ITIL) process artifacts, such as change records. This logical representation of the environment is used by the other ITIL processes in service support and service delivery. Thus, configuration management includes as subprocesses: identify CI, control CI, verify and audit CI, and report status of CI.

According to ITIL, configuration management should ensure that no item is changed, added, or deleted "without appropriate controlling documentation."[1] Prior to updating the life-cycle state of an RFC to "Closed," the change process should ensure that the environment and the CMDB have been updated appropriately. For some low-level changes, it may be difficult or impossible to know in advance how a change will manifest itself in the environment.

As part of the control-CI subprocess, every configuration item has a life-cycle state associated with it. The life-cycle state is used for tracking purposes and should be kept current and made available for planning, decision making and managing changes. Example life-cycle states are: `ordered`, `received`, `in acceptance test`, `live`, `under change`, `withdrawn`, and `disposed`. Transition between life-cycle states must be managed so that an item is moved only to another legal state. The history of these transitions is also kept and available for inspection.

Given a state transition diagram, a subset of the CI life-cycle states may be designated as *protected*, affording a greater degree of control over the way CIs can be modified. The designation implies that changes in this state must be associated with a change record.

The CI information in CCMDB can have several versions, each version representing a different aspect of the CI. This feature is often overlooked in the industry. The CI versions in a CMDB are the authorized version, the baseline version, the actual (discovered or audited) version, and the planned version.

The controlled attributes of a CI (i.e., the ones amenable to change) and its life-cycle states and transitions should be implemented using templates, which are mapped to data model entities in the CMDB. Data from discovery and other data entry processes provide the *actual* data, while the change-management process provides the *authorized* data.

A configuration *baseline* is a snapshot, at a specific instant, of a set of authorized configuration items. The concept is useful for documentation, recovery, and especially comparison. Baselines have a name and time stamp associated with them, and customers can create new ones as necessary.

A similar concept used in many service provider environments is the *gold standard*. A gold standard is a set of CI records that serve as a model or template for the way in which other sets of CIs should be configured. Relationships between a gold standard and any number of sets of CIs can be created to establish applicability of a gold standard to those sets. The gold standards are used by the configuration-management verify-and-audit-CI process for assessing compliance with established policy. For example, an organization might create a

gold standard to represent a typical UNIX** server configuration, then create relationships between the individual UNIX servers in the network and this gold standard. When the verify-and-audit-CIs process is executed on those UNIX servers, a report of any compliance discrepancies between the gold standard and the associated servers is generated.

Transition between life-cycle states must be managed so as to ensure that, from a particular state, a CI is moved to only another legal state. As part of the configuration-management process, this life-cycle transition should include attribute-level semantic validation. This validation capability recognizes that there are life-cycle states in which a greater degree of control is required than in other states. Because a CI record is a reflection of all changes that have taken place on a CI, it maintains a list of changes to that life-cycle state. For more details on CI life-cycle state management refer to Reference 19.

CONCLUSION

In this paper, we present the architecture and main features of the IBM Tivoli Change and Configuration Management Database (CCMDB). We describe the main features, which include a rich data model, automatic discovery and maintenance, visualization of application dependencies in CIs, multicustomer support, and integrated change-management and configuration-management processes. We also describe implementation aspects, such as relationship management, composite objects, sensor-based discovery, data federation support, inline reconciliation, and multicustomer security issues. This design was implemented in the CCMDB product release 1.1 (released June 2006).

A CMDB is clearly of great value to an IT organization. It is also the foundation upon which other IT processes, and many business processes, are built. Businesses now have the information required to analyze the business environment, which leads to improvements in efficiency and effectiveness. CCMDB provides the technology required to enable business compliance, business process intelligence, reporting for security markets, and financial controls.

ACKNOWLEDGMENTS

For lucidly articulating their requirements and for guiding us to better design solutions, we thank the many customers who contributed to this work. We also thank Yan Or, Krishna Garimella, Johan Casier, Shashank Joshi, Anand Sankaran, Girard Chandler, Robert Nielsen, Ling Tai, Ben Jeffcoat, Jogeswar Challapalli, and Jinfang Chen for their contribution to the design and implementation of CCMDB. We thank management for their extraordinary support: William Kopkind, Vinu Sundaresan, Mike Mallo, Jim Stubley and Craig Love. For helping us make use of the database federation technology, we thank Eileen Lin, Mei-Mei Fu, and the IBM Information Integrator development and management teams.

CITED REFERENCES

1. *Introduction to ITIL*, The Stationery Office, Office of Government Commerce, United Kingdom (2005), http://www.itil.org.uk.
2. *Foundations of IT Service Management Based on ITIL*, J. Van Bon, M. Pieper, and A. van der Verrn, Editors, ITSM Library, Van Haren Publishing (November 2006).
3. IBM Tivoli Change and Configuration Management Database (CCMDB), IBM Corporation, http://www-306.ibm.com/software/tivoli/products/ccmdb/.
4. *IBM Tivoli CCMDB Developer Reference Guide* (September 2006—available from the author).
5. D. Lindquist, H. Madduri, C. J. Paul, and B. Rajaraman, "IBM Service Management Architecture," *IBM Systems Journal* **46**, No. 3, 423–440 (2007, this issue).
6. BMC Software, Inc., http://www.bmc.com/.
7. Relicore Clarity, Symantec Corporation, http://www.symantec.com/enterprise/support/overview.jsp?pid=53294.
8. Mercury Application Mapping, Hewlett-Packard Development Company, L.P. http://www.mercury.com/us/products/business-availability-center/application-mapping/works.html.
9. nLayers, EMC Corporation, http://www.nlayers.com/.
10. S. Patel and S. S. B. Shi, *Managing Relationships with Tivoli Configuration Management Database (CMDB)*, IBM CMDB White Paper (October 2006, available from the author).
11. N. Ayachitula, L. Shwartz, K. Garimella, and Y. Or, *Methods and Apparatus for Composite Configuration Item Management in Configuration Management Database*, U.S. Patent No. 920,060,467 (pending).
12. N. Ayachitula, L. Shwartz, M. Surendra, K. Garimella, and Y. Or, *Methods and Apparatus for Automatically Creating Composite Configuration Items in a*

Configuration Management Database, U.S. Patent No. 920,060,469 (pending).

13. A. Betawadkar-Norwood, E. Lin, and I. Ursu, "Using Data Federation Technology in IBM WebSphere Information Integrator: Data Federation Usage Examples and Performance Tuning," developerWorks, IBM Corporation, http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0507lin/.
14. L. M. Haas, E. T. Lin, and M. A. Roth, "Data Integration through Database Federation," *IBM Systems Journal* **41**, No. 4, 578–596 (2002).
15. A. Betawadkar-Norwood, E. Lin, and I. Ursu, "Using Data Federation Technology in IBM WebSphere Information Integrator: Data Federation Design and Configuration," developerWorks, IBM Corporation, http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0506lin/.
16. *Data Federation with IBM DB2 Information Integrator*, IBM Redbook SG24-7052, IBM Corporation, http://publib-b.boulder.ibm.com/abstracts/sg247052.html?Open.
17. *A Universally Unique IDentifier (UUID) URN Namespace*, IETF Network Working Group RFC 4122, http://www.ietf.org/rfc/rfc4122.txt.
18. L. Shwartz, G. Aikens, N. Ayachitula, M. Benantar, K. Garimella, H. Madduri, M. Surendra, S. Weinberger, and Y. Or, *Method and System for Segmenting Data and Role Based Access Control for Multi-Account Infrastructure Management*, United States Patent No. 920,060,468 (pending).
19. C. Ward, V. Aggarwal, M. Buco, E. Olsson, and S. Weinberger, "Integrated Change and Configuration Management," *IBM Systems Journal* **46**, No. 3, 459–478 (2007, this issue).

Accepted for publication March 15, 2007.
Published online July 12, 2007.

Hari Madduri
IBM Tivoli Software, 11501 Burnet Road, Austin, TX 78758 (madduri@us.ibm.com). Dr. Madduri started his career as a System/370® assembler programmer/analyst, obtained a Ph.D. degree in computer science in 1985 from the University of Wisconsin-Madison. Since joining IBM in 1990, he played various lead technical and management roles in object-oriented systems (DSOM), data mining (chief architect of data-mining products), e-commerce hubs, electronic data interchange, and IBM Global Services service development (e.g., UMI). In IBM Tivoli, he contributed to early ITIL process prototypes, which led to the current IT service management strategy. He is currently lead architect for the CCMDB product. Dr. Madduri taught undergraduate and graduate classes in programming languages, compilers, and operating systems at University of Wisconsin-Madison, St. Thomas University (Minneapolis), and University of Hyderabad (India). He published over 20 papers and authored 20 United States patents.

Shepherd S. B. Shi
IBM Tivoli Software, 11501 Burnet Road, Austin, TX 78758 (sshi@us.ibm.com). Dr. Shi is a Senior Technical Staff Member. He has a B.S. degree in computer science from National Taiwan University, an M.S. degree in computer science from Stanford University, and a Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign. Since joining IBM in 1990, Dr. Shi has led the architecture and design work of a number of major projects, such as DB2 Connection Services, IBM LDAP directory, DCE, DFS™ WebSecure, and the Tivoli Security Management product suite. Dr. Shi has more than 30 patents and was recognized as a Tivoli Master Inventor in 2006.

Ron Baker
IBM Tivoli Software, 1516 Westfall Circle, Sanford, NC 27330 (rbbaker@us.ibm.com.) Mr. Baker started his career in the aerospace industry, designing and writing numerical programs for engineering graphics and robotics applications. As relational databases began to appear, Mr. Baker was one of their early users in large-scale financial and configuration management applications at Boeing. After several years, he moved into research on database parallelism and integrity constraints at Amoco's Computing Research Center, followed by work at Northrop Grumman as an engineering configuration database specialist, where he addressed transitive closure problems like bill-of-material processing and reconciliation between configurations. It is also where he gained experience with statistical process control in engineering and manufacturing systems. Mr. Baker joined IBM in 1989 to work on object-oriented language integration with relational databases, an area in which he holds several patents. Since then, he has worked on management products dealing with unstructured documents, search engines, and Internet services. He was the lead CMDB Architect for Tivoli Software, and is now a Senior Technical Staff Member responsible for overall data integration initiatives and advanced analysis reporting.

Naga Ayachitula (Arun)
IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (nagaaka us.ibm.com). A senior software engineer currently involved in computing services and IT service management, Naga Ayachitula has developed innovative approaches to automating compliance, network admission control and remediation in the IBM Integrated Security Solution for Cisco Networks. He has over 15 publications and 15 patents pending.

Laura Shwartz
IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (lshwart@us.ibm.com). An advisory software engineer with research experience in mathematics, computer science, and software design and development, the research interests of Ms. Shwartz include service management, autonomic computing, workload management and provisioning, data modeling, and non-commutative probability. She is working toward a Ph.D. degree in mathematics.

Maheswaran Surendra
IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (suren@us.ibm.com). Dr. Surendra received a Ph.D. degree in 1991 from the University of California at Berkeley and has been at IBM Research since that year. He has worked in technical areas ranging from semiconductor manufacturing to software systems management, and most recently in IT service delivery. He is currently a senior manager in the Services organization in IBM Research, and his focus is the application of IT service management technologies in service delivery operations.

Carole Corley
IBM Software Group, Tivoli Division, 11501 Burnet Road, Austin, Texas 78758 (ccorley@us.ibm.com). Carole Corley is an advisory software engineer specializing in management application security. She has a B.S. degree in engineering science and an M.S. degree in aerospace control systems, both from the University of Florida.

Messaoud Benantar
IBM Software Group, 11501 Burnet Road, Austin, TX 78758 (mbenanta@us.ibm.com). A senior software engineer,

Messaoud Benantar works on the security services of the WebSphere Application Server platform. His interests are in applications, systems, and network security. Dr. Benantar has a diplome d'ingenieur from the University of Science and Technology in Algiers, Algeria, and M.Sc and Ph.D. degrees from Rensselaer Polytechnic Institute in Troy, New York.

Sushma Patel
IBM Tivoli Software, 11501 Burnet Road, Austin, TX 78758 (patelsb@us.ibm.com). A software engineer, Ms. Patel works on the CCMDB portion of the ITSM portfolio; her development work is concentrated on the APIs, providing the command-line, RMI, and SOAP interfaces, as well as developing discovery sensors and working on relationships. She has a B.S. degree in computer science and an M.S. degree in science and technology commercialization from the University of Texas at Austin; she has submitted several patents and given presentations at conferences on various topics involving innovative teaming and collaboration. ■