CHAPTER 2

What Is a CMDB?

Rarely can an IT operations or management tool discussion occur today without someone inevitably bringing the conversation around to the Configuration Management Database (CMDB). It is a term that is intimately intertwined with the IT Infrastructure Library (ITIL) and any other IT operations concept. In fact, the whole notion of operational excellence is a panacea without a robust CMDB serving as its foundation.

When you think about it, every function performed in IT requires accurate information to drive decisions. Indeed, this relationship between information and decisions spans far beyond IT. With no information, decisions are based on mere guesswork. It is no wonder that these decisions are so frequently erroneous. To make informed decisions, you need to gather the necessary information first. This information is most effective if it is available from a known, trusted source. That trusted source is, in theory, the CMDB, and this is the ultimate goal of the CMDB.

Information is the key to all decisions, and information is constructed from building blocks of raw data. This data is encapsulated in the CMDB. To derive information from the data, one must have an application in mind. These applications of the CMDB are covered in Chapter 9, "Leveraging the CMS," but first, we must define the CMDB and how it is evolving into something we call the CMS. In Chapter 3, "Planning for the CMS," we walk you through all the different steps you need to take to transition from the CMDB to the CMS. The definition of the CMDB is not as straightforward as you might think and certainly not what many prevailing definitions would suggest. This chapter discusses the true definition of a CMDB.

The Birth of the CMDB

As long as there have been complex IT systems (since the 1950s and 1960s), there has been a need for some form of CMDB. Early implementations were likely paper and pencil (or even stored in some person's brain!) because life was so much simpler then. Few called them CMDBs, but this is precisely what they were.

The CMDB term did not come into use until after the 1989 arrival of ITIL. Indeed, the advent of ITIL was necessitated because of the hyperbolic expansion of complexity. IT systems were no longer isolated, simple machines. They had grown to encompass multiple computers across networks and distributed software components. The only way to keep track of the complex nature of these systems was through technology. This technology came in the form of a CMDB, or what most people called an *asset database*.

More notably, these complex systems had become inextricably bound to business execution. Prior to the 1990s, computers were either isolated, well-defined islands that ran the back office of the business or they were intellectual tools. By the late 1990s, it became clear that the state of IT had a direct impact on the state of the business, and the state of IT was not good. The 1990s saw the rise of distributed computing as the central nervous system of the business, and a mechanism was needed to bring discipline to the operation of this wily beast.

Along came ITIL. Although it grew across Europe in the late 1990s, its worldwide appeal finally exploded on the scene in 2004 when North American adoptions reached critical mass. The chart in Figure 2.1 shows how ITIL hit its inflection point in 2004, followed by the latent impact on membership in the U.S. branch of the IT Service Management Forum (itSMF),[1] the international organization dedicated to the development and promotion of ITIL and IT service management.

Because CMDB is threaded throughout the ITIL literature, the CMDB phenomenon has grown along with ITIL, albeit more slowly than the general adoption of ITIL. This relationship also consolidated the function around common terminology, so CMDB is now the prevailing term for the trusted data source.

1. The growth figures for itSMF membership were taken from a presentation by David Cannon, president of itSMF USA. Mr. Cannon presented these growth figures at the April 10, 2008, meeting of the National Capitol local interest group serving the Washington, D.C., area.

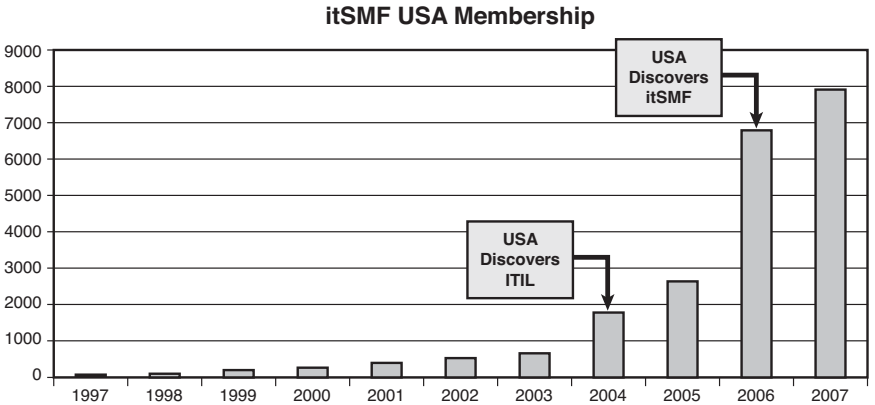

FIGURE 2.1 ITIL growth

The Configuration Management process in ITIL was inspired by the early work by the U.S. Department of Defense and by the newer Software Configuration Management (SCM) process used in software engineering. Because of this heritage, you will notice many similarities across the two process models. Among these are the concept of a Configuration Item (CI) and a CMDB. In SCM, a type of CMDB is called the Definitive Software Library (DSL). It is not a full CMDB in the context of modern ITIL taxonomy, but it is a good first step toward true CMDB. As you will see in Chapter 4, "The Federated CMS Architecture," the DSL has been preserved to play a role in a broader federated CMDB, employing multiple CMDBs in what is now called a Configuration Management System (CMS) in the language of ITIL v3.

Configuration Items

Each element in the IT environment is an individual entity requiring accurate capture of its attributes. The representations of these entities in the CMDB are *Configuration Items* (CIs). A CI is a software model that contains the attributes of the represented entity. In databases, this is described in the schema. Each entity consists of several attributes of specific data types (for example, string and integer). Each instance of the entity is its own CI (for example, 200 identical servers equals 200 CIs, all with the same schema, but as independent instances with some unique attribute settings). A CI can be physical—that is, real and tangible (for example, hardware and software code)—or it can be logical abstractions of these (for example, business processes and distributed applications).

If we examine 300 Windows servers and 20 Cisco routers, this subset of the overall environment represents at least 320 CIs. We say "at least 320" because it is possible, indeed likely, that each device is built from other devices. For example, each Cisco router may consist of a chassis and multiple cards within the chassis. It is possible to treat the entire router as one CI, but it makes more sense to also break down the CIs to more granular levels. We explain why later in the book, but the short reason is that it enables more flexible abstraction models to be constructed and manipulated.

The concept of a CI must be extensible. Some believe a CI is the most atomic level only, but a CI is also a more complex assembly of lower-level CIs. In the server example, a physical server has special attributes. A virtual server also has special attributes unique to that virtual instance, even though it is resident on the physical server. This relationship between the two means they will share certain attributes, but also contain other attributes unique to each perspective.

Applications, business services, and customer organizations are also CIs (see Figure 2.2) at even higher levels of the CI hierarchy. Similar relationships link these various layers to reflect reality. To accurately capture and maintain the atomic details as well as the relationships, the CMS and the CMDBs are built upon traditional relational databases and object-oriented models, as well as a myriad of other formats that include text files, Simple Network Management Protocol Management Information Bases (SNMP MIBs), and software Application Programming Interfaces (APIs). The object-oriented technologies are especially important in the continued development of Configuration Management information. We cover these technologies in more detail in Chapter 4.

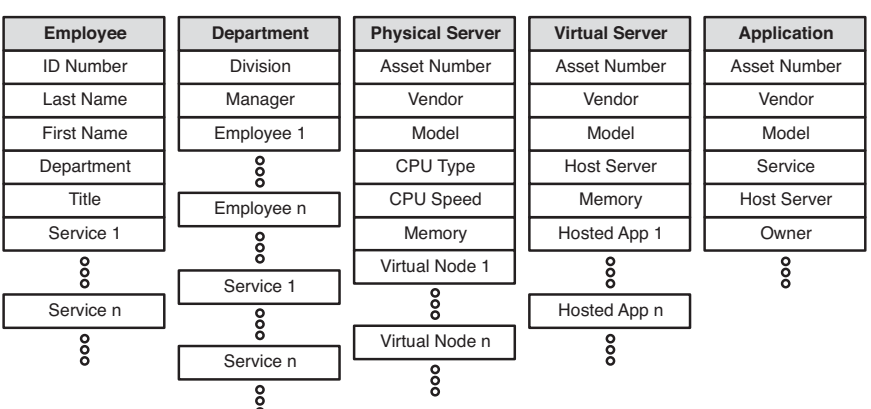

FIGURE 2.2 Example Configuration Items

Each CI has several attributes associated with it. Some are compound abstractions that refer to a number of other CIs (the Department CI consists of several Employee CIs). By simply viewing the CIs as individual items, it is difficult to envision how these various CIs interact to form something more useful. We need to link the correct objects and attributes by creating the appropriate relationships, as shown in Figure 2.3 by the dark black lines.

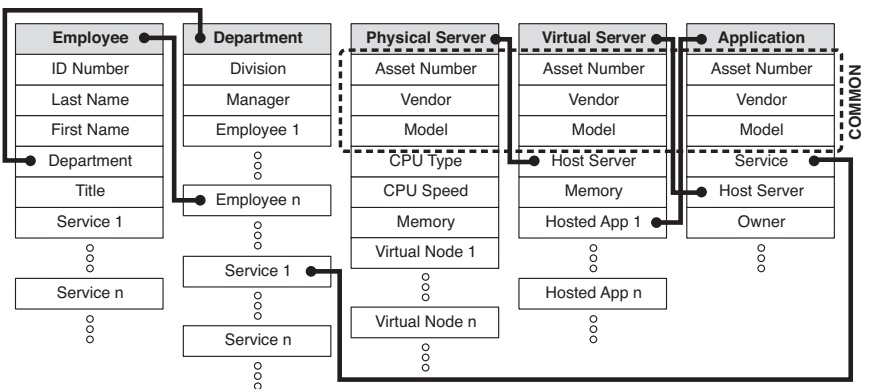

FIGURE 2.3 Relationships among the CIs

Here the CIs are linked to expand on the appropriate attributes, and it becomes easier to navigate the map of CIs to each other and to their impact upon the overall service. The links can be established in many ways, such as relational database table references and Java class pointers. When we talk relationships in the CMDB, the links are best implemented with XML. We explain these relationship links in more detail in Chapter 4 and Chapter 6, "Integration—There's No Way Around It!"

In this example, you can also see how there is common data across some CIs. Good database administrators try to normalize the data to eliminate the possibility of data overlap. With distributed objects, the normalization is a bit more difficult. In this particular case, it is possible that these specific attributes might be linked in from an Asset Management database as yet another CI.

You will see many references to relationships in this book—maybe so many that you will find the repetition annoying. We relentlessly preach about relationships for a reason. They are indeed the glue that binds everything together to result in information that is more meaningful. Without adequately capturing relationships, the CMDB crumbles into a worthless disarray.

A CI can be a small morsel of data or it can be a complex, multilayered composite of other CIs and other composites, or it can be any level in

between. This flexibility is essential to a smooth Configuration Management process. Do not limit your perspective of a CI to just the infrastructure devices. Such a myopic perspective makes the system of CMDBs inflexible, fragile, and even more complex than the object-based model. This is one aspect of the discipline (or lack thereof) noted in Chapter 1, "The Need for Process Discipline." The future of the operation can be placed in jeopardy if these deleterious factors exist.

CI Candidates

Many elements of the IT and business environment qualify as CIs as we describe in Chapter 5, "CMS Deployment Strategy." Ideally, every identifiable unit of importance can be captured in the CMDB, but realistically, you will be driven by your business service needs and bound by priorities, staffing, and technology capabilities. Exactly which CIs you incorporate and in which order will be determined by each of these factors. Existing technologies offer plenty of information that can help. In Chapter 4, we cover how this data is leveraged. Configuration items fall into many domains, which are discussed in the following sections and in yet more detail in Chapter 5.

Infrastructure CIs

When people think of a CMDB, they usually limit their perspective to the infrastructure elements (servers, routers, storage, and so on). This is a myopic view, but the rationale behind this mode of thinking is understandable. Throughout much of the history of IT, what is now called the CMDB was known by various other terms, the most common being an asset database. This infrastructure-centric view persists today, even though the CMDB contains far more than infrastructure elements.

Such a database was used to maintain a record of the individual hardware assets owned by the enterprise. Advanced organizations expanded the asset database to include software also. An asset database represents a limited subset of a CMDB, but an important one. It is often the initial phase of a CMDB journey because of this historical view (and the fact that much of the data has already been collected).

Infrastructure is most closely associated with the hardware devices in the environment. For each device, many attributes are identified, collected, and maintained. Figure 2.4 shows a few common attributes of a CI representing a server.

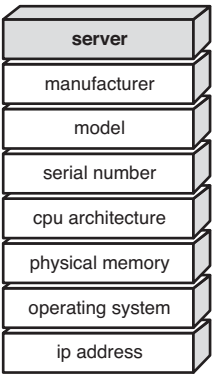

FIGURE 2.4 A few server CI attributes

Even in this simple server CI, you can see some attributes that raise additional questions. Attributes such as manufacturer and model are straightforward, but what about CPU architecture and operating system? CPU architecture could be Intel x86, s/390, or SPARC, and each opens up its own unique twist on the basic server definition. The operating system could be Windows Server 2003, Red Hat Enterprise Linux 5, VMware ESX 3.5, or even something like OpenVMS 7.2-1. Each of these attributes requires more detail, detail that may be unique to Windows, but totally irrelevant to Linux, for example.

Where such extensions are needed, object-oriented (OO) technologies can link the relevant extended attributes. This raises some interesting debates about precisely what constitutes infrastructure. Traditional examples of infrastructure are network devices (such as hubs and switches), servers, storage (including disk drives and SAN hardware), and mainframes. This is what we typically consider to be hardware.

As we continue to embed more sophisticated intelligence into the hardware, the line between infrastructure and applications becomes blurred. Virtually every single piece of today's hardware is a complex mixture of genuine hardware (for example, silicon, steel, and plastic) and software (for example, operating system, BIOS, microcode, and embedded web server). For the sake of the CMDB discussion, we consider infrastructure to be the self-contained package of hardware and its embedded software. Infrastructure is hardware *and* software.

Software infrastructure differs from applications, as we describe in the next section. This is sometimes called system software, but software infrastructure extends beyond traditional system software (such as Linux and VMware) and into those elements that support applications (such as Tibco and JBoss). Although they are software, we recommend that infrastructure should include these software elements as well as traditional hardware. Infrastructure is no longer limited to those tangible hardware components that require electrons. The basic intelligence that manipulates those electrons also counts as infrastructure.

The structural and behavioral aspects of certain attributes will morph over time. Virtual servers are a good example. Whereas a server once supported a single operating system, it can now support many.[2] The operating system attribute must now be abstracted to link to multiple possible virtual operating system instances (virtual machines [VM]) on a particular physical server. To complicate matters even more, that relationship between physical to virtual server can now be highly dynamic (for example, VMware's VMotion software easily shuffles a VM from one physical server to another).

Your CMDB plans, architecture, technology, and operations must be flexible enough to address these frequently shifting requirements. This is a major reason why object-oriented technologies are becoming more popular for the CMDB. An OO approach enables you to adapt data structures to changing demands. IT systems are irreversibly becoming more dynamic. The so-called *cloud computing*[3] that is gaining popularity marks an acceleration of dynamic systems, not a plateau.

Application CIs

Applications are the software elements that are directly touched and felt by end users. Examples include Microsoft Exchange, Siebel, and a custom shipment-tracking system. Products like SAP, Microsoft Internet Explorer,

2. Mainframes and some minicomputers have long supported virtual machines, but the concept's popularity has exploded in the distributed server market, thanks mainly to VMware.

3. Cloud computing is a new concept receiving the usual hyperbolic media attention as this book is being written. Its advocates tout it as the next form of delivering IT to the business. The most notable characteristic of *the cloud* is outsourced infrastructure and application services, sometimes including the entire IT operations organization. Cloud computing has some merits, but it is too early to declare it a success or failure. Wikipedia's page on cloud computing is even somewhat vague and not yet fortified by the usual verifiable references and citations (http://en.wikipedia.org/wiki/Cloud_computing).

and Adobe Flash Player can also be seen as applications, but here we are getting more into the domain of software infrastructure, as we just pointed out in the section on infrastructure CIs. What you deem an application and what is software infrastructure is up to you, but you should be consistent within the context of the CMDB.

As you build out your application-level models, you will find that many applications are abstractions built upon other applications. The gray line between applications and software infrastructure is where these situations will occur. Some applications that are used by some end users (such as a database front-end) may become software infrastructure for another, more complex application (for example, a shipping application built on top of SAP and using this database).

The value of clarifying the demarcation between applications and software infrastructure comes when using the CMDB for more sophisticated purposes. We cover these purposes in more detail in Chapter 9, but we present a simple example here—one that is deeply desired, but also extremely difficult to achieve.

Consider the exercise of identifying the root cause of an application performance anomaly. The application consists of many software and hardware components, so the first thing we need to do is understand the structure of the application's relationships to these elements (see Figure 2.5). The hierarchy of the relationship map will be multiple tiers for the typical modern business application.

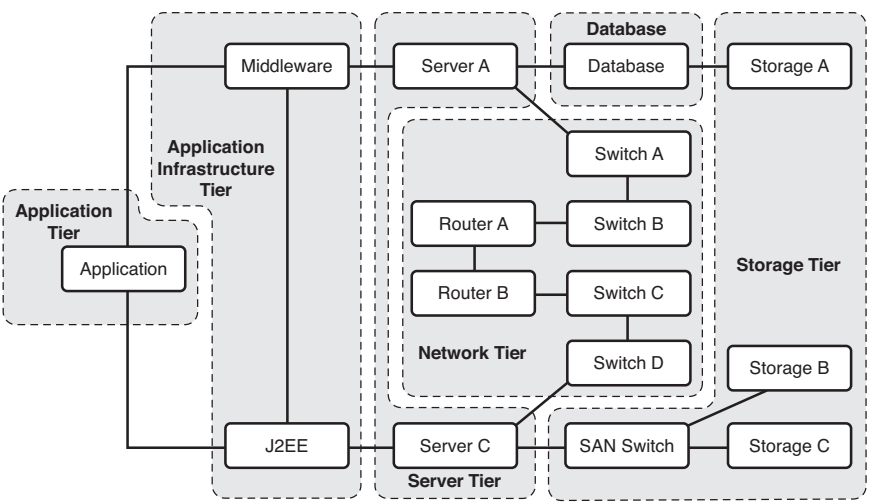

FIGURE 2.5 Example of tiered application structural relationships

To determine the root cause, this structure is navigated, and each component's impact on the overall application behavior is assessed. This assessment first capitalizes on the relationships that bind the various components together, as they form the paths followed in the navigation. They give us visibility to potential cause-effect mappings and significantly narrow the field of possibilities.

The performance, availability, and configuration state of each of these components can then be analyzed. The mapping significantly reduces the set of possible root causes by including only those elements that truly matter to the application in question. By narrowing the scope, you can then attack the problem more effectively.

Of course, it would be wonderful if you could automate this analysis to the very end, but this remains a dream for now. This is where the difficulty arises with application analysis, but such is the topic for another book.

In Chapter 9, we offer more guidance on how the CMDB/CMS works in conjunction with the Change Management process. In that chapter, we also show how changes can point to many of the root causes of a myriad of problems, including the elusive application performance problem. It all begins with having accurate configuration data or—as we like to call it—the truth. You must also not forget that there is a Continual Service Improvement component to ITIL v3; some of which we describe in Chapter 8, "Continual Improvement for the CMS."

Known Error CIs

There is a condition in Incident and Problem Management called a *known error*. A known error is an incident or problem whose diagnosis is complete, but whose final resolution is either delayed or implemented with a temporary fix. As incidents arise with the known error as a root cause, operations staff should be able to quickly identify the known errors and take appropriate action. Actions include proper notification of affected users and possibly implementing work-around solutions. As the operation gains experience, the known errors are tied to resolutions. This forms a cause-effect linkage that accelerates resolution. The error is detected, and the resolution is immediately found and presented. The responders therefore have a map to guide them to speedy service restoration.

One of the more important management aspects of known errors is the need to capture them and maintain them throughout their lifecycles, including through end-of-life. To enable this, a known errors database is maintained. Many of the common Service Desk products now include some form of known errors database. Service Desk staff developed a need for this database for the purposes we mention here, as they are the front lines of the IT

organization. Vendors of Service Desk automation software responded to this need with full inclusion of a known errors database or supplying it as an optional module.

Naturally, the known errors database represents yet another family of CIs for the CMDB. The value of this class of CIs helps streamline many of the daily functions such as incident analysis and long-term pattern analysis for Problem Management, capacity planning, and continuous improvement.

The known errors are related to infrastructure, application, and other classes of CIs. These relationships need to be captured and mapped, as the known errors are attributes of these "host" CIs. The known error attributes are subject to change, and there can be multiple known errors for a single host CI, which could be dependent, for example, on their particular environment, configuration, or architecture, so the object model must support such a zero-to-many set of similar relationship attributes.

Business Process CIs

If applications are abstractions of infrastructure, the next step above applications is the business service level. Like applications, there can be multiple layers of business services (for example, a web-based product ordering service is a component of the broader supply chain automation service). At the highest level of this structure are the business processes. Chapter 5 and Chapter 7, "The Future of the CMS," have figures that demonstrate this notion of business services sitting atop the remaining IT structure.

Business process modeling is a functional domain unto itself within the IT community. It is one of the tightest integration points between IT and the business and is increasingly owned and controlled by the business side rather than the IT side. Unfortunately, the CMDB and CMS vendors do not yet offer complete solutions that automate the discovery of the services without significant human intervention. The current state is still very much silo oriented, but it is changing.

You should work with the business community and the Enterprise Architecture organization to incorporate business process information into the CMDB/CMS. Stay up to date with the standards being developed by the Business Modeling and Integration Domain Task Force so you will be prepared to input your work more easily into the vendor tools when those tools do come along. All parties will benefit. Newer developments are making this much easier now because standard object modeling and integration technologies are gaining acceptance.

Standards have been under development for over a decade. Standards took a positive turn in 2005 when the Business Process Modeling Initiative (BPMI) merged with the Object Management Group (OMG) to form the Business

Modeling and Integration Domain Task Force (BMI).[4] BMI and OMG are responsible for some standards, OASIS for others, and the W3C for still others.[5] The flexibility offered by XML allows for many derivatives to be developed. That's the good part of XML-based standards. The downside is that it allows for many derivatives to be developed. You can easily become overwhelmed by the plethora of standards available.

As you pursue the CMDB, your role in the selection of these specific standards will be limited. You should leave the painful selection to those in the Enterprise Architecture team (who have this responsibility anyway), but we highly recommend you collaborate with that team to help determine the optimum path to linking the business process standards with your CMDB. Chapter 4 provides the basis for your discussions about a federated structure with the architecture groups.

Much discussion is taking place about Business Service Management (BSM), the higher-level aspects of IT that produce and manage technology in the eyes of the business, not so much in the eyes of technologists. The business processes and the business services that feed them are the linkage between core IT (infrastructure and applications) and actual business value. The relationships between and within these tiers enable us to determine business impact of a server failure, for example. Without these relationships, assessing the business impact is mere guesswork—and very often wrong. Accurately understanding business impact via the CMDB and its use cases is absolutely essential if you want to position the IT organization as a valued business partner. Just as it is in all other functions of IT, the CMDB is a fundamental supporting pillar of BSM.

Business processes are an often-overlooked element of the CMDB because of the aforementioned misconception that the CMDB is all about infrastructure. The omission of business processes further exacerbates the "us and them" division between IT and business. Inclusion of business processes ensures more business relevance to the CMDB and enables use cases to have

4. The Business Modeling and Integration Domain Task Force web site (http://bmi.omg.org/) contains many details about the group's standards efforts.

5. Standards are a veritable alphabet soup that confuses actual progress. The BMI initiative owns BPMN (Business Process Modeling Notation), OASIS (Organization for the Advancement of Structured Information Standards) owns BPEL (Business Process Execution Language), and W3C (World Wide Web Consortium) owns WS-CDL (Web Services Choreography Description Language). There are still others in this ever-growing family of standards. Collectively, they are all part of the greater Service-Oriented Architecture (SOA) movement to componentized business processes and the software used to automate business processes.

more value to business service management efforts. The result will help change the language from "us and them" to just "us" in a unified journey toward business execution excellence.

Human CIs

Human beings are the most important elements of any business. IT services are no different. You and your colleagues, your end users, your reporting structure (in both directions), vendors, partners, and many, many others all contribute to your technology and business services. How each of these people fits into the overall picture is a requirement of any good CMDB. "People" are identified, tracked, used, and managed according to each individual's contributions and responsibilities. Here are just a few attributes of people that are useful:

- **Identity.** Each person needs a unique identity, sometimes called a digital identity. This identity is a complex data structure far beyond the person's name. Among the many attributes tied to the human CI is a unique key, a means to identify that John Smith is the John Smith you think he is.

  Software tools used to manage IT services are notoriously disjointed with respect to identities. Each usually has its own identities, requiring tedious replication of data across tools, data that nearly always falls out of sync. Common identity management technologies are finally showing promise to unify identities across tools, but the various tool vendors must support them. Happily, there is increasing momentum to support common mechanisms like Active Directory and Kerberos.

- **Authorization.** After identity is established, the next step is to assign authority to take certain actions. These authorizations can be for specific software or built into managed infrastructure. Here also, the multitude of mechanisms and individual points of management greatly complicates this situation.

  The identity and authorization must be intertwined. Companies are using identity management products that work in conjunction with other products using AAA protocols[6] and related technologies to

6. AAA stands for authentication, authorization, and accounting, a generalization for the interrelated technologies for maintaining secure systems. See the Wikipedia page, http://en.wikipedia.org/wiki/AAA_protocol, for more information on AAA.

understand, manage, and enforce access controls, control privileges, and other authorization policies.

When a management software function is executed, it must ensure that the action is allowed (for example, "Can I change the OSPF parameters on this router?" or "Can she view the schema of the HR database?"). Proprietary and isolated mechanisms can support these actions, but managing the dizzying array of identity-authorization mappings without automation technologies is intractable.

Here lies the solution to this dilemma and one that is already starting to take root in IT organizations. Configuration and Change Management (CCM) products highlight a good example that is bringing some sanity to the tedious and error-prone tasks of manual configuration change execution. The tool manages a common AAA mechanism for all covered elements and then the tool itself is executing the changes. The disparate access controls are managed centrally in this common tool.

Logically, each CCM tool has a strong CMDB at its heart, so the tool can sufficiently understand the configuration of what it is acting upon and various policies regarding these actions. The result is a common platform for normalizing task execution that is faster and more trustworthy, but there are yet other benefits to the CMDB effort.

CCM tools include a CMDB, so they are valuable as elements of an overall CMDB architecture. By virtue of their basic function, they almost always do their own discovery of their covered domain and usually more effectively than other discovery tools in their domain (there are many). This means the data is highly accurate. As you build out your wider CMS, these tools are effective CMDBs for their individual domains.

- **Roles.** Individuals are the soldiers in the service management war, but these individuals can be classified to guide common tasks across a related team of individuals. You have many of these teams, and members can belong to multiple teams. The teams may not even be formal; they just have common goals, so we'll call them roles instead.

  By assigning authorizations to roles, the tools that drive actions can be simplified. Instead of managing authorizations for each individual, you can just manage them for roles. The system then cross-checks roles with people who are assigned to those roles.

> Roles institute an additional layer of structure into the human CIs, but the additional layer is well worth the minimal overhead because of the operational simplicity we just mentioned. As with the known errors CIs and many others, the model requires flexibility to support many roles per person. Here the relationship will be at least one role, but probably more.

The human CIs are certainly included in the bigger CMDB structure (CMS, to be more precise). You need not and indeed should not create this CMDB from scratch. Leverage what already exists, a data source full of excellent data on your people. It's called the HR database, and it is most often embedded within software tools used for HR functions, such as Peoplesoft.

In Figure 2.3, we showed a CI called Employee. The populated Employee CIs and their attributes can all be extracted from the HR database, requiring little additional embellishment for CMDB purposes. You won't need a lot of the detail existing in the HR databases (they won't give it to you anyway, nor should they!), although basic identity information (for example, employee number, name, department number, title, reporting structure, and roles—if they exist there) will be very handy and more available.

Document CIs

Finally, we present another class of CI that is often overlooked, despite repeated references in the formal ITIL positions on the CMDB and CMS. That class is documentation. Together with known errors and other sources, they embody what some call *Knowledge Management*. We view this as a valid categorization.

Documents are different data types than those normally maintained in a CMDB or relational database. They are files, such as PDF, Excel, MS-Word, and so on. The document CI has as a core attribute, the path name or (preferably) the URL that points to the document file. The host CI (infrastructure, application, and so on) then points to the document CI as appropriate.

If you need to see the installation instructions for a Dell PowerEdge™ M600 server, the attribute for the installation guide in the document CI can be set to the specific PDF document on Dell's web site, or you can have your own internal copy (likely a better idea). The UML diagram for the Order to Cash business process can be a PDF link included as an attribute in the business process CI for Order to Cash.

How Much Data Does a CI Need?

Raw CIs (the actual devices, software elements, and other real-world entities, not the data stores) contain far more data than you will need in the CMDB. You must trim this overwhelming data set but still maintain visibility into enough to be meaningful and useful.

Consider the example of an SNMP MIB[7] for a small branch office router. This device tracks thousands of individual attributes in its internal memory! Try performing an SNMP Walk operation on one, and you will see firsthand the massive set of data that can be gathered. Clearly, you don't need all of this. Trying to manage such a voluminous data set would be unwieldy, even for a few devices.

Where then do you draw the line to delineate what is tracked and what is not? A good starting point to this answer lies in the DMTF's CIM standard.[8] CIM is an impressive body of work, developed by many brilliant people over several years. It is an elegant and comprehensive object model that is finally getting some traction after a slow start. We go into more detail on the DMTF and the CIM specification in Chapter 6.

The final determination of the CI data's richness lies in the use cases for the data. Some use cases (for example, fault analysis) require very little detail. Some such as data center automation need much deeper detail. You will struggle to reach the appropriate level of detail on your own, so this is why we recommend starting with CIM. Part of the brilliance of CIM is the work they've already put into assessing the right level of detail.

A Brief Explanation of CMDB Federation

Federation, described in much greater detail in Chapter 4, is a new approach to maintain accurate configuration information about the IT environment,

7. The Simple Network Management Protocol (SNMP) was originally defined in the IETF document RFC 1067 (http://www.ietf.org/rfc/rfc1067.txt?number=1067) in August 1988. It uses a Management Information Base (MIB) structure to define the attributes represented in a networked device. It has been expanded to be applicable to servers, applications, and many other configuration items, but it continues to be seen as largely related to the network.

8. The Distributed Management Task Force (DMTF) produced and maintains the definition of common data specifications called the Common Information Model (CIM). The first version of CIM was released in 1996, so it has benefited from years of refinement. XML, UML, and MOF files for CIM Version 2.18.1 are available at http://www.dmtf.org/standards/cim/cim_schema_v2181.

although the concepts behind federation date back to the 1980s. CMDB federation was proposed soon after the CMDB first gained traction on the coattails of ITIL, but formal federated CMDB acceptance has been elusive until recently. The new breakthrough is a definition for integrating data sources called the CMDB Federation standard. The name is not the most clever title, but its publication marks an important step toward a successful CMS. Developed by an ad hoc group of vendors called the CMDB Federation Working Group (CMDBf), the specification has now been handed off to the DMTF for further development. We explain the concepts and some details of the CMDBf specification in Chapter 6.

The federated model enables more flexibility than a monolithic database model. The standalone model was originally the accepted model and assumes all data is collected in a single database. This is impractical for a CMDB, as it is easily corrupted by synchronization issues with data that is usually geographically diverse and rapidly changing. The monolithic model places a heavy burden on the database, its related infrastructure, and the related administrators. Sheer resource utilization forces lengthy update intervals, which hamper timely updates to the database. Data quickly becomes stale in such a scenario, and stale data is more harmful than no data at all.

Federation enables a "divide and conquer" approach to the CMDB architecture, distributing the load and leveraging the optimum tools for each technology domain and locale. By using the best tool for a specific domain, the CMDB contents are better tailored to a domain's unique needs and can be updated more frequently. Both result in higher accuracy.

To overcome CI differences and still capture the relevant data for the CMDB, the CI definition extends beyond a simple single-table database structure. Relational databases can be used to organize and capture these CIs in a more complete form across multiple linked tables, although the interchange and federation of data is best addressed by object-oriented technologies such as object repositories and XML. If a relational database is used for the remote linking, federating the data is much more cumbersome. Many will dispute this assertion, but the explosive success of XML is evidence enough that it is the superior mechanism for linking and exchanging remote data. This fact does not diminish any RDBMS for what it does best. The RDBMS is indeed integral to the many pockets of the CMDB. It just doesn't work well for the complex linking of the distributed and seemingly disparate data sources. For this, XML works best in partnership with the databases.

In federation, the data is spread across multiple CMDBs, more accurately referred to as Management Data Repositories (MDR), illustrated with the simplified example in Figure 2.6. There is no single CMDB. Instead, the aggregate "CMDB" is an abstraction built upon the contents of the MDRs, and

metadata models dictate the assembly of relevant MDR data based upon needs. The MDRs may reside in different systems and data storage formats. Federation is at the heart of what ITIL v3 now refers to as the Configuration Management System (CMS).

If the CIs from Figure 2.3 are reflected in a federated structure, you would see something similar to Figure 2.6. Here, the MDRs are aligned with domains as we advise in the CI candidate descriptions.

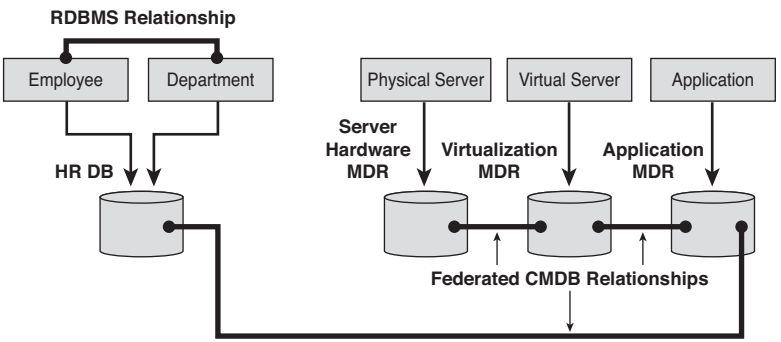

FIGURE 2.6 CMS relationships across MDRs

Note how the HR database is represented as one of the MDRs. Two of the CI categories are embodied within the same database, with a relationship inherent inside the database. This relationship is a branch of the federation structure, even though it appears to be isolated. This illusion of isolation results from viewing federation too myopically.

The HR database is maintained separately from most other MDRs, but its inclusion emphasizes a growing practice in the whole CMDB arena, that of the *foreign CMDB*. A foreign CMDB is an element (an MDR, if you will) that is not under direct control of the IT organization or those responsible for the CMDB architecture, design, and operation. It is a data source rich in information that can be used, but it is available only in a "read-only" mode. There is nothing wrong with this. In fact, it is indicative of the future, as third parties will assume more responsibility for certain domains and therefore the MDRs and CMDBs of those domains. Your overall CMDB needs this data, so proper integration and federation are needed beyond just the HR database.

CIs as the CMDB "DNA"

We can propose a simple analogy to highlight the power of CI reuse and federation from the CMDB. Our example is DNA, the basic building block of all organic life forms (at least those on Earth!) and how the basic genetic elements are abstracted to eventually form complex organisms.

Every living organism is built from the same four building blocks. Geneticists call these nucleotides and represent them by their letters: A (adenine), T (thymine), C (cytosine), and G (guanine). The reason each plant or animal is different is because these nucleotides are assembled differently. Arrange them one way, you get a Bengal tiger; arrange them another way, and you get Tiger Woods.

If we consider the genetic CMDB to contain only four CIs, being the four nucleotides, we can turn this raw data into extremely sophisticated organisms using abstraction models known as codons, genes, chromosomes, and organisms. Everything uses the exact same CIs. The secret to each unique finished organism lies in the abstractions.

Abstractions are themselves CIs with their own special attributes, so we need a CMDB mechanism to store and retrieve them. It is best to store the abstractions in separate CMDBs from the one containing the four nucleotides. This is where federation brings value to the overall CMDB structure. As we explain later and in Chapter 4, the federated model allows flexible references to the CMDB data to produce actionable information. In this case, the abstractions point to the nucleotide CMDB to obtain the correct low-level details, and all of this is assembled in a usable form based on the structure defined in the codon, gene, chromosome, and organism CMDBs.

Figure 2.7 depicts the genetic hierarchy followed as we navigate from the top-level organism all the way down to the nucleotides. Obviously, genetics is a far more complex field than is shown in this example, but it does illustrate how the various abstractions interact and relate. Note how each abstraction layer uses its own CMDB to hold the models that define the structure of the abstraction and the references to the layers below. An organism model defines the right mix of chromosomes, each chromosome model defines the right mix of genes, each gene model defines the right mix of what geneticists call codons, and each of the 64 codon combinations is built from a unique structure of nucleotides. If there was a need to further deconstruct the nucleotides, one could break down each nucleotide to its molecules, atoms, and subatomic particles through the use of additional abstraction layers.

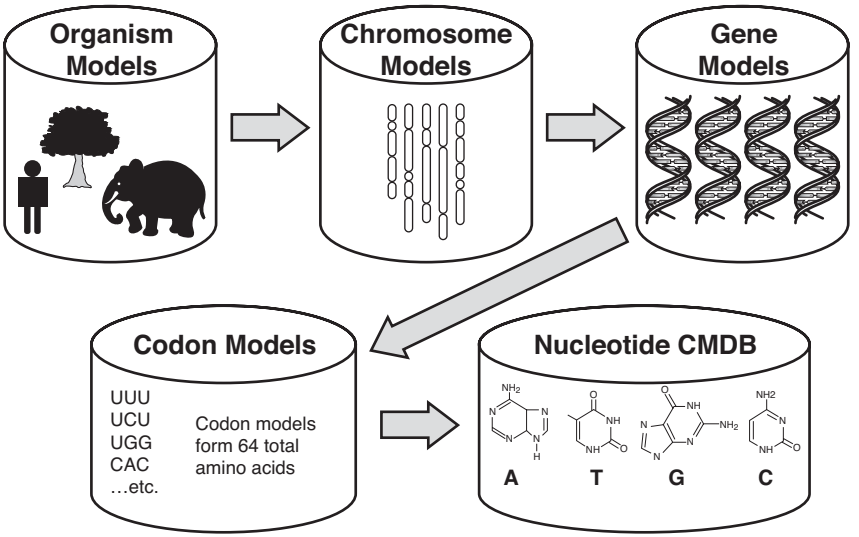

FIGURE 2.7 Genetics example of federated CIs

It would be wasteful to replicate copies of data from each lower level of this structure. By the time we get to the chromosome level, the number of nucleotide combinations is astronomical. The use of abstraction tiers greatly simplifies this situation. For example, the chromosome level only needs to reference models of individual gene strands, not the entire structure of each. The raw data remains accurate and up to date, and we have the flexibility of creating new abstractions merely by rearranging the most practical model. Everything below that model remains unchanged.

Every good CMDB follows a similar pattern of hierarchical references as the models and data are navigated for a particular use case. The idea is similar to database normalization, where the goal is to optimize flexible data construction with a minimum of data replication and maximum performance. Object federation in a CMDB structure differs from database normalization in that it can link a wider variety of data sources in many physical places (if needed). Object models can also contain behavioral information such as rules. Good object models go far beyond mere structural information.

Data from the level below is not duplicated in each model. This would cause severe data overload and synchronization issues, especially at the higher levels. Each abstraction merely references the appropriate elements of the layer below. For example, there is no need to define the molecular structure of every codon when simply referring to the A, T, C, and G nucleotides and their assemblies are sufficient. A behavioral aspect of the genetic CMDB

that can be encoded in the model further simplifies the nucleotide references. Because the only possible nucleotide combinations in the genetic code can be A to T and C to G, there are not four possibilities, but only two (an A-T pair and a C-G pair—or a one and a zero, if you wish).

It is ironic that life forms can be deconstructed down to the same binary code used in computer science! This is one reason we chose the genetic example. Clearly the genetic code is hideously complex, but building the layered models greatly simplifies the navigation of the system and optimizes performance, data reuse, and reliability. IT services are no different. If a federated model can allow us to tackle something as convoluted as the genome, IT services should be much easier. Nobody can credibly state that a federated CMDB is easy, but we often overcomplicate an already complex mission by our failure to leverage best practices and novel solutions. CMDB federation is such a novel solution, and ITIL (or similar process models) can offer the best practices.

Reconciliation

In the existing world of CMDB, reconciliation is considered to be a necessity and indeed it is...today. *Reconciliation* is the synchronization of two or more matching database segments to ensure consistency across them. Data stores that should be identical copies are compared. If any differences exist, the reconciliation engine attempts to correct this inconsistency. Most existing CMDBs are isolated, and integration involves a periodic upload or download of bulk data. It is not federation. In this "unfederated" model, data synchronization decays over time. The copies drift apart and become inaccurate. Reconciliation attempts to sync the CMDBs, usually in a brute-force manner that scans all CMDBs and makes corrections as the reconciliation engine finds discrepancies.

Reconciliation is driven by policies that determine who "wins" when a conflict arises. Some may be by majority vote (for example, three of the four CMDBs agree, so the fourth is forced into agreement), some by pre-establishing the winner (for example, the network CMDB is the trusted source for the network), and others can get more complicated. Full reconciliation can be automated to correct the discrepancy, or it can merely inform a CMDB manager to take action to reconcile. Initial phases will be the latter, but those that are trustworthy or become annoyingly repetitive can be automated.

Figure 2.8 shows a simplified example of three CMDB instances where reconciliation is needed. This VMware virtual server is identical across all

instances except for the middle one's physical host. This is a bit like that *Sesame Street* game "one of these things is not like the other."

Europa	
mfgr	VMware
model	ESX
version	2.5.5
host	Saturn
memory	4 GB

Europa	
mfgr	VMware
model	ESX
version	2.5.5
host	Jupiter
memory	4 GB

Europa	
mfgr	VMware
model	ESX
version	2.5.5
host	Saturn
memory	4 GB

FIGURE 2.8 Three CMDBs needing reconciliation

The reconciliation engine scans these three and recognizes this discrepancy. It can either notify someone of the issue or it can correct it to match the other two. If automated action is taken, you must be absolutely certain that the results will be as desired. In this situation, one might think that the "majority vote" rule would apply, but that middle instance might be the trusted source extracted directly from VMware. Because it reflects the truth, it has precedence over the other two, and they, not the middle one, must be corrected. As systems become more dynamic (for example, virtualization), situations like this will become more common. By the way, Europa is the sixth moon of Jupiter, so of course the middle instance is right!

No matter how you view reconciliation, all agree that it is difficult to get it right and even harder to get it properly automated. From firsthand experience, we can tell you that there will be many weeks and months spent tweaking your reconciliation rules to try and automate the reconciliation process. The ideal CMS would eliminate reconciliation because it is too painful (our VMware example is only the tip of the iceberg). As the CMDB gets ever closer to the CMS ideal, we will approach the ideal of eliminating the need for reconciliation. When we finally achieve that ideal is a great mystery, but don't expect to totally shed reconciliation before 2012.

A properly federated CMS will significantly minimize the need for reconciliation because the very nature of federation (as you will see in Chapter 4.) implies that data is always captured and managed only once. There is no need to reconcile CI details because there should never be duplicate copies! The only attributes that are duplicated—and therefore need reconciliation—are those representing the matching criteria for referencing the relevant data. As in RDBMS development, these are the indices (the matching attributes) to point to the right federated data. Even these attributes will eventually graduate beyond a need for reconciliation as better XML-based remote-linking

technologies emerge. Technologies that enable an idealized view of federation are only now beginning to appear. Over the next several years, vendors and end users alike will chip away at the pockets of duplicate data. Someday we will tackle that final step, but be patient and diligent. If there was ever a real-life situation where "slow and steady wins the race" in IT, it is in the pursuit of a true CMS.

The CMS Arrives

In June 2007, the UK Office of Government Commerce (OGC)—the official ITIL caretakers—released ITIL version 3 (ITIL v3). In this book, we do not expand on the many improvements in ITIL v3, but the most significant change to configuration data is the emergence of the CMS.

A CMS is a vast improvement over a simple CMDB for many reasons. A few of the more prominent benefits of CMS are the following:

- CMS implies distributed data in line with, but beyond, the historical notion of a CMDB. The CMS is inherently federated, whereas the CMDB required a dramatic new twist. As you will see, the new concept of a federated CMDB is really a CMS.
- ITIL v3 more clearly articulates practical applications for configuration data and endorses references to and from the broader CMS, rather than direct integration to a CMDB. The distinction between federation and integration is important, and we describe this difference in detail in Chapter 4.
- The genesis of the CMS marks the beginning of the end of the CMDB term. CMDB is a misleading term, but its ubiquity will make it persistent for years. Still, it is much more fruitful to shift many of the CMDB discussions to CMS instead. Even in the CMS description in the ITIL v3 literature, the CMDB term remains, but it is redefined into what is essentially an MDR. We believe that the CMDB Federation Working Group's concept of the MDR will prevail in the long run and that "CMDB" eventually will fade away from the vernacular.
- CMS is richer in its inclusion of applications and services. Prior notions of the CMDB were vague and fragmented beyond simple infrastructure characteristics. Because the CMS is inherently object-oriented, it can make use of behavioral information to make the structural information more accurate and more useful.

- Relationships are at the heart of the CMS. CMDB was often viewed as just a repository of attributes. Linking and federating these attributes were usually limited by technology solutions that were in fact designed according to these misguided, restrictive attribute-centric requirements. CMS is a force to break this cycle by mandating the relationships necessary to make raw data meaningful.

Throughout the remainder of this book, we use the MDR, CMDB, and CMS terms. As much as we dislike the CMDB term, it remains relevant for now. A CMDB in the newer order represents a specialized repository, usually targeted at a specific technology domain. We prefer to call this an MDR, and we will usually do that. Note that the two are nearly interchangeable in the new order of the CMS. It is also important to sometimes refer to CMDB in its historical context. To properly convey the meaning and value of the CMS, we must address, and in many cases overturn, common CMDB perceptions.

Populating the CMS

The CMS is only useful if it is accurate. In fact, the most dangerous situation with a CMS is when the data is wrong, which is why the concept of federation, as described in Chapter 4, is so critical to your design. Decisions are based on the CMS, and when the data is wrong, the decisions are wrong. Instead of the CMS being an enabler for improvement, it can actually cause further deterioration when the data is suspect. CMS population, therefore, is among the most important of all facets of Configuration Management. How you populate your CMS and keep it accurate will directly affect the success or failure of the CMS and—as we pointed out in Chapter 1—the entire IT operation itself.

The CMS is populated in two ways: manually and automatically. The majority of CMDB population so far has been manual. This is one main reason most CMDB initiatives have suffered or failed altogether. Manual population is risky because it is too difficult to maintain its accuracy. By the time population is finished, the contents are already partially obsolete.

To optimize CMDB accuracy, you want to automate as much of the population as possible. We call this *automated discovery*, or *auto-discovery*. For our purposes, we refer to *discovery* as the automated population mechanism.

Discovery is a wonderful innovation for the CMDB, but alas, many CMDB elements cannot be discovered. Therefore, you will inevitably have a mix of both population modes. The following figures show how this mix works.

Figure 2.9 is a simple diagram showing a collection of CIs in a CMS. It is merely illustrative of the point, not an actual CMS.

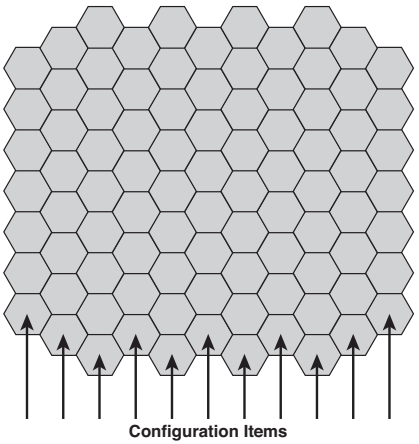

FIGURE 2.9 CIs in a CMS

You need to identify what can be discovered and what must be manually populated. By segmenting the CMS in this way, you can set forth with your plans to build automation technologies and operational tasks to build and maintain the CMS. Figure 2.10 shows how the CMS is divided between the two. Discovery usually gives you the core elements, whereas manual methods are used to supplement this core.

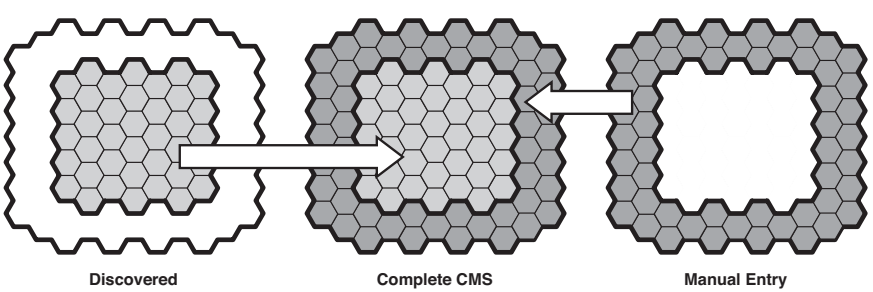

FIGURE 2.10 Two modes of populating the CMS

Both modes can be further broken down into the relevant CMS domains, which are most effectively aligned with the MDRs. Figure 2.11 shows this

additional breakdown. Note how many domains will have both discovered and manual components.

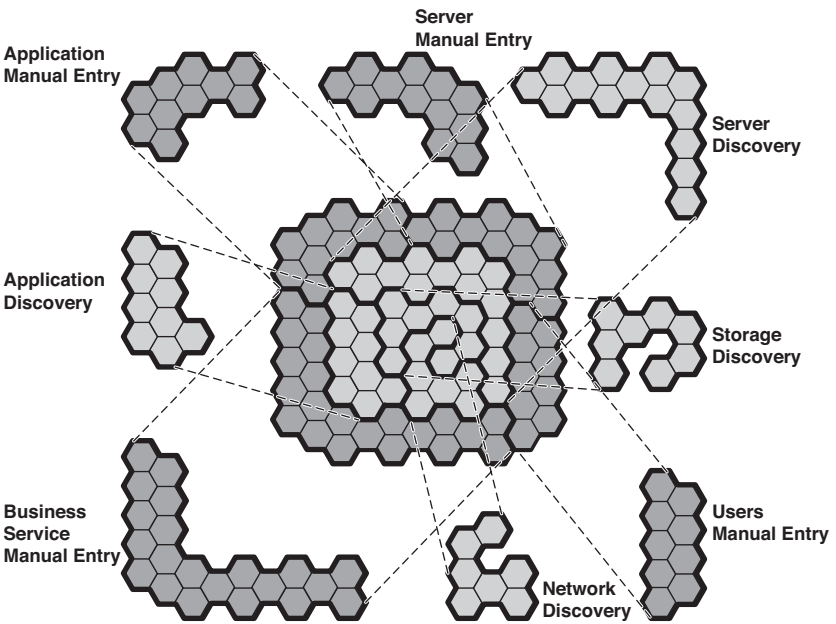

FIGURE 2.11 A further breakdown of CMS population

Domains like the network lend themselves well to discovery because the common instrumentation (for example, SNMP) is pervasive and full of useful data. Others, such as business services, are more heavily manual because of a lack of instrumentation. As we explained earlier, even these domains are improving for discovery. As new technologies emerge to enable discovery, you should capitalize on them to continue building more accuracy into the CMS.

CMS as an Integration Mechanism

Data integration is a longstanding problem with software systems, especially with management software. Proprietary management tool interfaces have long ruled this market. They created a nice development partner ecosystem for the major vendors, but the practice makes it difficult to plug tool A into tool B to attain more comprehensive value from the union of the two. In defense of the major vendors, they appeared on the scene at a time when proprietary integration was the only option. Once established, these interfaces

took on a life of their own and are now so deeply entrenched, they are nearly impossible to replace. That said, however, it is time to change. The integration pain has become so severe that progress toward unified service management is being hampered.

Software tools are gravitating toward functions that act as either data providers (such as discovery) or data consumers (such as analysis tools). These classifications establish the major points of integration, where one must integrate with the other. Some tools act as both providers and consumers (for example, the server agent), consuming data for one purpose (localized analysis on the server, for instance) and providing data for another (in this example, presenting the server's data for analysis in a broader context). Either way, integration can be a quagmire.

Figure 2.12 is a picture of a simple integration challenge. As you can see, many integration points are required, each with its own data model and its own access and exchange protocols. As the number of tools increases, the integration challenge grows exponentially. Clearly, this is not sustainable in a large environment.

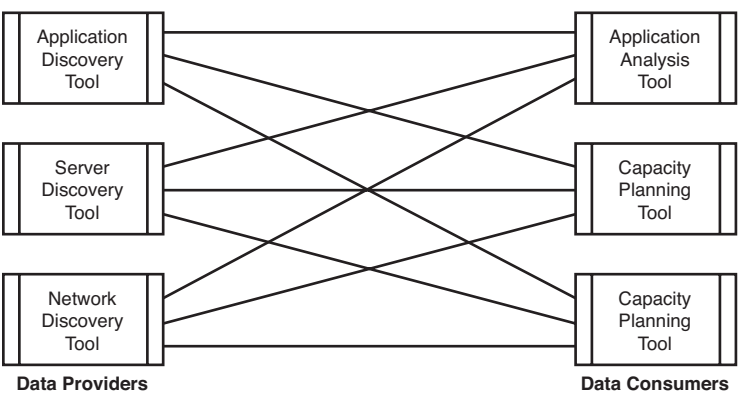

FIGURE 2.12 Proprietary tool integration

The DMTF's CIM and other standards promised to address this situation long ago. Until recently, the standards were held back. Either vendors were unwilling to support the standards effectively or technology limitations prevented a vendor from "shoehorning" a standard into its antiquated technology. New technologies based on true object models and object integration are finally now gaining enough of a groundswell to move the industry toward acceptance of standards-based integration.

Data and messages are the basis of tool integration, with much of the emphasis being on the data. Standards for message exchange are already

accepted practice (for example, SNMP trap or CIM Event), although more work is needed. The data side of integration is the gap, and that gap is filled very nicely by a CMS.

As the CMDB gains popularity, vendors and users alike are recognizing its potential to simplify the integration problem. The very nature of the CMS is grounded in common data formats and interchange specifications, so it seems to be suitable for tool integration. Indeed it is. You will see many more possibilities emerge over the next few years to act in this capacity. Most of the innovations will not even be so explicit, but if two tools are using a common CMS, there is no need to explicitly send a piece of data from one tool to another. The data is already in the right place. Figure 2.13 indicates how the CMS simplifies the integration problem of Figure 2.12.

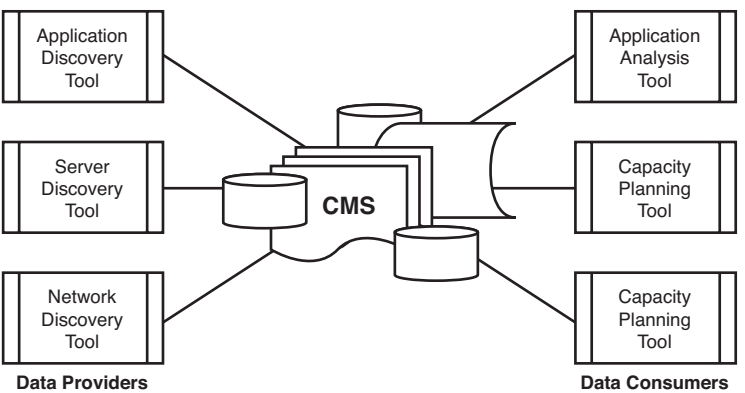

FIGURE 2.13 CMDB as a common tool integration point

Frequently Asked Questions

Question: What are the Citizen CMDB and Core CMDB? You do not mention these here, but many others do.

Answer: The Citizen and Core CMDBs are relics of the monolithic model of the CMDB. The premise is that the core CMDB is the "master" CMDB that holds the unified expanse of all data. The Citizen CMDB "feeds" data into the Core. It implies, in most cases, a two-tier hierarchy. We already mentioned that a true CMS will be multiple levels, built upon a chain of Management Data Repositories and liberal use of relationships. You can view the Citizen as the

MDR at the lower level of a link in the chain and the Core as the higher level of that same link. As you traverse the chain, Cores become Citizens at the next higher link, and the inverse is true as you move lower. It is admittedly rather confusing, but we explain this more suitable structural chain in Chapter 4.

Question: Why did the early asset databases fail to deliver the promise of the CMDB?

Answer: There are two major reasons, among others. First, the asset databases did not generally contain relationships rich enough to support the navigation of dependencies between assets and the services they support. Second, the data itself was difficult to populate and therefore inaccurate. As we mentioned in this chapter, inaccurate data is not just useless—it is harmful!

Question: The CMDB appears to be very similar to a data warehouse. Can we use a data warehouse as a CMDB?

Answer: Technically, the CMDB is similar in many philosophical ways to a data warehouse. Where they differ is in the actual delivery of the technology and how the whole system of data is used within an ITIL-based organization. As a CMS with federation, the parts (the MDRs) are linked differently than a data warehouse, and the tools that produce and consume the data are aligning with the CMS direction. It is certainly conceivable—and indeed expected—that data warehouses will play some role in the CMS, just as traditional relational databases will. The overall system that brings all of these parts together is the CMS.

Question: I would love to include my applications in my CMDB, but it is too difficult. How can I do this?

Answer: The majority of CMDB efforts are now aimed at infrastructure. Applications are indeed more difficult because of the lack of instrumentation to tell us the real structure of the applications. New discovery tools are now available to help. We explore how these tools work in Chapter 4.

Question: How can you advocate extending the CMDB to such high-level concepts as business processes, documents, and the human element when we are still struggling with the basic infrastructure? Isn't this an overly aggressive "boil the ocean" approach that is bound to fail?

Answer: A "big bang" attempt at a CMDB/CMS is almost guaranteed to fail. Like any other ambitious journey, you make progress one step at a time. We do advocate building the higher layers of abstraction, but infrastructure is a necessary foundation that supports these higher layers. Get the lower levels to a reasonable state of maturity before moving "up the stack," and your journey will be easier. A growing number of organizations are now in a position to make this move, as their infrastructure layer is in a more capable state.

Question: I have implemented a CMDB, but my people spend too much time populating and reconciling the data. The overhead is diminishing the value of the CMDB so much that many intended beneficiaries are revolting. How can I minimize this work and save the CMDB effort?

Answer: Automation is the key to simplifying ongoing CMDB/CMS maintenance, including initial mapping and ongoing population updates. The category of automation products is discovery tools. We explain discovery in detail in Chapter 4. A CMS without discovery is doomed, so it is wise to implement discovery as soon as you can.

Question: You imply that ITIL v3 is the end of the CMDB, but it too references a CMDB. What is the truth here?

Answer: On page 68 of the ITIL v3 *Service Transition* book, the CMS diagram contains an "Integrated CMDB" and Physical CMDBs within the mix of parts. Note that the diagram is just an *example* of a CMS, not a definitive description. Also note that this section of the book on Service Asset and Configuration Management (SACM) is rather vague on CMDB. This is intentional, as the ITIL v3 authors share our critical view of the CMDB term. They talk extensively about CIs and their interrelationships, which is all good. The ambiguity about CMDB is also good because it marks the beginning of the end of the CMDB, not the instant death of the term. We continue to foretell that the end is coming, but it will take a while. The CMDB portions of the CMS diagram can be more effectively represented in the view taken by the CMDBf Working Group. Both CMDBf and ITIL v3 were developed simultaneously and a bit isolated from one another. This question is related to the next one, so please read on.

Question: If the proper, forward-looking term is CMS—and not CMDB—why does the groundbreaking CMDB Federation Working Group even call it a CMDB?

Answer: The CMDBf Working Group began its work long before the release of ITIL v3. During this period, CMDB was still the prevalent term and gaining momentum. ITIL v3 outlines the overarching purpose and principles of the CMS, whereas CMDBf clarifies the technology needed to make the CMS work. They both address the same challenge and align very well with each other. Each is an important innovation that together drives us forward from the CMDB of yore and the CMS of the future. One major intent of this book is to bridge the gap between these two brilliant developments. The CMS is an evolution and will continue to be. Neither CMDBf nor ITIL v3 is perfect, nor is this book, but all are forward steps in the ongoing continuum of the CMS.

To eliminate confusion, we encourage a simplification that generalizes the "CMDB" parts of both initiatives according to the hierarchical structure presented by the CMDBf. We call each individual part an MDR, and the whole system is the CMS. We expand on this structure much more in Chapter 4.

We prefer to euthanize the CMDB term, but we realize that will take time. A growing number of other influential members of the IT service management community agree, including authors of both CMDBf and ITIL v3. It is indeed uncanny how many people enthusiastically agree when we express our disdain for the CMDB term!

Summary

At this point in the book, we hope we have conveyed a clear picture of the macro-level challenges and opportunities of the CMS. In this chapter, we explained the enigma that is a CMDB. The CMDB is many things, but it certainly is not a single database. The CMDB is bound by the limitations imposed on it by rampant misinformation. Although we dislike the term, the CMDB's woes are more a product of culture than definition. We strongly endorse the

transition from discussing CMDB to the more flexible and powerful CMS. CMS is fresh, and we all have a profound opportunity and duty to get it right.

Leading thinkers and practitioners have embraced the principles of the CMS. The publication of ITIL v3 was a catalyst for this inflection. It will take a long time, probably years, to abandon the CMDB term, so we will all continue to use it in conversation. For the purposes of this book, we call the entity a CMS. This is the future. Because we encourage its use, it is our responsibility to follow our own advice. We use CMDB only in a historical context throughout the remainder of this book.