

A dark grey horizontal bar with a white triangle on the left side pointing to the right.

What Do You Need from a Configuration Management Database (CMDB)?

TABLE OF CONTENTS

TABLE OF CONTENTS

Executive Summary	1
Why a CMDB?	2
- The importance of configuration management	2
- Data is the key	2
- Evolution of the CMDB	3
Recommended CMDB approach	6
- The contents of an ITIL CMDB	6
- How the pieces fit together	8
What a CMDB should do for you	10
- Federation of data	10
- Flexible data model	10
- Partitioning of configurations	11
- Reconciliation of configurations	11
- Open access to data	12
Conclusion	13
Glossary	14

Executive Summary

You may have been told you need a configuration management database (CMDB), but do you know why, and more importantly, do you know how to approach it? This white paper from BMC Software demonstrates the need for configuration management and a CMDB in particular, based upon IT Infrastructure Library (ITIL®) goals. It defines the different types of data that are usually stored in a CMDB, from configuration items (CIs) and their relationships to related data, such as change requests and a service impact model.

After a quick review of CMDB approaches that have been tried already, such as integrating data stores or putting everything into one central database, this paper sets forth our approach to a CMDB: a federated model that serves as your source of record for CIs, but preserves your investment in existing data stores.

The paper reviews the features that a CMDB needs, including a flexible data model, partitioning of configurations, reconciliation of configurations, and open data access. It explains how each of these features contributes to an overall configuration management solution that will allow your IT organization to provide the best service possible.

Why a CMDB?

Due to a growing interest in adopting best practices across IT departments, particularly according to standards such as ITIL, many organizations are now deciding to implement a configuration management database (CMDB). They realize there is a business value in having a single “source of record” for their organization that provides a logical model of the IT infrastructure to identify, manage, and verify all configuration items (CI) in the environment.

THE IMPORTANCE OF CONFIGURATION MANAGEMENT

IT departments face numerous challenges in providing dependable services that support a company’s business goals. Solving most of them requires a good configuration management strategy: without knowing what’s in your environment, you cannot hope to control it, maintain it, or improve it.

Goals

According to the ITIL *Service Support* manual¹, configuration management should pursue these goals:

- > Account for all the IT assets and configurations within the organization and its services
- > Provide accurate information on configurations and their documentation to support all the other Service Management processes
- > Provide a sound basis for Incident Management, Problem Management, Change Management and Release Management
- > Verify the configuration records against the infrastructure and correct any exceptions

Benefits

Achieving these goals can benefit your organization in significant, measurable ways related to control, integration, and decision support.

Control

Verifying and correcting configuration records gives you a greater degree of control over your infrastructure. For example, by controlling the versions of configuration items, you reduce the complexity of your environment, reducing desktop support costs. Items that disappear or that appear without being paid for will be noticed, helping you control assets and avoid legal issues. Exercising greater control over your environment also means you can increase overall security.

Integration

When processes such as incident management, problem management, change management, and release management are based on a current record of your configuration, they can be integrated, reducing administrative costs and errors. For example, you might integrate incident management and change management processes in two ways:

- > When resolving an incident requires a change, the incident management application can automatically create that change request.
- > An incident or problem management application can use a service model to identify previous changes that may have caused a failure.

Integrating all configuration-related IT processes can reduce the number of staff needed to administer your environment, saving you money.

Decision support

Your IT managers benefit from having this accurate configuration information mapped to your service management processes. Making decisions is easier when you have complete and accurate data, resulting in better resource and performance estimates. You can commit to service levels more confidently, and your risk management will improve, reducing unplanned downtime.

DATA IS THE KEY

You can choose to begin your configuration management efforts with any of the processes we’ve mentioned already, or several others. But no matter which configuration management processes you implement, the thing that makes them effective is the data they use.

¹ Office of Government Commerce, Best Practice for Service Support (London: The Stationery Office, 2000).

WHY A CMDB?

Your configuration data must be accurate, which means it must be updated frequently. Configurations are constantly changing, so last week's correct data could be horribly obsolete this week, resulting in a purchase of ten servers when you only needed five, or worse, the installation of a security patch that causes a system to fail.

Your configuration data must also be available to all your IT processes, because even the most accurate data is useless if you can't get to it. For example, if the network topology data provided by your discovery application is not accessible to your change management application, you won't be able to intelligently plan a network redesign.

The solution that allows you to maintain accurate configuration data that is shared by multiple IT processes is a CMDB.

EVOLUTION OF THE CMDB

The concept of a CMDB has evolved over the years from a collection of isolated data stores to integrated data stores to a single, central database, each time getting closer to a database that can be the source of record for configuration data without taking a toll on your infrastructure.

Isolated data stores

At first, a CMDB consisted simply of several applications that stored their own data and often other databases holding configuration data, as shown in **Figure 1**.

This approach could meet ITIL's first goal of accounting for IT assets and services, but because the data was not integrated, it fell short of the others. Your asset management application

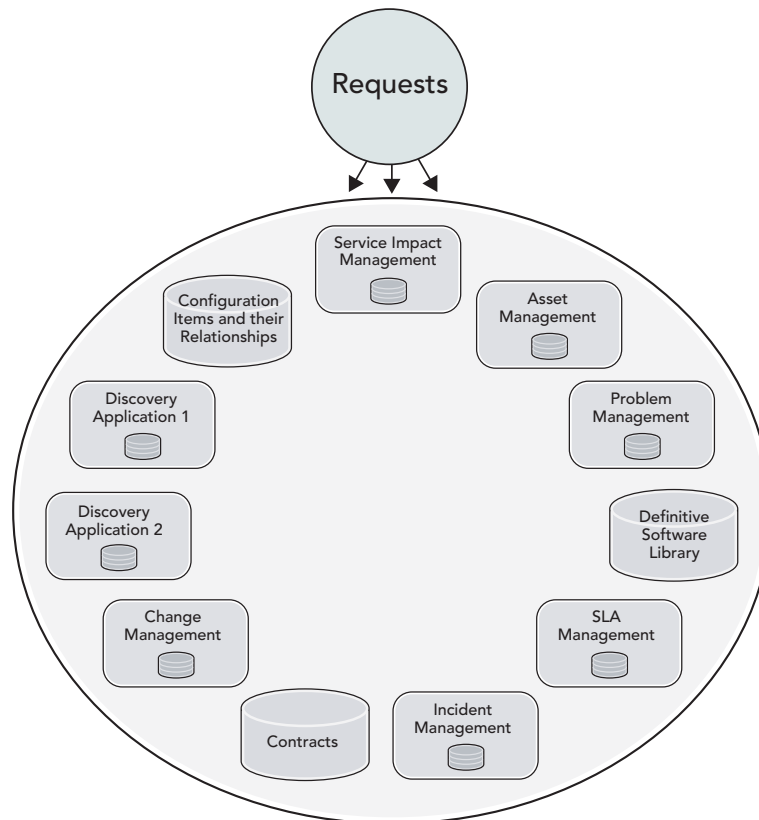


Figure 1: Isolated data stores

WHY A CMDB?

could not see data from a discovery application, and your service impact management could not modify service level agreements (SLAs).

Another drawback was the lack of a single entry point, forcing anyone who needed data to know where to find it and how to access it. Lastly, this approach did not allow you to store information about the relationships between CIs. For more information on relationships, see the "Relationships among CIs" information found in the "Recommended CMDB approach" section of this paper.

Integrated data stores

Next, IT organizations created CMDBs by directly integrating their various data sources and applications, connecting each data consumer to each provider from which it needed data, as shown in **Figure 2**.

This approach allowed different configuration management processes to share data, greatly improving the CMDB's usefulness. But it required a lot of resources to create and maintain the various integrations. And, as with the isolated data stores approach, someone unfamiliar with the system might not know where to look for a given piece of data.

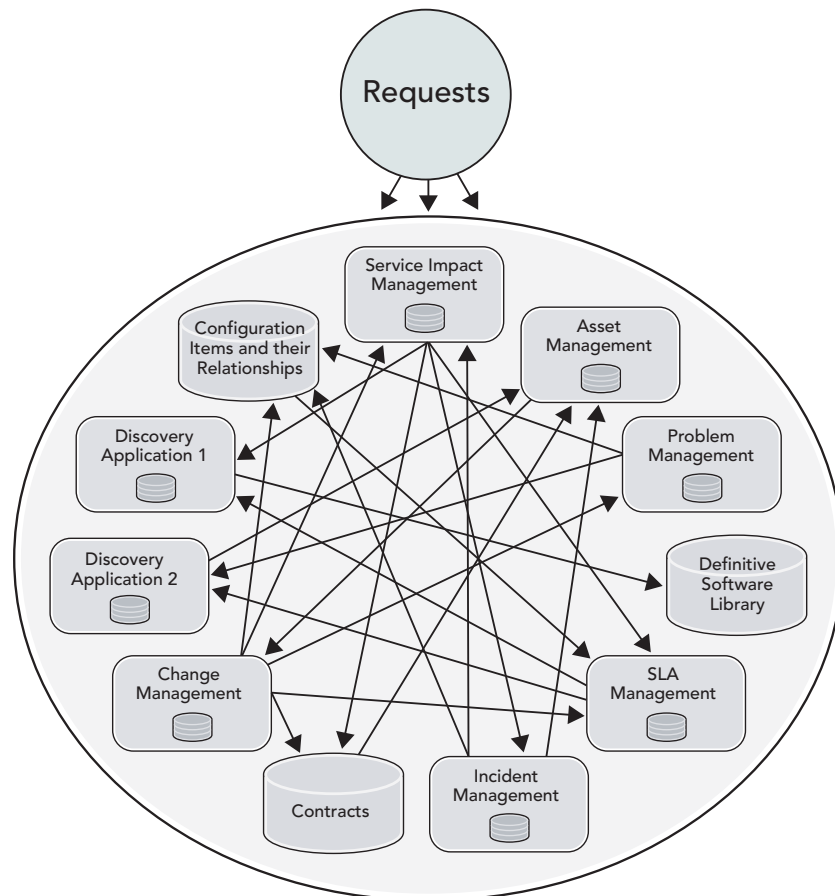


Figure 2: Directly integrating disparate data stores

One centralized database

Most recently, vendors have been offering a single, all-encompassing CMDB to hold configuration data, accessible by all applications that need the data, as shown in **Figure 3**.

In this approach, any application that is integrated with the CMDB—both the consumers and providers of data alike—can access all configuration-related data, which takes sharing a step further than the integrated data stores approach. And, it offers a single point of entry, making the CMDB the source of record to which users can submit all requests.

But an all-encompassing database has its drawbacks, too. It requires a large capacity in one place, and creates a bottleneck because all requests for and updates to data pass through the same path. It also requires a massive migration to get all of your data into the single database, creating a complicated data model that must change if any application integrated with the CMDB changes. And, unless those applications come from the same vendor as the CMDB itself, all of the integrations are likely to be a daunting task.

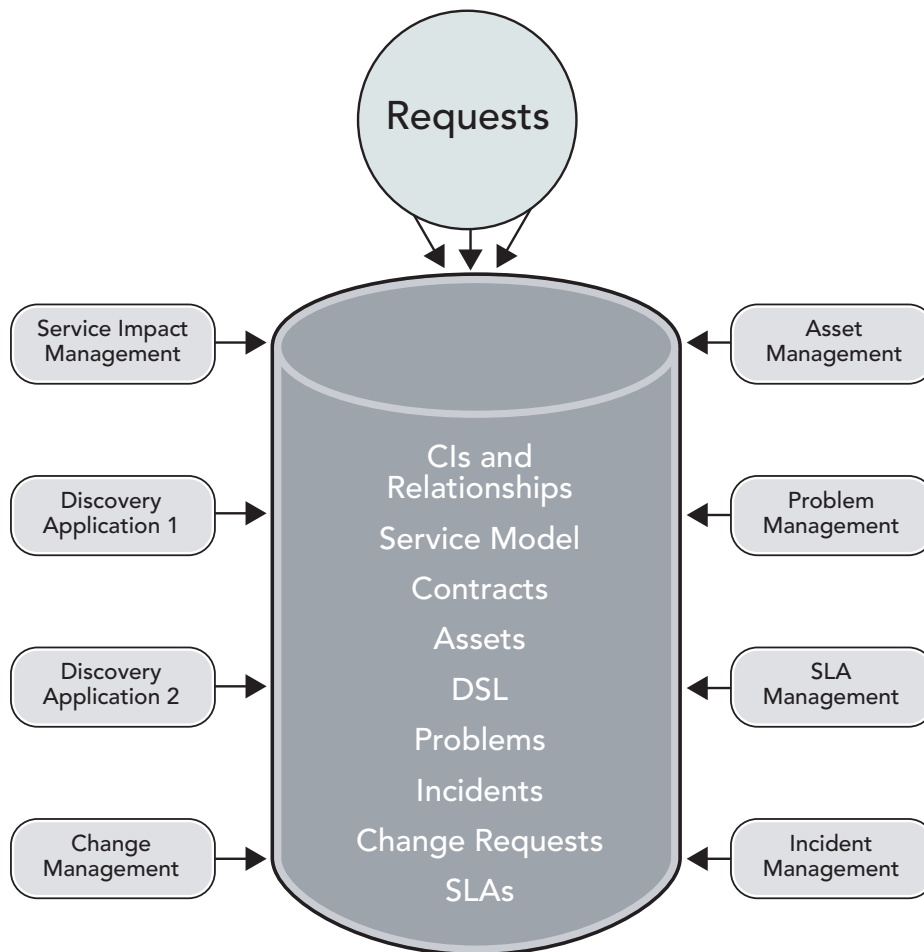


Figure 3: One centralized database

Recommended CMDB approach

BMC Software believes a CMDB with a federated data model, featuring a centralized database linked to other data stores, is the best way to share configuration data without the high setup and maintenance costs associated with the pure centralized approach. This section describes the types of data involved and then explains the details of how the federated model separates data.

THE CONTENTS OF AN ITIL CMDB

ITIL recommends that you store several types of data in the CMDB. Its main purpose is to hold CIs and the relationships among them, which together form a configuration at a

particular time or state. ITIL also suggests that the CMDB can hold data related to CIs, such as help desk tickets or SLA definitions.

What is a configuration item?

CIs are the focal point of a CMDB. Without a clear definition of what qualifies as a CI, you'll constantly struggle with deciding whether to put different kinds of data into the CMDB.

Simply put, a CI is *an instance of an entity that is part of your environment and has configurable attributes specific to that instance*. These entities can be physical (such as a computer system), logical (such as an installed instance of a software program), or conceptual (such as a business service). But they must be a direct part of your environment, rather than information *about* such a part. Some examples will help illustrate the boundary we've just drawn:

Configuration Items	Not Configuration Items
<ul style="list-style-type: none"> > A computer system is part of your environment and has configurable attributes, such as serial number, processor speed, and IP address. > A building is part of your environment and has configurable attributes, such as number of rooms, climate control system, and alarm system. > An employee is part of your environment and has configurable attributes, such as skills, hours, and department. > A software instance installed on a computer system is part of your environment and has configurable attributes, such as serial number, patch level, and deployment method. > A business service is part of your environment and has configurable attributes, such as criticality to the business and cost of interruption of service. 	<ul style="list-style-type: none"> > A help desk ticket has configurable attributes, but is not a direct part of your environment. It is information about other entities (a computer system, for example) that are part of your environment. > An archived software package is part of your environment, and is usually stored in the Definitive Software Library (DSL). > A service level agreement has configurable attributes, but is not a direct part of your environment. It is information about other entities (a Web server, for example) that are part of your environment. > A contract has configurable attributes, but is not a direct part of your environment. It is information about other entities (a photocopier, for example) that are part of your environment. > An event does not have configurable attributes and is not part of your environment.

Of course, not everything that qualifies as a CI is worth tracking, so you probably won't create records in the CMDB for all the office chairs in your organization.

Relationships among CIs

CIs don't exist in a vacuum, they affect each other. One CI might use, depend on, be a component of, enable, be a member of, or be located in another CI, just to give a few examples. Storing these relationships in the CMDB allows you to see how CIs interrelate and affect one another.

Relationships can be simple, such as a disk drive being a component of a computer system, or more complex, like those shown in **Figure 4**.

Relationships not only exist between physical CIs, but also between logical and conceptual CIs, such as the software instances and service in **Figure 4**. Two CIs can have more than one relationship with each other: for example, an employee might own a server and also operate it.

Relationship data makes the CMDB a powerful decision support tool. Understanding the dependencies and other relationships among your CIs can tell you, for example, how upgrading Processor A would improve Server B's performance, or which services would be affected if Router C failed. Most downtime is caused by problems stemming from configuration changes, and this is the information that can help you prevent that.

Related data

There is also a lot of information related to CIs, such as help desk tickets, change events, contracts, service level agreements (SLAs), a Definitive Software Library (DSL), and much more. Though these things are not CIs themselves, they contain information about your CIs and form an important part of your IT infrastructure.

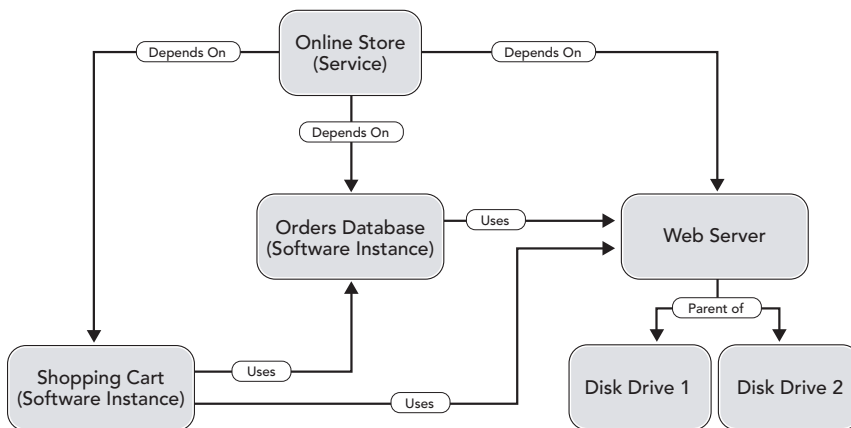


Figure 4: Example relationships

HOW THE PIECES FIT TOGETHER

The CMDB and its infrastructure should be broken into three layers. These are the CMDB itself; related data linked to or from the CMDB, called the CMDB Extended Data; and applications that interact with these two layers, called the CMDB Environment as shown in **Figure 5**.

The CMDB and CMDB Extended Data layers together make up what ITIL refers to as a CMDB. Separating this into two layers is what distinguishes the federated approach from those described in the "Evolution of the CMDB" information found in the "Why a CMDB?" section of this paper.

The CMDB

The CMDB holds only CIs and their relationships, but some of their attributes can be linked to the CMDB Extended Data. Not all available CI attributes must be stored in the CMDB: in fact, you should store only the key attributes here and link to the less-important ones in the CMDB Extended Data.

Even though the CMDB doesn't hold all attribute data or related data, it still serves as the source of record for configuration data because it links to the CMDB Extended Data. You can make all requests to the CMDB, and when the data you need is not stored there, you will find reference links to where that data is stored and information on how that data can be accessed.

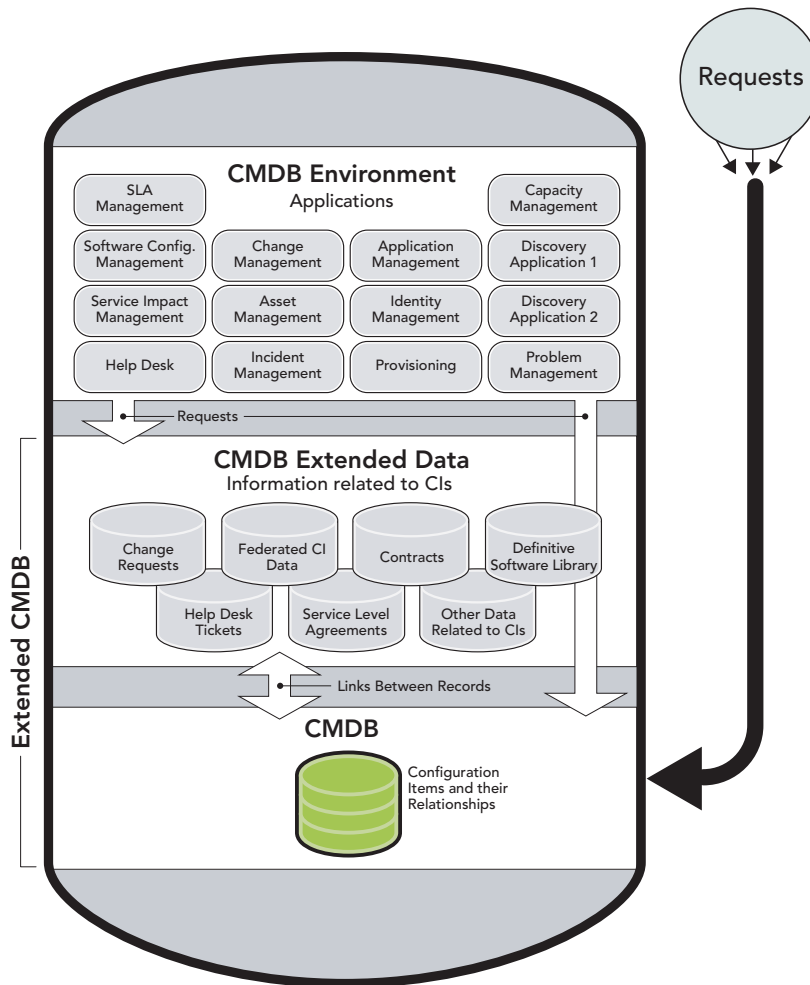


Figure 5: Recommended CMDB infrastructure with federated data model

The CMDB Extended Data

The CMDB Extended Data holds the data specified in the “Related Data” information found in the “Recommended CMDB approach” section of this paper, as well as any CI attributes judged as unnecessary to be stored in the CMDB.

The data in the CMDB Extended Data layer is linked to the CI data in the CMDB. By definition, federated CI attributes are linked from their instances in the CMDB, allowing requests to the CMDB to reach these attributes. But for other types of extended data, the link can be in either or both directions. For example, a change request record could have a link through which you can access the instances of the CIs it will change, and each CI instance could have a link through which you can access the change requests that affect it.

This has several benefits:

- > The CMDB can focus its functionality on CIs and their relationships. This functionality includes partitions for multiple “snapshot” versions, reconciliation of data from multiple sources, and federated data, and is detailed in “What a CMDB should do for you” section of this paper.
- > The overhead required to provide this functionality is not wasted on data that doesn’t need it. For example, multiple snapshots of a DSL are unnecessary, so making your DSL part of the CMDB would waste valuable storage space.
- > You don’t have to modify the CMDB to hold related data. With the boundary drawn at CIs and their relationships, the question of whether to store some new type of data in the CMDB is already answered. You store it instead as part of the CMDB Extended Data, and save the trouble of changing the data model in the CMDB to accommodate the new type of data. You also avoid pitfalls inherent in trimming down the data model if you later decided to move data out of the CMDB.
- > Transactional data can be stored in databases better able to handle a high volume of requests, instead of in the CMDB.
- > Data is provided more efficiently. Instead of getting all their data from the CMDB, data consumers can get it from individual data stores that are optimized to provide the specific type of data being requested.
- > You don’t need to undertake several data migrations and application integrations to move your change requests,

help desk tickets, and other CI-related data into the CMDB. Applications that use this data can continue to access it where you currently store it.

- > The CMDB doesn’t become a bottleneck. With requests for related data on its own being handled by other databases, the CMDB does not have to accommodate all such traffic in addition to CI-related requests. You can spread the load across multiple systems.

Though you could store your CMDB Extended Data in one place, you don’t have to. The different types of data in this layer are not necessarily linked to or related to each other. The only thing they need have in common is a link to or from the CMDB.

The Extended CMDB

Together, the CMDB and CMDB Extended Data form the Extended CMDB. *This is equivalent to the term “CMDB” as used by ITIL.*

The CMDB Environment

Where the Extended CMDB contains data, the CMDB Environment is devoted to the applications that provide and consume that data. These applications can access the CMDB, the CMDB Extended Data, or both. For example, an asset management application that views and modifies CI instances in the CMDB is part of the CMDB Environment as a consumer, and a discovery application that creates CI instances in the CMDB is part of the CMDB Environment as a provider.

These applications sometimes store their information in their own databases, but those components are still considered to be part of different layers of the CMDB infrastructure. An application is part of the CMDB Environment, while its configuration-related data is part of the Extended CMDB. Of course, applications in the CMDB Environment can also access data unrelated to CIs. This data is not part of the Extended CMDB.

What a CMDB should do for you

After reading the previous section, “Recommended CMDB approach,” you know how to structure your CMDB and relate it to the rest of your infrastructure. But even with the right structure, there are several features that a CMDB needs to effectively manage your CIs. These are:

- > Federation of data
- > Flexible data model
- > Partitioning of configurations
- > Reconciliation of configurations
- > Open access to data

FEDERATION OF DATA

We’ve touched on this concept earlier in the paper, so you already know that federation refers to a central repository holding some data directly, while linking to other data in other sources.

You might choose to federate some attributes if you want to track them, but not track them as often or as vigorously as a CI’s key attributes. These secondary attributes are the first of two kinds of data you can federate.

This means that, for example, the CMDB record for an employee might have a Skills attribute that contains a list of the employee’s skills, and a Department attribute that contains the employee’s department name. It might also be involved in a relationship with an HR data store where additional attributes, like Salary, that are not really important from a configuration perspective are stored.

The other type is data that is related to CIs but where the data is not really attributes of a CI; that is, data that refers to or is referenced by a CI to provide additional content on extended functionality to the CI but is not part of the CI itself.

For example, your CI records for software instances might have a License relationship containing the URL to an intranet page where the license is posted, or each CI record might have a Problems relationship that contains the information necessary to search a problem database for all issues concerning that CI.

Benefits of federated data include:

- > You save the overhead of importing, tracking, and reconciling the data in the CMDB.
- > You have a standard way to cross-reference related data.
- > The federated data can be in multiple locations.
- > You retain your investment in other data stores.

FLEXIBLE DATA MODEL

You have many different types of CIs, from computer systems to network hardware to software servers. Without a data model that accurately reflects these types and the types of relationships that can exist between them, your CMDB could store attributes that don’t pertain to their CIs, leave out necessary attributes, and make it harder to search for groups of CIs. This data model must be both object-oriented and extensible.

Object-oriented

An object-oriented data model has a hierarchical set of classes where each class inherits attributes from its superclass, the class above it, in the hierarchy—and then adds its own attributes to create a more specific type of object, a subclass. Subclasses can have their own subclasses, extending the hierarchy to whatever level of detail you want to track.

For example, the class ComputerSystem might have the attributes Domain, ProcessorType, and Manufacturer. The ComputerSystem class might have subclasses called LaptopComputer, DesktopComputer, and MainframeComputer. These subclasses each have the three attributes of their superclass, plus attributes specific to themselves. **Figure 6** shows part of an object-oriented CMDB data model, encompassing a superclass and two levels of subclasses.

The benefits of an object-oriented data model include enforcement of common attributes among similar types of CIs and the ability to search within not just a given class of CIs, but within any branch of the hierarchy. If the data model has one base class from which all others are subclassed, you can search for all CIs and their relationships.

Extensible

Your infrastructure, and the technology that comprises it, is constantly changing. That means the types of CIs and relationships in your CMDB must also change, so you need a data model that is extensible. You should be able to add and

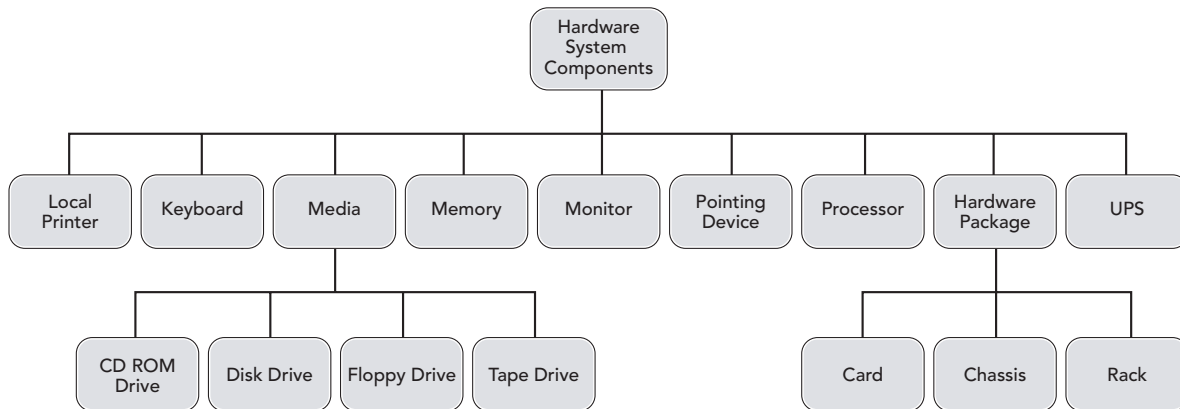


Figure 6: Part of object-oriented data model

remove attributes from your classes, and even add and remove classes.

Though this feature is important, you should take care not to overuse it. The CMDB should hold only common CIs and their relationships. Adding classes and attributes for unimportant CIs will needlessly tax your CMDB. Also, subclassing too far can leave you with classes so narrowly defined that they have very few members. Balance the need for categorization with the need to store similar CIs together.

PARTITIONING OF CONFIGURATIONS

Partitioning is the ability to divide your configuration data into pieces called datasets, each representing a set of data at a certain point in time. This allows the same CI or relationship instances to exist in more than one dataset.

This is important for the goal of verifying and correcting configuration records against the infrastructure. You can create one dataset representing your intended configuration, then use a discovery application to create another dataset representing your actual configuration, and verify the former against the latter.

Partitioning is a powerful tool that you can use for many other purposes. Datasets could represent:

- > An obsolete configuration
- > A future configuration

- > A tested "gold standard" configuration
- > Different versions of a current configuration
- > Subsets of an overall configuration
- > Data provided by different discovery applications
- > Data from different clients' configurations (multi-tenancy)
- > Other ideas you can invent

RECONCILIATION OF CONFIGURATIONS

When you have more than one dataset containing the same instances, reconciliation is the process of *identifying* the matching instances in all datasets and then either *comparing* the different versions of each instance and reporting on the differences or *merging* the datasets into a new dataset. This allows you to see changes over time, or to determine one desired configuration when you have data from multiple discovery sources.

Identifying instances

Before you can compare different versions of something, you must determine that they indeed represent the same entity. Identification accomplishes this, applying rules you specify against instances of the same class in two or more different datasets.

For example, a rule intended to identify computer system instances might specify that the IP addresses of both instances be equal. When the rules find a match, both instances are tagged with the same identity, an extra attribute showing that they each represent the same item in their respective datasets.

Comparing datasets

A comparison activity operates against instances in two datasets and produces a report showing those instances that appear in only one of the datasets and detailing the differences between instances that appear in both. Only instances that have been given an identity should be considered by a comparison activity.

A comparison function lets you compare an expected configuration against an actual one, which you could use for more than one purpose. You might use comparison to alert you that something has changed in a configuration that you expected to remain static. Alternatively, if you have a change request in progress, you might use comparison to verify that the configuration reaches its expected new state.

Merging datasets

Merging takes two or more datasets and distills them into a new unified dataset according to precedence rules you specify. This is usually done to determine one valid configuration when different discovery applications provide overlapping data about the same items.

Only instances that have been given an identity should be considered by a merge activity. The precedence rules should specify weight values for desired classes and attributes in each dataset. Whichever dataset is given a highest weight for a given class or attribute will have its value for that class or attribute placed in the result dataset.

A merge function is essential when you have two or more discovery applications that discover the same CIs. Each discovery application will likely have areas of strength and weakness compared to the others, so you can create precedence rules that favor those strengths. This gives you one CI instance with the best of all discovered data.

OPEN ACCESS TO DATA

As we mentioned earlier, even the most accurate data is useless if you can't get to it. It is important to remember that you need to allow users and applications to both read and write to the CMDB. *Consumers* view and modify existing data, while *providers* create and modify the data.

This requires at least these features:

- > Programmatic access: The CMDB must provide an application programming interface (API) or other method for programs to view and modify its data. This should include both instance data and the classes of its data model.
- > Bulk data load: The CMDB must provide a way to import multiple instances at once, so that discovery applications and others can rapidly populate the database.
- > Database and platform independence: The CMDB should be compatible with multiple operating systems and database vendors to allow you flexibility with your environment.

Conclusion

To best pursue ITIL's goals for configuration management, your CMDB should:

- > Store only CIs and their relationships, with related data stored in the CMDB Extended Data
- > Federate data so that the CMDB can be the source of record while linking to related data and less-important attributes
- > Support an object-oriented and extensible data model
- > Support partitioning of configurations
- > Support reconciliation of configurations
- > Allow open access to data

BMC Software offers a product line built on this philosophy. To learn how these products can help you reach your configuration management goals, contact your BMC Software sales representative or visit www.bmc.com/cmdb. For further reading on configuration management and CMDBs, we recommend these sources:

- > The ITIL manuals, particularly Service Support, available at www.ogc.gov.uk
- > The Common Information Model (CIM) standard from the Distributed Management Task Force (DMTF), available at www.dmtf.org

Glossary

business service

A service you provide to IT customers that depends on configuration items you track.

CMDB

A Configuration Management Database. Should hold important attribute data about your CIs and their relationships, and link to related data and lesser attributes.

CMDB Environment

The consumer and provider applications that work with data in the CMDB and CMDB Extended Data.

CMDB Extended Data

All data stores, whether or not related to each other, which hold federated CI attributes or CI-related data, such as help desk tickets.

CI

See configuration item (CI).

CIM

The Common Information Model, a standard data model from the Distributed Management Task Force (DMTF).

compare

To report on the differences between two datasets or parts of datasets.

configuration item (CI)

An instance of a physical, logical, or conceptual entity that is part of your environment and has configurable attributes specific to that instance.

consumer

An application that views or modifies data already in the CMDB. See also provider.

dataset

A group of CI and relationship instances representing a set of data at a certain point in time. Different datasets can hold versions of the same instance.

Definitive Software Library (DSL)

A repository containing one copy of the approved versions of software used by an organization, used to control installed versions.

discovery

The process of scanning a network and automatically detecting the IT assets installed on it.

DMTF

The Distributed Management Task Force, an IT standards organization that created the Common Information Model (CIM).

DSL

See Definitive Software Library (DSL).

Extended CMDB

The combination of the CMDB and the CMDB Extended Data. See also CMDB, CMDB Extended Data, CMDB Environment.

federated data

Data that is not stored directly in a central repository such as a CMDB, but is linked to from that repository, extending the amount of data available from it.

identity

A unique string to identify an instance for matching it with instances in different datasets, allowing them to later be compared or merged.

instance

A single existing occurrence of a class of item.

ITIL

Information Technology Infrastructure Library, a set of best practices developed by the British government for managing IT services.

merge

To create a new dataset by taking instance attributes from two or more datasets according to precedence rules.

partition

To separate data into datasets.

provider

An application, usually a discovery application, that populates the CMDB with data. See also consumer.

reconcile

To perform some combination of identifying, comparing, and merging data.

relationship

A connection between two CIs that affect each other in some way.

source of record

A definitive source for a certain type of data. Either contains the data or can point you to it.

subclass

An object-oriented class that is derived from another class, called its superclass. The subclass inherits all the attributes of the superclass.

superclass

An object-oriented class that has subclasses derived from it.

About BMC Software

BMC Software is a leading provider of enterprise management solutions that empower companies to manage their IT infrastructure from a business perspective. Delivering Business Service Management, BMC Software solutions span enterprise systems, applications, databases and service management. Founded in 1980, BMC Software has offices worldwide and fiscal 2004 revenues of more than \$1.4 billion. For more information about BMC Software, visit www.bmc.com.

