CHAPTER 3

Determining Scope, Span, and Granularity

After gathering and documenting effective requirements, you have a solid foundation on which to build a configuration management system. Some of those requirements most likely deal with the data you will put into the Configuration Management Database (CMDB). This chapter explores that topic in much more depth, recommending that you look at the shape of the CMDB as a three-dimensional matrix.

The first dimension of the matrix is called *scope*. Scope indicates what the potential contents of the CMDB will be—that is, the categories of objects that will be included and what kinds of relationships you can capture between those categories. Scope provides the general outline of your database schema and is easiest to think of using object-oriented terms. Scope indicates the object classes that will be included.

Span is used to indicate which specific groups will be tracked within each object you've defined in the scope. We'll learn that span is a necessary dimension because there are many cases where you want to track some members of a group, but not all of them. A classic example is documentation, where you certainly want to track some very important documents as configuration items, but it would be outrageously expensive to track every single document that the information technology (IT) organization produces.

The third dimension of the matrix is called *granularity*. While scope indicates what will be contained, granularity determines what will be known about each configuration item or relationship. Most people relate to a computer screen, so think of the granularity as a definition of which specific fields will be on a screen whose purpose is to show a single configuration item (CI) or relationship. If you are familiar with object-oriented programming terms, the granularity indicates the attributes of each object.

We cover scope and granularity in separate sections in this chapter, and then pull together some thoughts on the entire CMDB schema at the end of the chapter.

Scope

Although a CMDB can be extremely complex, it is built of only two elementary constructs, called *configuration items* and *relationships*. *C*onfiguration items represent static portions of the IT environment, such as computers, software programs, or process documents. Relationships, as the name implies, track how these configuration items are related to one another, and are much more dynamic because these relationships can change frequently. Given these simple building blocks, defining the scope of a configuration management system is as simple as deciding which types of configuration items you want to track and which relationships will be important.

Note that we define scope as which *types* of configuration items will be tracked, not which configuration items. Once we decide that a particular type of thing is going to be tracked, it becomes part of our scope, even if we choose to track only a single instance of that type of thing. The choice of how many of each type, and exactly which ones, is part of the span of the CMDB, and is discussed later in this chapter.

Although scope is a very simple concept, it can be extremely difficult to define for your organization. Because scope gives the CMDB its shape, you should choose a set of categories that are easy to understand and demonstrate complete coverage of the IT space. You will certainly want to include types of objects from the different areas you manage—servers, mainframes, data networks, telephony, packaged software, developed software, and whatever else is part of the purview of your organization. Additionally, you may want to consider a set of categories for documentation because IT Infrastructure Library (ITIL) best practice describes configuration management as including control of the user guides, process documents, and other artifacts that are part of the operational space.

Fortunately, help is available in determining which categories to use. The Desktop Management Task Force (DMTF) has created a standard called the Common Information Model (CIM), which conveniently enough is a set of categories for describing an IT environment. If you would like to see the full standard, visit the DMTF web site at www.dmtf.org/standards/cim/. The standard can be somewhat daunting because it covers a very wide range of situations and is necessarily complex, but it is well worth considering. Increasingly, tool vendors are adopting the CIM as the basis of their "out-of-the-box" database, which can greatly reduce your implementation time if you adopt the model. Most organizations will not select a scope that covers the entire standard but will choose a subset of the available categories from the model.

One very good aspect of the DMTF standard that should be part of your configuration management scope is the inclusion of what the DMTF calls associations, and what we've been calling relationships. It is hard to overemphasize the role that these relationships between configuration items play in the overall CMDB. A beginner's mistake would be to focus on the configuration item scope but overlook the importance of defining which relationship categories will be used. Be sure to carefully consider which kinds of associations you will include.

Let's consider some specific examples of scope decisions. Some obvious configuration items to include are UNIX servers, licensed software packages, and your organization's key process documents. In the relationship area, you will most certainly want to include relationships between software and hardware and between servers and the networks they reside on. But there is also plenty of gray area. Take, for example, the hard disk that comes in each personal computer. You could call this a separate configuration item and potentially get important information about which machines will need to have more disk space to accommodate the latest operating system. On the other hand, do you really want the complication and expense of tracking every disk replacement done throughout the organization? A cost analysis might show it is less expensive to just bump up the size of ALL disk drives purchased than it would be to track all the change records needed to replace workstation disks. It is easy to see that setting the scope of the configuration management process can involve hundreds of business decisions.

One distinct possibility to keep in mind is that you might have a multi-tiered CMDB system—that is, you might have a CMDB dedicated to each smaller domain of information such as servers, business applications, network equipment, and others. These “sub-CMDBs” could then have a much lower level of scope than is needed or wanted at the enterprise CMDB. This kind of model requires that you think about cross-CMDB relationships as well as those relationships that fit neatly within a single sub-CMDB, requiring a much more elaborate scope definition.

Examples of Included and Excluded Scope

As a further illustration of choosing the appropriate scope, let’s look at some broad categories that are most often included or excluded. This is not meant to be a comprehensive list of possibilities, but a starter list to provoke some thoughts around what is appropriate for your organization.

Musing about the scope of IT infrastructure, it is easy to come up with broad categories such as hardware, software, networks, and documents. These are all included in the scope of configuration management, but are still too broad. Under hardware, there are workstations, servers, network equipment, telephony equipment, personal devices (PDA’s, cell phones, pagers), and perhaps even process control equipment. All these categories are commonly included in the scope of configuration management. In the software realm, scope probably includes categories such as operating systems, business applications, desktop productivity software, database management systems, web application server software, transaction processing middleware, and even embedded software.

Relationships categories are not as easily identifiable. In most cases, you can define the types of relationships you want to maintain by looking at the list of configuration items. Are you keeping track of operating system software? If so, you’ll want a category for describing the relationship between an operating system and a computer. If you decide to track logical network concepts like IP subnets or security zones, you will want to create a category for tracking hardware relationships to those logical networks, and perhaps even the relationships between logical networks and physical network devices. Throughout this book you’ll see many other ways to leverage the exciting possibilities of relationships, and this may give you ideas of categories you want to use in your own scope.

On the other hand, some items are clearly out of the scope of your configuration management system. Incident records, change records, and similar kinds of information may be stored in the same physical database as your configuration management data, but they are not configuration items in and of themselves. Likewise, many documents such as business continuity plans and change back-out plans are not configuration items, even though they likely reference configuration items. Relationships between a printer and the workstations configured to use it are an example of an association that is probably out of scope for most configuration management systems.

Perhaps the most unclear aspect of choosing scope concerns documents. One reason documentation is a difficult category is that few solid relationships to documentation can be tracked easily. ITIL is clear that documents should be referenced in the CMDB. Documents that substantively help define the IT environment should definitely be included in scope. Some examples are the document describing how to perform change management, the list of all software installed on a standard Linux® server, and the installation guide for a custom-developed business application. But it is clearly ridiculous to store every document that describes some aspect of the IT configuration. You wouldn't, for example, store the standard installation manual for Microsoft Office, but you would most likely store a document that one of your people wrote describing how to quickly roll out Microsoft Office across your network domains. Think carefully through the issues surrounding the scope of documentation and especially the relationships you want to define for documents.

Another often-debated area of configuration management is the role of people and organizations. Some organizations choose to manage these expensive IT resources just like any other resource and thus decide that people and organizations are two of the categories which are part of the scope of configuration management. Others decide that the intricate relationships between people, organizations, and IT are best managed in a dedicated human resource system and would be too costly to also try to track in the IT configuration management system. There are advantages and disadvantages to both approaches, so the decision is never an easy one. Follow the guidelines in the next section to make the best decision for your organization.

Complex decisions clearly must be made around the scope of configuration management. Rather than proscribe exactly what should be in your CMDB, I'll provide a set of guidelines you can use as you make these decisions. By starting with the model proposed by the DMTF and following the outlined criteria, you can establish a scope that will be perfect for your organization.

Criteria Used to Determine Scope

You can use the following criteria to determine the scope of configuration management for your organization. Like all criteria, they must be used with a good knowledge of how your organization works and a dose of common sense. But used correctly, they can help guide you to a good scope for your CMDB.

Start Simple

As with most complex problems, the way to start is the simplest way possible. Simplicity of scope typically involves going after those things that can be quickly learned and easily maintained. If you already have an IT asset management system, for example, you probably have existing processes for gathering and maintaining asset data. Use the same scope for your configuration management efforts, even though in the long run asset management and configuration management serve different purposes. Simplicity might also be geographic, involving only one or two of your locations in the initial scope.

This guideline of simplicity also can be applied to the depth of your coverage. For example, you might want to include business applications that you've developed yourself in the CMDB. That's a good, simple thing to do. But undoubtedly some of these applications consist of a back-end database component, a web or other presentation component, and maybe an application layer component. Tracking each application as a single entity simplifies both the configuration item scope and the set of relationships that must be created, but your approach needs to be balanced between simplicity and the needs of the organization. One of my favorite sayings is "Keep it as simple as possible—but no simpler!"

Consider the Cost

As illustrated earlier, the scope of configuration management should be determined largely by what you can afford to manage and what you can afford to ignore. On one end of this spectrum is mainframe equipment, which is relatively expensive and changes only after very careful deliberation. In almost all cases, this is well worth tracking. On the other end of the spectrum are cell phones, which are relatively inexpensive and very difficult to keep track of. This is not to say that cell phones cannot be part of your configuration management scope—just that you would want to have a very strong reason to expend the energy and money to keep track of them.

Remember that the cost of managing a configuration item over its lifetime is almost always more than the cost of gathering information about it in the first place. What drives up the cost is the degree to which the relationships change. It is amazing that the actual CIs don't change very often, but it sometimes seems that the relationships can change daily. As you develop your scope, consider the relationship model carefully and be sure that you don't add relationships that would be nice to have but will also be very expensive (or technically impossible) to maintain.

Score Easy Victories

Much of the success of your configuration management effort will be determined by how much confidence your entire organization puts in the database. Choose your scope in order to build confidence early. Include only those items you can track with a high degree of certainty, and leave out those things you are less sure of. Adding data to your scope that just ends up being incorrect later will bring your entire project under suspicion. It is possible to institute processes and tools that improve the quality of data, but consider how effective those are likely to be and how much extra risk your project must manage to put that extra process and tooling into place. If you decide the risk is worth managing (based on other criteria here), then go ahead and include the scope; but if not, then leave out the risky parts of the scope. The entire configuration management program can be killed if data is incorrect early on, so you may not get a chance to recover later.

After you've begun to track configuration items and their relationships, you should ruthlessly eliminate other sources of that information. This may seem like strange information, but experience shows that information stored in two separate places is automatically assumed to be incorrect in both of them. Your entire organization will begin to depend on the configuration management database only if they see it has content nobody else can give them. This isn't to say you should dismantle genuinely useful systems like your asset management database or software license management servers, but as part of your scope decision you should carefully consider whether you are setting up duplicate sources for the same records. You can often score a quick victory by simply referencing those other data sources rather than replicating their full content.

Look Ahead to Value

In thinking about the scope of your efforts, consider the future value of having accurate information. Experience shows that there is a class of IT projects which fail for lack of accurate information. Look forward to projects your organization is likely to tackle in the near future, and structure your configuration management database to provide the information that will be needed. Are you going to consolidate servers? Servers should be a key category in your configuration management scope. Need to roll out the next desktop operating system? Then make sure desktop systems and their current operating system are in your configuration management system. Ultimately every element you add to the scope of the database should be driven by some business value you can gain by tracking that element.

Of course, it is impossible to predict every possible need for information in the future, but it is remarkable how often looking at past projects will reveal a pattern of missing data. Think just of the large IT projects your organization has conducted in the past two years, whether those were successful or not. How many of them would have cost less or been done faster if accurate data had been available at the onset? Thinking along this line will often reveal some very obvious additions to your scope, which will add significant value to future projects.

Reduce Your Risk

Configuration management systems fail because of erroneous data. The best scope for configuration management is one that reduces the risk of errors. It is critical to plan ahead for not only how you will gather the data on all items within your scope, but how you will manage that data. Proper management involves accounting for all changes to the configuration item, whether they are planned or unplanned. You must consider carefully what will happen as configuration items break, are retired from service, are enhanced, become obsolete, and many other things that can happen to technology. Remember that each of those events is more than a simple change to one record—all of the relationships involved need to be changed as well, and this could be extensive. The more completely you have planned for every possible event in the lifecycle of your configuration items, the lower will be your risk of erroneous data. Similarly, if there are lifecycle events that you know you'll never be able to track, you will want to think twice about attempting to manage the CIs that participate in those events. For example, maybe your management team interacts directly with a supplier to request the latest model of cellular phone whenever they like. You would have no chance of tracking which phones are in your organization or which person has which phone. If that is your situation, simply don't include cell phones in your scope.

Expand Slowly

Your initial scope will not be your final scope. This shouldn't discourage you from carefully considering your scope at the beginning of your effort to implement configuration management, but it should make you realize it doesn't need to be perfect. As your configuration management efforts mature and prove their worth, you will want to expand your scope to cover a broader range of IT infrastructure. If the value is proven, it should be easy to get additional funding to implement additional processes and tools to expand to a wider scope.

Please be aware, however, that you can jeopardize your initial success if your scope outpaces the maturity of your process. Consider again the criteria provided in this section and decide which items are prudent to allow into your scope, and which ones you're still not ready to tackle. Slow and steady growth will increase your business value and keep your risk to a manageable level.

Documenting Scope

All the thought involved in determining the scope of configuration management needs to be captured somewhere. You could simply take a piece of paper and try to write out every decision you've made and why you made it. But by far the best way to document the scope is to create something that will be needed later anyway—a category structure.

The category structure is simply a hierarchical way to organize all the elements that will be included in the scope of your configuration management efforts. Start at the top with a broad category like "IT Infrastructure" or "ABC Company IT Environment." Then break that into discrete pieces that make sense given the scope you've chosen. For most people, hardware, software, and documentation will be categories at this first level. There may be others like "networks" or "telephony." Continue down the structure, going to as much detail as you've determined you need. When you've run out of new subcategories and levels to create, you'll have two very important items—the complete scope of your configuration management system and the "authoritative" name of each kind of configuration item. This set of categories will serve as the basis for working with and reporting on your configuration management system, so it is worth taking some time to consider what the categories will be and what they will be called.

As much as possible, use terms that your organization will be familiar with. If you routinely group your servers into midrange and mainframe servers, don't get formal now and call them Complex Instruction Set and Reduced Instruction Set computers in your hierarchy. On the other hand, if there is confusion among different groups or business units in your organization, try to use a name that everyone will relate to. The name will show up in change and incident tickets, reports, and of course the configuration management database itself, so make them as usable as possible. If you've adopted the CIM, you'll find that you already have the hierarchy defined. You simply need to make sure that you keep all of the parent classes to every child that you've adopted in your model.

In your scope document, draw the categorization hierarchy using a drawing tool, and then describe each class that you intend people to use. Where confusion will be possible, give explicit instructions such as "This category is for network routers that operate at network layer 3.

For switches and other devices that operate at layer 2, use the network hub category." Most implementation projects choose to document the scope, span, and granularity together in a single document called the "Configuration Management Plan" or the "CMDB Schema."

Span

As mentioned earlier, the span of your CMDB documents which items of each class you're going to capture. The obvious question is why you wouldn't capture every item of the types you've identified. There are many reasons for choosing to capture only some of the possible range of configuration items. This section describes some of those reasons and helps you document the span of your CMDB.

Span is all about establishing boundaries, and the ways to describe span are as varied as the kinds of boundaries that can be considered. For an international organization with different regulatory compliance needs in different geographies, span can be used to include only those CIs of some specific class that are in regulated countries. An investment banking concern might decide that workstations on the trading floor are critical but those in general office space are not. Perhaps one division of your organization is going to be sold soon, so you want to exclude all the business applications from that division from your span. You might also define different spans when different outsourcing vendors support different parts of the infrastructure, or even when a single IT organization supports different "customers" with different services. These are all valid ways to establish boundaries identifying which CIs will be in the database and which will not.

Of course, the moment you introduce such boundaries, you start to have issues with relationships. You need to make a policy decision on how to handle a relationship between a configuration item within your span and one outside it. These kinds of relationships can be extremely difficult to support, but sometimes there are legitimate reasons to include them in your span. For example, you might want to include only the items at a specific geographic location in your span; however, because a wide area network line connects to that location, you should include it in the database even though it has a relationship to equipment at another location as well. When this happens, be aware that you are creating a need for special procedures to deal with the relationships. You may even need special features of your configuration management tool to handle relationships between something your CMDB knows about and something it doesn't include. Unfortunately, most of the popular tools today don't offer help with this situation.

Criteria Used to Define Span

This section describes some criteria that can help you determine the correct span of your CMDB. Like those for scope above, these are general guidelines that can be used in conjunction with your knowledge of your own organization and some common sense.

Forgetting Span

I've been told in several projects to ignore the issue of span and simply gather all the data we could find. In each case, I've asked (politely of course) where I should start. The resulting direction inevitably consists of a statement of span. Even if you're going to tackle a bigger project with multiple spans, it makes sense to define the span boundaries as a tool to make project management easier.

Span Defines Projects

The definition of span can be used to help define projects. For a very large, global organization, it's nearly impossible and very risky to populate the entire CMDB as part of a single, long-running project. By partitioning the span into more manageable chunks, you can create the scope for several phases or releases of the implementation project. Geographic regions typically are a good choice for creating boundary lines, although you need to be careful of global CIs. Business applications that are used throughout the world or wide area network links between different regions are examples of global configuration items that should specifically be named in your statement of the CMDB span.

Your span does not have to define only one kind of boundary. For example, it is quite possible to define a span that includes servers at several geographic data centers and applications from one or more business units together into a single statement of span. This will almost certainly be the case if you follow the advice of Chapter 10, "Choosing and Running a Pilot Program," and start with a pilot program. For that pilot, you might need to bound the span by three or even more boundaries to get a small enough data set to make the pilot workable.

Tools Sometimes Dictate Span

The tools you choose to use sometimes dictate the span of your database. If you have an excellent tool that scans details of Windows based servers, but doesn't work on UNIX servers, you might want to exclude UNIX servers from the CMDB for a while. Once you've implemented better scanning technology, you can increase your span to include those servers. But perhaps you still don't have a good way to capture network information. This is unfortunate, but occasionally you will find yourself constrained by the tools and able to use only the span that those tools support.

Follow the Leader

When choosing the span of your database, it is important to understand the degree of risk your project sponsor will tolerate. The definition of span is the single best way to control the risk of your implementation project because it directly determines how much data you need to gather, manage, audit, and report on. Normally it is best to define the span incrementally so that you can be sure of managing a smaller set of data before trying to grow to a larger set. Your stakeholders should offer guidance in the best way to approach getting the entire enterprise.

Some items might be permanently out of your span, and some may simply be out of the span of a single project but assumed to come into the span later. These should be clearly differentiated. Policy statements should be documented to indicate which configuration items and relationships you will never incorporate and why. These policy statements will save countless future hours of confusion and renegotiation each time a separate project is begun. For those items that are simply deferred to a future project, include rationale statements in the span statement to indicate why you've chosen to defer them.

Documenting Span

There is no perfect format for documenting the span of your CMDB. Most projects I've participated in simply used a document that listed each boundary along with a rationale for placing that boundary in the span, and a consideration of the configuration items and associations that would cross the boundary as exceptions. For example, suppose you want to include executive workstations in your span, but not other workstations. In the span document, indicate your definition of an executive workstation and the reasons these are included while others are not. Also indicate whether there are exceptions, such as some other workstations to include, some relationships between included configuration items and general workstations, or some executive workstations to exclude. If there are no exceptions, simply state that no exceptions exist. The span document should be concise and understandable. Your organization may prefer to combine the scope and span information (along with granularity) into a single document.

Granularity

Scope and span determine which items you will track, whereas granularity determines what you'll track about each item. While scope describes the breadth of the configuration management effort, granularity describes the depth. Just like scope and span, granularity is determined by trading off the value of information against the cost of discovering and maintaining that information.

Granularity is best defined as the set of attributes you want to understand about each of your configuration items. As an example, let's assume you've decided that network hubs are part of the scope of configuration management for your organization. To determine granularity, you need to decide what information you will maintain about each network hub. Some obvious examples are the location of the hub, the person who is responsible for supporting the hub, and the serial number on the device. But perhaps there would be value in knowing other details, such as SNMP community string, the management IP address of the hub, and even the backplane speed. If these details are of importance to your organization, you would include them as part of your configuration management database. If you're never going to use them, it would be useless to include them and add extra cost and complexity to your project.

The three dimensions of your CMDB are described in this chapter in the order that you must define them. It would be very difficult to describe span without first understanding what the universe of all possible types of configuration items and relationships will be. Likewise, it is difficult to assign specific granularity before you know what the scope and span dimensions look like.

Granularity is typically defined for each individual category from your scope, but often the attributes that you need will be changed by the span you've agreed on. Going back to our example of the executive workstations, if your span includes all workstations, you might want to include an attribute that indicates whether the workstation belongs to an executive. Of course, if you choose to disregard all other workstations, you wouldn't need this attribute as part of the granularity definition.

Setting the granularity is by far the most complex of the three dimensions. This section will help you understand granularity better and choose the right level of configuration granularity for your organization.

Fixed or Variable Granularity

One of the key decisions to be made is whether all configuration items will have the same attributes. If you insist that all configuration items and relationships share the same set of attributes, you've chosen a fixed granularity. If you allow the category to determine which attributes are present, you're setting up a system with variable granularity. This section describes the benefits and drawbacks of each scheme.

Fixed Granularity

Whether talking about a network router or a custom-built business application, you could store the exact same set of attributes. This is called fixed granularity, and it makes your configuration management efforts much simpler. The thought is that you can define a fixed set of attributes that is wide enough so that all configuration items can be fit into this single set. Fixed granularity is most often used by organizations very comfortable with IT asset management systems and choose to convert their asset management system into a configuration management system.

The benefit of a fixed granularity system is that data collection and maintenance are greatly simplified. You can create a single spreadsheet template and fill in every configuration item as a row using the same template. Just add another template for relationships, and you have described the complete CMDB granularity. Tool implementation, reporting, and even data collection are greatly simplified using a fixed granularity.

The drawback of fixed granularity is that you must choose between two inconvenient options for things that don't fit the model. The first option is to add extra attributes and simply make them optional. For example, you could add a CPU speed attribute, and then just leave that value empty for configuration items that represent documents or software. The other option is to store detailed configuration data in another system and simply use the CMDB as an index or card catalog to point to those external sources. This can be useful if you already have all of the configuration items documented in other tools, but you want to use a centralized configuration management system to track relationships between the CIs.

There are a couple of interesting variations on fixed granularity that attempt to achieve some of the benefits of variable granularity, but without all the costs. The first option is to define a set of "extra" attribute fields that are fixed. These fields are often called "Custom1," or "Variable1," or something similar. The intent is that these fields might have different values depending on the category of the configuration item. So if the configuration item is a server, you might use the first field for the server name and the second for the main IP address of the server. If the configuration item is a software application, you might use the first field for the software version and the second field for the type of license. This represents fixed granularity because all configuration items have the same number of attributes and the same types of attributes, but it simulates variable granularity by overloading the definition of some fields. The downside of this scheme is that anyone wishing to search on or report on the configuration data will need to know the mapping of fields to categories so they know what type of data to expect in these variable fields.

Another variation of fixed granularity is the attachment of an XML document to each configuration item record. The content of the XML document can then contain additional fields and values, which represent additional attributes that are category specific. This is really just another means of representing a variable granularity in a system that doesn't natively support variable field structures. These attached XML files really represent what you would have put in as fields given another system.

Although fixed granularity seems very logical, it is seldom practical. If you're putting effort into implementing configuration management in the first place, you've already signed up for a series of challenges. Working with variable granularity is another challenge, but one worth pursuing.

Variable Granularity

What is variable granularity and why would you want to use it? This simply means that the set of information you track differs based on the type of configuration item. So, if you're tracking a PDA, you might want to have attributes about its screen size and whether it can connect to wireless networks, but those attributes wouldn't be any part of the attribute set describing a process document. Being able to describe each type of configuration item and relationship in separate terms allows much more flexibility to store information.

Variable granularity generally is defined following the hierarchy indicated in the scope definition. Consider the very simple scope described in Figure 3.1. For all the hardware devices, you've decided you want to capture the manufacturer. But documents and software don't have a manufacturer—they have a publisher instead. So manufacturer becomes an attribute only of the hardware node. By putting manufacturer there and not at both the servers and network nodes, you don't have to repeat, and you don't have to remember it if you add additional nodes (like workstations) under hardware. You can add more specific attributes, such as an SNMP community string, to leaf nodes like network. That indicates an attribute which applies only to that specific node. In defining granularity this way, you're leveraging a very powerful capability of object-oriented programming, called *inheritance*. Essentially, each category or "object" inherits all the attributes from its ancestors.

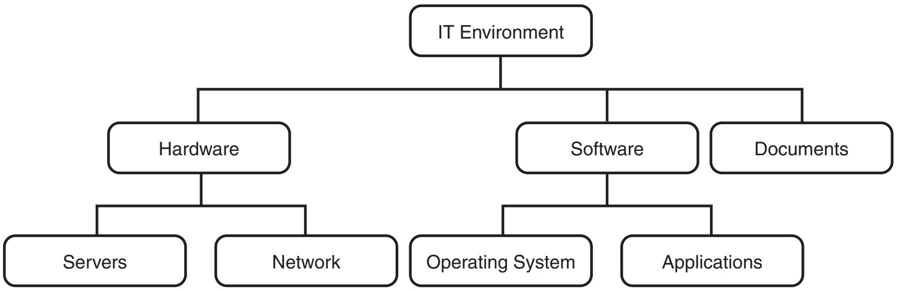

Figure 3.1 A simple scope to illustrate variable granularity.

This brings us to the most important feature of variable granularity. Attributes defined at the top impact *every* category in the CMDB. In other words, if you confined yourself to defining attributes only as this top level, you would be back to a fixed granularity. But why would you?

Although the discussion of attributes is largely around CIs, don't forget that relationships can also have attributes. In a variable granularity system, different kinds of relationships might have different attributes. For example, the relationship between a server and a network might include a "how attached" attribute, whereas the relationship between a business application and a server might have a "date deployed" attribute.

The downside of variable granularity is cost. It is more complex to define which attributes will be tracked for each category. This extra complexity will be reflected directly in the cost of data gathering. Having to stop and think about what to gather based on the type of device adds significant cost to your data gathering efforts. Variable granularity will also add to the costs of implementation, maintenance, and reporting. Clearly having more data will make your project more expensive over all.

Criteria Used to Define Granularity

You have to decide whether the extra cost of variable granularity is worth the extra value earned. The following criteria apply to either fixed or variable granularity; but as you read through them, you will most likely get a picture that variable granularity is almost always worth the extra costs. But whether you use fixed granularity or variable, these criteria can be combined with your business sense to settle on the best definition of granularity for your CMDB.

Get the Essentials

The first rule is to make sure that your granularity includes the essential elements that are critical to know about each configuration item. For many organizations, this will be the candidate list for a fixed granularity. The essentials will always include a unique identifier or name for the configuration item, the status of the item, who owns the item, where the item is located, and the category of the item. Your organization might also include other essentials, such as the organizational owner of each configuration item or the support person for each configuration item. The key is to determine which pieces of information are absolutely critical for your organization to accomplish the business goals defined for configuration management.

Remember that you must also define the attributes of relationships. Beyond the two configuration items participating in the relationship, the essential attributes might include a status indicator, a date the relationship was formed, and the name of the person or system that created the relationship.

Understand the Source

After the essentials, you begin working on what other data is available. In many cases, this involves looking at many potential sources for data. Do you have an existing asset management database? If so, this is an obvious source for data. How about IT inventory systems that automatically scan for hardware and software data? That can be an excellent source for configuration information. You will need some source for information about the people in your organization so that you can capture both owner and organizational information. Understanding which sources are available helps in determining which attributes you want to track. It makes perfect sense to track attributes that have a conveniently available source. If you can't think of how you would be able to get a data element, it might be best to leave it out of your plans. This is especially true in terms of relationships. It might be great to assign a "Date updated" attribute to the relationship between a workstation and its memory, but how will you find out the installation date of the memory on all your workstations?

Know Your Needs

When considering the level of granularity you need, consider what will be needed to convey configuration management information adequately to all of your stakeholders. If nobody needs to know whether the user is right or left handed, you won't need to gather and maintain that data element. But if your organization deals with highly classified data or government contracts, you may need to know the security classification of every configuration item. Careful consideration of your data needs will lead to the correct granularity for your efforts. If you have decided that granularity does not need to be fixed, you can then make the secondary decision concerning which set of configuration items you need any given data element to cover. It might be appropriate to know the support person for every shared printer, for example, but not necessary for each process document. Build your granularity based on what is needed by your organization.

This is another reminder to look at past projects and potential future projects to help you define the CMDB. What information would have made your recent projects more successful? Where has lack of information availability put projects in jeopardy? What upcoming projects will need specific pieces of information? Questions of this type will help you think more deeply about what attributes your configuration items and relationships should have.

Balance Knowledge and Effort

After understanding the needs of your organization, you need to balance those needs with the effort required to both initially gather and later manage the information. Some pieces of information are so important that they need to be managed at any cost. Other pieces of information are so costly to maintain that they are not worth managing. Although you may not be able to avoid the need to gather and maintain security classifications for every item, you should at least be able to understand what the cost will be to keep that information. A pharmaceutical company, for example, may have IT equipment connected to laboratory devices. Because of the oversight involved in drug testing, it may be necessary to have attributes describing those connections in great detail, even if it means a physical inspection of the interface on a regular basis.

In some cases, you can make trade-offs in precision to save cost. A perfect example is location—it might be very expensive to track each configuration item down to a specific room or grid location, but much less expensive to simply assign each item to a building. Determine whether the extra cost of adding attributes for floor, room, or grid is worthwhile. Organization can be another example where tracking to the division might be easy whereas tracking down to a department level might be very difficult and thus expensive. Use your knowledge of the organization to achieve the best balance between detailed knowledge and expensive effort, and then hold the line against "feature creep" by reminding your stakeholders that you've made the decision based on cost.

Avoid Dead Weight

The tendency in configuration management systems is to include too much detail. Just because you have scanning software that can get the speed at which your hard disks spin doesn't mean that you need to store this information in the configuration management system. Avoid choosing data items that simply add mass to the CMDB without adding value to your configuration management service. Each data source should be filtered before flowing across an automated interface into the database, and those filters should be created carefully to avoid keeping data that you haven't specified. Resist the strong temptation to include data items just because you can easily discover them. Keeping this kind of dead weight in your system leaves the impression that you're running a curiosity shop for technicians rather than a business information system.

Caution: Manual Attributes

At the opposite extreme, avoid using attributes that are too difficult to maintain because the only way to keep them is with manual effort. Of course, there will be cases where the information contained in the attribute is worth maintaining. But be aware that any time you have to gather and validate data manually, you are open to a whole set of errors that are difficult to detect. All attributes that are maintained manually should be flagged as suspect, and audits of your configuration management system should pay special attention to understanding how accurately those data items are being kept.

The procedures used to maintain the manual attributes must be flawless and reviewed frequently. The single largest cause of inaccuracy in the CMDB is attributes that are maintained by some person who is supposed to be following the right procedures. You can avoid this problem altogether by minimizing the manual attributes in your granularity definition.

Documenting Granularity

While a simple tree structure is sufficient for documenting the scope of configuration management, getting all the attributes documented requires a bit more complexity. The best way to capture your granularity decisions borrows from object-oriented programming and uses a class diagram. To create an effective granularity diagram, begin with a single box at the top, and include in this box the essential attributes that all configuration items will share. Remember that an attribute definition usually includes a type, a length or maximum value, and could include some validation criteria like a set of valid values or a minimum value. At this point, you have a simple list of attributes, and if your granularity is fixed, you are finished documenting all of the attributes.

If you have opted for variable granularity, however, your work is just beginning. Next, you list all categories that will have additional attributes beyond the essentials. If this involves many categories, you probably want to use the hierarchy that you determined in analyzing scope. For each second level box, put only the attributes that are incremental—don't repeat the common attributes in each box. Also, add only those attributes that will be common to every subcategory into this second level box. Remember the principle of inheritance. Continue to define attributes at the second level until you've finished. Move down to the next level and repeat the cycle until you've completed all attributes at all levels. A useful exercise is to double check that attributes make sense by starting at the bottom of the tree and listing all the attributes of a category by working up the tree to the top. If all of those attributes make sense for the category you're looking at, you've defined a reasonable granularity.

As an example of documenting attributes, let's consider a very simple hierarchy that has a root called configuration items with branches for hardware, software, people, documents, and intangibles. Under hardware you're going to have servers, workstations, network devices, and components. You've decided that all elements will have attributes for unique ID, name, description, status, and criticality, so those attributes are listed in the box for "Configuration Items" at the root of the tree. You next decide that hardware will also have manufacturer, model, and serial number fields, so those go in the "Hardware" box. Finally, you've decided to record physical memory, number of processors, and purpose for each server, so those attributes get listed in the "Servers" box. They key to this exercise is to make sure everyone understands that for each server entered into the CMDB, the full set of attributes is the union of all 3 boxes in the tree—so the total server has 11 different attributes, not just the 3 that are specific to the server class of CIs.

Defining the overall schema for your CMDB should be an iterative exercise. As attributes are defined and documented, you might find that it is more convenient to reorganize the hierarchy, thus changing the scope slightly. This may generate some new thoughts about span, and will certainly make you think about new relationship types. Also as new classes of CIs are defined, you might want to change the attribute list of parent classes to include more detail at a higher level of the tree. In general, you'll find that pushing attributes as high up the tree as possible will simplify both data collection and data maintenance as you go forward.

Don't be afraid to let the definition of the CMDB schema take a while. In practical terms, it can run in parallel with requirements gathering, and may take just as long. Just like requirements, scope, span, and granularity are best defined by teams of stakeholders, project team members, and sponsors. Documents should be reviewed, questioned, rewritten, and reviewed again to make sure that they are accurate and your organization is comfortable with the design that is taking form.

Having a documented schema is a major step toward implementing configuration management. In documenting requirements, you defined what would be implemented; but in working through this chapter, you have begun to define how it will be implemented. You'll recognize, however, that the schema is essentially static at this point. In the next chapter, we'll work on the process and see how the process sets the static structure in motion.