This is an excerpt from the book "***Cloud Computing and SOA Convergence in your Enterprise: A Step-by-Step Guide***" by David S. Linthicum ISBN 0136009220, Copyright 2010 Pearson Education, Inc. and contains the author's excerpts from the following chapters in the book:

- Working from Your Data to the Clouds
- Working from Your Services to the Clouds

Working from Your Data to the Clouds

We know that if we look at cloud computing, we need to understand where the data exists, gather information about the data (e.g., schema and metadata information), and apply business principles to determine which data flows where and why.

In short, implementing a cloud-based solution demands more than the movement and persistence of data within all systems in the problem domain. A successful solution requires that the enterprise also define how that information flows through it and how it is related to core services and core business processes.

In essence, what we are doing here is getting down to the details about the existing data, or data that is a part of new information systems. We are not analyzing this information as a component that will certainly reside in the clouds but to provide a base of understanding to consider if the information and the core systems could or should reside in the clouds and a good understanding of how to build those cloud-based systems if they do.

Identifying the Data

Unfortunately, there are no shortcuts to identifying data within an enterprise. All too often, information about the data—both business and technical—is scattered throughout the enterprise and is of a quality that ranges from "somewhat useful" to "You've got to be kidding me!"

The first step in identifying and locating information about the data is to create a list of systems in the problem domain. This list makes it possible to determine which databases exist in support of those systems. The next step requires determining who owns the databases, where they are physically located, relevant design information, and such basic information as brand, model, and revisions of the database technology.

Any technology that can reverse-engineer existing physical and logical database schemas will prove helpful in identifying data within the problem domains. However, while the schema and database models may give insight into the structure of the database or databases, they cannot determine how that information is used within the context of the application. Moreover, in some instances, that data is part of a packaged information system, such as an enterprise resource planning (ERP) package or a customer relationship management (CRM) package, and you must rely on that packaged application vendor to provide the information around the data and sometimes access to the data.

Creating the Data Dictionary

You need to create a data dictionary as a base to hold all of the metadata and other information about the data you analyze. We do these per system because in many instances, the systems are so different that it is difficult to find a common set of properties to track in the data dictionary. Information typically tracked includes

- The reason for the existence of particular data elements
- Ownership
- Format
- Security parameters
- The role within both the logical and physical data structures

There are tools, such as repositories, that provide prebuilt systems for creating and maintaining a data dictionary, and some databases have data dictionaries built into the DBMS. Do not get caught up in how the data dictionary will be maintained—you can transfer it from tool to tool. For now, just focus on listing the correct information.

Understand Integrity Issues

When analyzing databases for cloud computing, integrity issues constantly crop up. In order to address these, it is important to understand the rules and logic that were applied to the construction of the database. For example, will the application allow the update of customer information in a customer table without first updating demographics information in the demographics table?

Most middleware, such as the middleware that connects on-premise systems to clouds, take into account the structure or rules built into the databases being connected. As a result, there exists the very real threat of damage to the integrity of target databases when relationships are not properly understood and/or defined.

While some databases, including on-premise and cloud-based systems, do come with built-in integrity controls (such as stored procedures or triggers), most rely on the application logic to handle integrity issues on behalf of the database. Unfortunately, the faith implicit in this reliance is not always well placed. All too often, it is painfully naïve when you consider that your cloud computing system will be widely distributed, which makes it difficult to create a common control mechanism to protect database integrity.

The lack of integrity controls at the data level (or, in the case of existing integrity controls, bypassing the application logic to access the database directly) could result in profound problems. Architects and developers must approach this danger cautiously, making sure they do not compromise databases' integrity in their zeal to move to cloud computing.

Understand Data Latency

Data latency, the characteristic of the data that defines how current the information needs to be, is another property of the data that must be determined for the purposes of leveraging cloud computing. Such information allows the architects to determine when the information should be copied or moved to another enterprise system, such as on-premise to the clouds, and how fast.

While an argument can be made to support a number of different categories of data latency, for our purpose of defining architecture for cloud computing, there are really only three:

1.   Real-time
2.   Near-time
3.   Some-time

Real-time data is precisely what it sounds like: information that is placed in the database as it occurs, with little or no latency. Monitoring stock price information through a real-time feed from Wall Street is an example of real-time data. Real-time data is updated as it enters the database, and that information is available immediately to anyone or to any application requiring it for processing.

While zero-latency real time is clearly the goal, achieving it represents a huge challenge. In order to achieve something near zero latency, cloud computing implementation requires constant returns to the database, application, or other resource to retrieve new and/or updated information. In the context of real-time updates, database performance must also be considered; simultaneous to one process updating the database as quickly as possible, another process must be extracting the updated information.

Near-time data is information that is updated at set intervals rather than instantaneously. Stock quotes posted on the Web are a good example of near-time data. They are typically delayed 20 minutes or more, since the Web sites distributing the quotes are generally unable to process real-time data. Near-time data can be thought of as "good-enough" latency data. In other words, data only as timely as needed.

Although near-time data is not updated constantly, providing it still presents many of the same challenges as real-time data, including overcoming performance and management issues.

Some-time data is typically updated only once within a given time period. Customer addresses or account numbers are examples of some-time information. Within the context of an SOA using cloud computing architecture, the intervals of data copy or data movement do not require the kind of aggressiveness needed to accomplish real-time or near-time data exchange.

The notion of data typing goes well beyond the classification of the data as real-time, near-time, or some-time. It is really a complex process of determining the properties of the data, including updates and edit increments, as well as the behavior of the data over time. What do the

applications use the particular data for? How often do they use it? What happens with the data over time? These questions must be addressed in order to create the most effective SOA using cloud computing solution. Here is where you do that.

Data Cataloging

Data cataloging is really about formalizing the information we gather in the previous two steps, including the data dictionary. The difference is that the data dictionary is typically local to a single system or application, whereas the data catalog spans all systems in the problem domain.

A good data catalog should list all data elements within all systems, making sure to record

- Description
- Ownership
- System
- Format
- Security parameters
- Integrity parameters
- Dependencies

Once accomplished, it is possible to create an enterprisewide and/or cloudwide catalog of all data elements that may exist within the enterprise or on the cloud computing platforms. The resulting catalog becomes the basis of understanding that is needed to create the core information model— the foundation of our architecture and the basis for figuring out what will exist in the clouds and what will not.

For most medium- to large-sized enterprises, the creation of this data catalog is a massive undertaking. In essence, it demands the creation of the Mother of All Data Dictionaries, a complex directory that includes not only the traditional data dictionary information but also all of the information that is of interest to a cloud project, such as system information, security information, ownership, connected processes, communications mechanisms, and integrity issues, along with such traditional metadata as format, name of attribute, description, and so on.

While there is no standard for cataloging data within cloud computing projects, or any architecture project for that matter, the guiding principle stands clear: the more information, the better. The catalog will become both the repository for the new architecture to be built and the foundation to discover new business flows. It will also become a way to automate existing business flows within the enterprise.

It is an understatement to suggest that this catalog will be huge. Most enterprises will find tens of thousands of data elements to identify and catalog even while reducing redundancies among some of the data elements. In addition to being huge, the data catalog will be a dynamic

structure. In a very real sense, it will never be complete. A person, or persons, will have to be assigned to maintain the data catalog over time, assuring that the information in the catalog remains correct and timely and that the architects and developers have access to the catalog in order to create the SOA using cloud computing.

Building the Information Model

Once all the information about all the data in the enterprise is contained in the data catalog, it is time to focus on the enterprise metadata model, or what we call the information model. The difference between the two is sometimes subtle. It is best to think of the data catalog as the list of potential solutions to your architecture problem and to think of the information model as the final data architecture solution.

Let's review:

- • *Data dictionary:* What data exists currently, per system and/or application, and the definitions of that data (metadata)
- • *Data catalog:* Domainwide, sometimes enterprisewide, data dictionary
- • *Information model:* The final to-be state of the data architecture for our SOA using cloud computing and the jumping-off point for figuring out which data should reside on cloud computing platforms and which should reside on-premise

The metadata model defines all of the data structures that exist in the enterprise as well as how those data structures will interact within the architecture solution domain. While you can express your information model in any number of ways, it is usually best to create a logical model and a physical model. Keep in mind that we do not know the target platforms as of yet, so the models should be technology and platform independent at this point.

Logical Model

Just as with traditional database design methods, the enterprise metadata model used for information-level architecture can be broken into two components: the logical and the physical. And, just as with the former, the same techniques apply to the latter. Creating the logical model is the process of creating an architecture for all data stores that are independent of a physical database model, development tool, or a particular DBMS (e.g., Oracle, Sybase, Informix).

A logical model is a sound approach to an architecture project in that it allows architects and developers the opportunity to make objective information-level architecture decisions, moving from high-level requirements to implementation details. The logical data model is an integrated view of business data throughout the application domain or data pertinent to the architecture solution under construction, typically represented as an ERD, or entity relationship diagram. The

primary difference between using a logical data model for architecture versus traditional database development is the information source. While traditional development, generally speaking, defines new databases based on business requirements, a logical data model arising from an architecture project is based on existing databases.

Physical Model

The myriad of database types in any given enterprise minimizes the importance of the physical enterprise model because with so many database types, the physical model will rarely be used. The reason is clear: there is simply no clear way to create a physical model that maps down to object-oriented, multidimensional, hierarchy, flat files, and relational databases all at the same time. However, if those databases are to be integrated, some common physical representation must be selected. Only then can the model be transformed as required.

Our discussion of the physical model is only for those times when it is possible to map the logical to the physical—that is, when an enterprise uses a homogeneous database approach, usually all relational. The input for the physical model is both the logical model and the data catalog. When accessing this information, consider the data dictionary, business rules, and other user processing requirements.

Importance of Data with SOA using Cloud Computing

We begin with the data because it is the foundation for most information systems and a good way to define what these systems do before we potentially relocate them to the clouds. However, the concepts and activities presented here are not at all new; they map back to core architectural activities, including SOA. The purpose here is to understand the existing state of the data, define the data at a detailed level, and then define the final to-be data architecture that allows us to figure out which data should physically reside on-premise or on cloud computing platforms. However, this is all about doing SOA with the new architectural options of cloud computing.

What is new is that, for our target architecture, some of the information systems will not be under our direct control. However, not much more than that changes. We still need to define how the data is structured, as well as relationships between the data, integrity, security, and all things in between. The big difference is that eventually we need to figure out how to manage data between on-premise and cloud-based systems, making sure that they function as if they exist within the same data center. That should be the objective.

Many times when moving to new concepts such as cloud computing, architects and application designers have a tendency to neglect the fundamentals of the architecture, including the data. You do this at the risk of architecture failure and the failure of any instances of cloud-based systems you build from that architecture. Those who do not grasp the underlying data will not be able to define the information systems that they will need, clouds or no clouds.

Working from Your Services to the Clouds

Services are not specific to cloud computing or to SOA, for that matter. They are a way to describe how we can access specific behaviors and data. Services are typically exposed out of an application and provide a way to access the value of the application, both behavior and data, using a well-defined interface.

We can think of services as collections of functional behaviors, each callable individually or as a group. You can mix and match services to form composite applications, or you can combine many services into a single composite application. This approach provides flexibility for the architecture, including the ability to leverage cloud computing resources and the ability to change the architecture as needed by the business.

In the case of cloud computing, this means the ability to run applications and their services, on-premise or within the clouds, and have those services available to any application, on-premise or in the clouds. This is an important step in looking at candidate applications and services that may make sense to move to cloud computing platforms. The idea is to address them as services and make them platform and location independent. Can you invoke them from any platform, cloud-based or on-premise? That is the goal: defined services that may exist on cloud-based or on-premise systems and are accessible from both systems.

To make this a bit easier to understand, let's walk through this process. Typically with all services and systems hosted on-premise and nothing yet in the cloud or clouds. Each system has a set of services that it can expose (represented by the yellow circles), thereby providing interfaces to other applications by using services—the basic idea behind a SOA.

Thus, at a high level, the process is as follows:

1. Understand the existing as-is architecture.
2. Identify the existing services within the architecture.
3. Document and list those services within a directory.
4. Define the to-be architecture, including the use of cloud computing.

Once we have a service-level understanding of the problem domain, we can identify systems and services that are good candidates for relocation into cloud platforms and those that can stay on-premise. (We get into how that relocation occurs in subsequent chapters.) The idea is that we will relocate the systems to find a more cost-effective way to process the same applications and services but never give up the ability to leverage these systems and services. Services are tyically location and platform independent. This means that no matter where the services exist, on-premise or cloud-based, they are accessible as if they were local.

In many architectural systems, we would create new services as part of the services model. My book, "Cloud Computing and SOA Convergence in your Enterprise: A Step-by-Step Guide," looks at methods to extend an SOA to cloud computing if that approach make sense. Thus, we limit our discussion to identifying existing services, documenting those services, and relocating those services if cost justified. Keep in mind that you could be building new services as part of this process.

Again, the ability to address all of these systems through services provides the architect with the ability to access the underlying application behavior and information as if the application and services were native and local. The use of services provides location independence, meaning you can leverage services no matter where they exist. A single application could be made up of dozens of services hosted in a dozen different locations, on-premise and in the clouds. This is why we use services within the context of a widely distributed system, including an SOA using cloud computing.

Since they all expose information and behaviors as services, which typically use a common and standard interface such as Web Services, they are location independent (cloud or on-premise), platform independent, programming language independent, and user interface independent—that is, if they are properly created.

Thus, we have the purpose of this article: the need to deal with all of our IT assets as services. We must define them with enough detail to figure out how they work and play well within our SOA and how some of those services may reside on cloud platforms if we need them to live there. Assuming that you are an enterprise or SOA architect, we provide you with enough information about the "how" that you can be productive in your own cloud computing replatform project.

The service model is the listing of all services from the candidate services list that are relevant and thus selected as services for our architecture. Moreover, they are decomposed and ordered so all services dependent on other services are understood, from the highest level to the lowest. For instance, the service Rent a Car, is made up of Reserve a Car, Pick Up Car, Drop off Car. However, the service Reserve a Car is made up of Identify Customer, Select Car, and Enter Reservation. You get the idea. In essence, we define the services as they exist in a hierarchy, from the highest to the lowest level, not forgetting to define the data that is bound to those services.

While decomposing services should be considered an architectural exercise, this process can be done by anybody who is looking at cloud computing as an architectural option, including developers and project leaders. It is really some quick analysis to determine a basis of understanding to look at cloud computing options, not a complete and rigorous architectural process as defined by some SOA and enterprise architecture approaches and methodologies.

Now that we know what we are looking for—basic concepts, the general process, and how to proceed—let's spend the remainder of the chapter getting into some useful details.

Defining Metaservices

The next step is to understand and collect information about services in the problem domain. We can call these *metaservices*, or data about services. Then we will talk about creating a services directory, where the services are documented in detail. This information goes into the services directory leveraged to understand the requirements of our cloud computing architecture.

While we certainly understand the use of metadata, or data about data, lately in the modern world of application integration, we have seen the same need to better define information around services existing in an SOA or an SOA using cloud computing. If you think about it, it is a logical step. While we need to keep track of semantics, validation constraints, formats, and so on, around data, we have the same needs in terms of how we manage, understand, and track services.

Behavior (the core value of services) is a notion very different from information. Behavior is much more "living and breathing" than data and provides not only information but dynamic and complex services around that data. This is why we leverage services as well as data for integration. There is, however, no standard approach to describing services at a higher level. With the public nature of services these days, as well as the rate of service replication and abstraction, this should be on the top of our to-do list.

With the advent of Web Services, we now have thousands, sometimes tens of thousands, of services under our management (private services). To make matters even more complex, we also have to consider services that are out of our direct control: those shared from cloud computing platforms (public services). While we do have some basic information about them, as defined by standards such as Web Services Description Language, we really need a more complete set of information surrounding the services. This information should include the purpose, interfaces, parameters, rules, logic, owner, semantics, included services, and other important data. We can define these attributes within the services directory, which we define next.

Creating the Services Directory

Using the previously defined concept of metaservices, let's now explore how to document those services for the purposes of our SOA using cloud computing architecture. Earlier in this chapter, we defined the need to create a list of candidate services as well as to create a final services model, using our case study as an example. Now we look at the concept of a services directory, which is basically a directory of services that you discover, identify, and define as you gain a service-level understanding of your problem domain, and look to replatform some of these

services for cloud computing. We know by now that we need the following in our services directory:

- Semantics
- Purpose
- Authentication
- Dependencies
- Service levels

However, we need to add some other important categories, including

- Owner
- Enabling technology
- Programming model
- Performance attributes
- Data validation
- Services leveraged within this service
- Where this service is leveraged
- Function points/object points
- Flow diagram
- Structure charts
- Interface definitions
- Code revisions
- Test plans
- Test results
- Development tools (include version)

What exactly is a service directory? It is a database of information, a repository, about service, including the properties just described. Although many experts view repositories primarily as a part of the world of application development and data warehousing, they do not question their value to the world of enterprise architecture and our ability to track services that extend out to cloud-delivered platforms.

We create the services directory as a way to drive through our process, including the creation of the services and information model, as well as the final service model. Creating the directory is a way to understand the services and the information bound to the services in detail. The services directory becomes the starting point for the SOA repository. It can be passively defined, not dynamically interacting with the services, or actively defined, meaning that the repository is in direct interaction with the services, both on-premise or cloud delivered.

The goal is to provide a sophisticated directory that can keep track of a good deal more than the rudimentary information (such as directory data). It should track more sophisticated information about services in the problem domain. The directory should provide all the information required by the architect and programmer to locate any piece of information within the on-premise or cloud platforms and to link it to any other piece of information. The directory must be the master directory for the entire architecture spanning on-premise and the clouds, which eventually becomes the registry.

In more sophisticated cloud computing solutions, the services directory is becoming the *axis mundi*, able to access both the source and target systems in order to discover necessary information (such as metadata and available business processes). Engaged in this "autodiscovery," the SOA solution can populate the directory with this or any other information that may be required. Ultimately, the services directory will become the enterprise metadata repository, able to track all systems, services, and information, on-premise or on the cloud delivered platforms.

The value of a services directory should be clear. With the directory as a common reference point for all connected processes, services, and databases, integrating data and services is straightforward. The services directory can also track the rules that the architect and developer apply within the SOA problem domain.

The Need for a Service-Level Understanding

The core concept you want to take away is that we need to understand of the core services that exist in our problem domain in order to make the right calls around which services should reside on-premise and which should exist within a cloud computing platform.

Through this exercise, we have a much better understanding of the interworkings of our applications: what they do, the information they process, which are good candidates for cloud computing, and which are not. Those who choose to skip this process will find that services exist on the wrong platform, either cloud or on-premise, and the new cloud computing architecture is likely to fail.

While this seems like a lot of work, it really is only a quick survey and understanding of the present services. While we recommend basic concepts and basic approaches, the needs of your IT environment are going to be unique and may require slightly different approaches. As long as the objective of having a complete understanding of the services within the problem domain are achieved, the way you go about doing that project is up to you.