Chapter 5

Understanding Design Principles

Principles help shape every aspect of our world. We navigate ourselves through various situations and environments, guided by principles we learned from our family, society, and from our own experiences. Historically, many parts of the IT world encouraged the use of design principles so that when you did something, you would "do it right" on a consistent basis. Often, though, their use was optional or just recommended. They were viewed more as guidelines than standards, providing advice that we could choose to follow.

When moving toward a service-oriented architecture, principles take on renewed importance primarily because the stakes are higher. Instead of concentrating on the delivery of individual application environments, we usually have a grand scheme in mind that involves a good part of the enterprise. A "do it right the first time" attitude has therefore never been more appropriate. SOA projects have the potential to shape and position solution logic in ways that can significantly transform an enterprise. We want to make sure we steer this transformation effort in the right direction.

As documented in Chapter 4, the design principles explored in this book establish a paradigm with many roots in previous computing generations. None of them are really that new. What is distinct about service-orientation is which of these existing principles have been included and excluded—that and the high-minded goals promised by its successful application.

5.1 Using Design Principles

The *Design Fundamentals* section of Chapter 3 formally defined the term "design principle" and determined that it essentially is "a recommended guideline for shaping solution logic with certain goals in mind." We subsequently covered the following list of service-oriented computing benefits:

- Increased Intrinsic Interoperability
- Increased Federation
- Increased Vendor Diversification Options
- Increased Business and Technology Domain Alignment

- Increased ROI
- Increased Organizational Agility
- Reduced IT Burden

These benefits represent the most common strategic goals associated with service-orientation. The application of the eight principles explored in this book results in the realization of very specific design characteristics, all of which support these goals.

We therefore need to ensure that the principles are effectively applied. Following is a set of best practices for getting the most out of the design principles in this book.

Incorporate Principles within Service-Oriented Analysis

Because we have labeled the principles in this book as *design* principles, there is a natural tendency to focus on their application during the design stage only. However, because of the unique form of analysis carried out as part of the common SOA delivery lifecycle, it can be highly beneficial to begin working with a subset of the principles during the analysis phase.

While iterating through the service modeling process of a typical service-oriented analysis, we are tasked with defining a conceptual blueprint for the inventory of services we will eventually be designing and building. This provides us with an opportunity to begin conceptually forming some of the key service design characteristics ahead of time.

Of the eight service-orientation design principles, the following three are most commonly incorporated within the service modeling process:

- *Service Reusability*—Reusability considerations are highly relevant to defining the inventory blueprint because they help us group logic within the contexts of proposed agnostic service candidates and further encourage us to refine the definition and functionality behind agnostic capability candidates.
- *Service Autonomy*—One of the goals of the information gathering steps that comprise the parent service-oriented analysis process is to determine where, within an enterprise, autonomy will ultimately be impacted. Knowing this in advance allows us to adjust service candidate granularity and capability candidate grouping in response to practical concerns. This prevents the inventory blueprint from becoming too abstract and out of touch with the realities of its eventual implementation.

- *Service Discoverability*—Although service meta data can be added to a contract at any time prior to deployment, the analysis stage enables us to leverage the expertise of subject matter experts that will not be participating in subsequent project phases. This is particularly relevant to the definition of business services. Analysts with a deep insight into the history, purpose, and potential utilization of business logic can provide quality descriptions that go far beyond the definition of the candidate service contract.

As illustrated in Appendix B, a separate step dedicated to applying select service-orientation principles is part of a standard service modeling process.

FOR EXAMPLE

A US-based shipping company created their own expanded variation of the service modeling process documented in Appendix B. Instead of bundling service-orientation considerations into one step, it included the following separate steps:

- *Business Reusability Survey*—A step during which representatives from different business domains were questioned as to the applicability of a given service that was being modeled. Those surveyed were asked to provide feedback about how any agnostic service could be potentially extended in support of business processes that resided in their domains.
- *COTS Evaluation*—This was carried out for each service capability candidate required to encapsulate functionality that resided in an existing COTS environment. It provided insight into potential autonomy constraints for some of the planned services.
- *Service Profile Copyedit*—This was a step toward the end of the modeling process during which the service profile document was refined by one of the on-staff technical writers (which is also a best practice discussed in Chapter 12).

Each of these steps was carried out by different individuals, all part of the service modeling project team.

Incorporate Principles within Formal Design Processes

The key success factor to leveraging service-orientation design principles is in ensuring that they are applied consistently. When services are delivered as part of different projects that are perhaps even carried out in different geographical locations, there is a constant danger that the resulting service inventories will be comprised of incompatible and misaligned services, varying in both quality and completeness.

Design synchronicity is important to achieving the harmonization and predictability required to ultimately compose services into different configurations. Establishing formal service design processes that exist as part of the organization's over-arching project delivery methodology requires that project teams give serious thought as to how each principle can or should be applied to their planned services.

The design processes listed in Appendix B have steps dedicated to applying service-orientation principles. These processes can be further customized and expanded to incorporate a dedicated step for each principle.

FOR EXAMPLE

The aforementioned shipping company formalized service design processes that included separate steps for applying Service Reusability, Service Autonomy, and Service Composability principles. The remaining principles were also incorporated in the design processes but grouped together with other design considerations.

The Service Composability step actually introduced a sub-process during which service contracts were combined into a variety of composition configurations in order to assess data exchange compatibility.

Establish Supporting Design Standards

Design principles are design guidelines, essentially recommended approaches to designing software programs. Due to the importance of creating consistent programs (services) in support of service-oriented computing, it is highly recommended that design principles take on a larger, more prominent role.

Once an organization has determined to what extent it wants to realize service-orientation, design standards need to be put in place in full support of the consistent application of these design principles. This often leads to the principles themselves forming the basis for multiple design standards.

Either way, if you are expecting to attain meaningful strategic benefit from a transition toward SOA, design standards need to be in place to ensure the consistent realization and proliferation of service-orientation across all affected services.

FOR EXAMPLE

An enterprise design specification for a government agency contained upwards of 300 separate design standards, many of which were directly or indirectly defined in support of service-orientation.

One of these standards, for example, required that all XML schema definitions support null values by allowing an element to exist zero or more times (via the `minOccurs="0"` attribute setting). If the element was not present, its value was considered to be null.

This simple design standard ensured that null values were consistently expressed across all XML document instances, thereby supporting the Service Contract Standardization and Service Reusability principles and also avoiding some of the negative coupling types described in Chapter 7.

Apply Principles to a Feasible Extent

Each of the eight service-orientation design principles can be applied to a certain extent. It is rare that any one principle will be fully and purely realized to its maximum potential. A fundamental goal when applying any principle is to implement desired, corresponding design characteristics consistently within each service to whatever measure is realistically attainable.

The fact that principles are always implemented to some extent is something we need to constantly keep in the back of our minds as we are working with them. For example, it's not a matter of whether a service is or is not reusable; it's the degree of reusability that we can realize through its design that we are primarily concerned with.

Most of the chapters in this book explore specific measures to which a principle can be applied and further provide recommendations for how these levels can be classified and documented. Additional supporting practices are provided in Chapter 15.

SUMMARY OF KEY POINTS

- Design principles can be effectively realized by applying them as part of formal analysis and design processes.
- Design principles can be further applied consistently by incorporating them into official design standards.
- Every principle can be applied to a certain extent.

5.2 Principle Profiles

Each of the chapters in Part II contains a section that summarizes a design principle within a standard profile table. Provided here are brief descriptions of the fields within the standard profile table:

- *Short Definition*—A concise, single-statement definition that establishes the fundamental purpose of the principle.
- *Long Definition*—A longer description of the principle that provides more detail as to what it is intended to accomplish.
- *Goals*—A list of specific design goals that are expected from the application of the principle. Essentially, this list provides the ultimate results of the principle's realization.
- *Design Characteristics*—A list of specific design characteristics that can be realized via the application of the principle. This provides some insight as to how the principle ends up shaping the service.
- *Implementation Requirements*—A list of common prerequisites for effectively applying the design principle. These can range from technology to organizational requirements.
- *Web Service Region of Influence*—A simple diagram that highlights the regions within a physical Web service architecture affected by the application of the principle. The standard Web service representation (consisting of core service logic, messaging logic, and the service contract) is used repeatedly. Red shaded spheres indicate the areas of the Web service the principle is most likely to affect. The darker the shading, the stronger the potential influence.

Chapters are further supplemented with the following sections:

- *Abstract*—An introductory section that explains each design principle outside of the context of SOA. This is a helpful perspective in understanding how service-orientation positions design principles. The title of this section incorporates the name of the principle as follows: *[Principle Name] in Abstract.*
- *Origins*—A section that establishes the roots of a given principle by drawing from past architectures and design approaches. By understanding the history of each design principle, it becomes clear how service-orientation is truly an evolutionary paradigm. The format of this section's title is as follows: *Origins of [Principle Name].*

- *Levels*—As explained earlier in the *Apply Principles to a Feasible Extent* section, each principle can be realized to a certain degree. Most chapters provide suggested labels for categorizing the level to which a principle has been applied, primarily for measuring and communication purposes. The section title is structured as follows: *Levels of [Principle Name].*
- *Service Design*—Several chapters explore supplementary topics that highlight additional design considerations associated with a principle. These are found in a section called *[Principle Name] and Service Design*. (Note that the following *Service Models* and *Relationships* sections exist as sub-sections to the *Service Design* section.)
- *Granularity*—Whenever the application of a design principle raises issues or concerns regarding any of the four design granularity types (as explained in the upcoming *Principles and Design Granularity* section) a separate section entitled *[Principle Name] and Granularity* is added.
- *Service Models*—Where appropriate, a principle's influence on the design of each of the four primary service models (entity, utility, task, and orchestrated task) is described in a section titled *[Principle Name] and Service Models*.
- *Relationships*—To fully appreciate the dynamics behind service-orientation, an understanding of how the application of one principle can potentially affect others is required. Each chapter provides a section titled *How [Principle Name] Affects Other Principles* wherein inter-principle relationships are explored.
- *Risks*—Finally, every chapter dedicated to a design principle concludes with a list of risks associated with using or abstaining from the use of the principle. This list is provided in a section titled *Risks Associated with [Principle Name]*.

Every effort was made to keep the format of the next eight chapters consistent so that aspects of individual principles can be effectively compared and contrasted.

NOTE

Principle profiles should not be confused with service profiles. The former represents a regular section format within the upcoming chapters, whereas the latter is a type of document for recording service meta details. Service profiles are described in Chapter 15.

SUMMARY OF KEY POINTS

- Each chapter summarizes a design principle using a standard profile section.
- Design principles are further documented with additional sections that explore various aspects of their origin and application.

5.3 Design Pattern References

The eight design principles in this book were documented in alignment with an SOA design pattern catalog published separately in the book *SOA: Design Patterns,* another title that is part of the *Prentice Hall Service-Oriented Computing Series from Thomas Erl.* This book expresses service-orientation through a fundamental pattern language and provides a collection of advanced design patterns for solving common problems.

Because these two books were written together, there is a strong correlation between the utilization of design principles and select design patterns that provide related solutions in support of realizing service-orientation. Fundamental design patterns are often tied directly to the design characteristics established by a particular design principle, whereas advanced design patterns more commonly solve problems that can be encountered when attempting to apply a principle under certain circumstances.

Throughout the chapters in Part II, references to related design patterns are provided. These references are further summarized in Appendix C.

5.4 Principles that Implement vs. Principles that Regulate

Before exploring the design principles individually, it is worth positioning them as they relate to the realization of physical service design characteristics. On a fundamental level we can group principles into two broad categories:

- Principles that primarily result in the implementation of specific service design characteristics.
- Principles that primarily shape and regulate the application of other principles.

The following principles fall into the first category:

- Standardized Service Contract
- Service Reusability

- Service Autonomy
- Service Statelessness
- Service Discoverability

As explained throughout Chapters 6, 9, 10, 11, and 12, the application of any one of these principles results in very specific design qualities. Some affect the service contract, while others are more focused on the underlying service logic. However, all result in the implementation of characteristics that shape the physical service design.

This leaves us with the remaining three that fall into the "regulatory" category:

- Service Loose Coupling
- Service Abstraction
- Service Composability

After studying Chapters 7, 8, and 13, it becomes evident that while these principles also introduce some new characteristics, they primarily influence how and to what extent the service design characteristics associated with other principles are implemented (Figure 5.1).

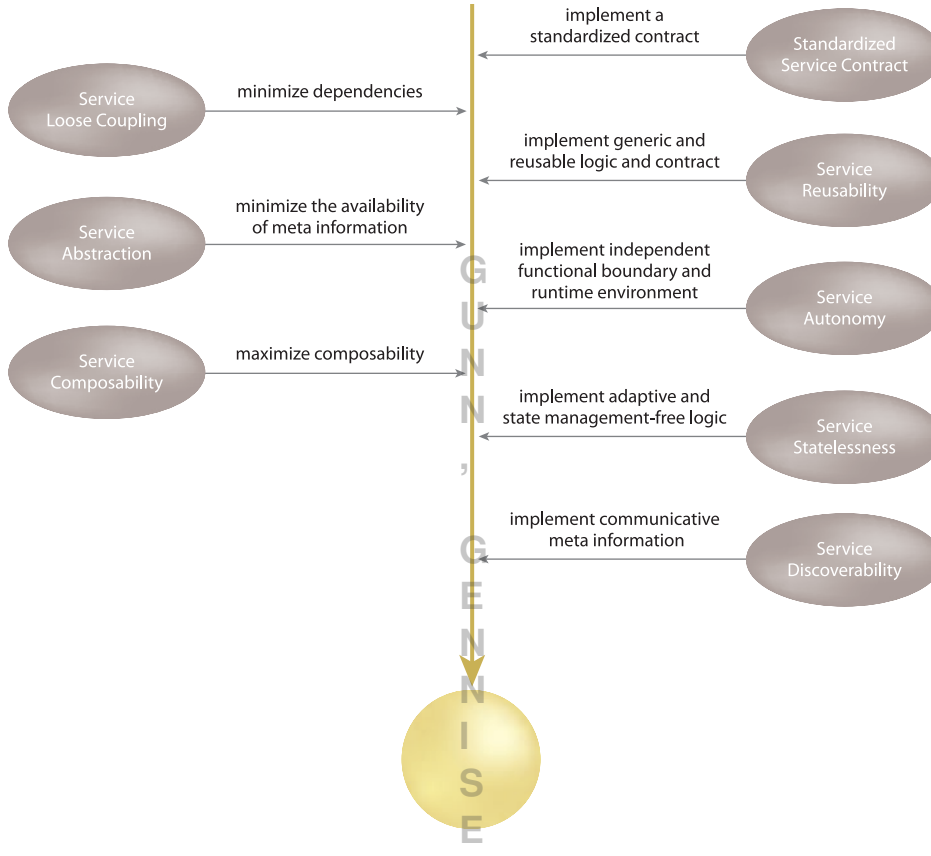

Figure 5.1

While the principles on the right-hand side want to add specific physical characteristics to the service design, the principles on the left act as regulators to ensure that these characteristics are implemented in a coordinated and appropriate manner.

Furthermore, each chapter explores how principles inter-relate. Specifically, the manner in which a design principle affects the application of others is documented. Figure 5.2, for example, provides an indication as to how two of the "regulatory" principles relate to each other.

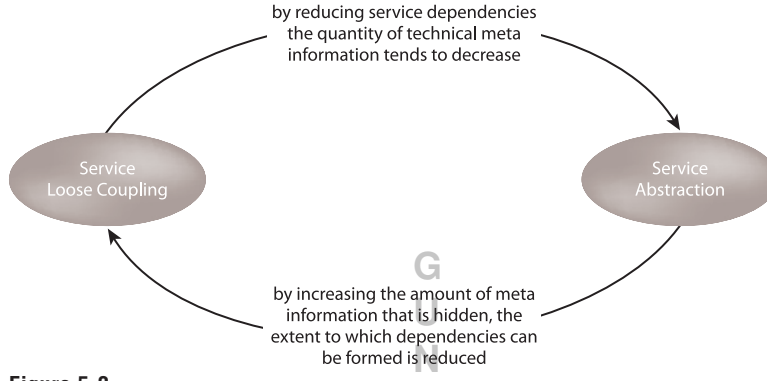

Figure 5.2
The Service Loose Coupling and Service Abstraction principles share a common dynamic in that the application of each supports the other.

SUMMARY OF KEY POINTS

- Five of the eight design principles establish concrete service design characteristics.
- The remaining three design principles also introduce design characteristics but act more as regulatory influences.

5.5 Principles and Service Implementation Mediums

Service logic can exist in different forms. It can be implemented as the core logic component within a Web service, as a standalone component with a public interface, or even within an event-driven service agent. The choice of implementation medium or format can be influenced by environmental constraints, architectural considerations, as well as the application of various design patterns.

Service-orientation design principles shape both service logic and service contracts. There is an emphasis on the Web service medium because it provides the most potential to apply key principles to the greatest extent. For example, contract-related principles may not apply as much to logic encapsulated within an event-driven service agent. This does not make the logic any less service-oriented; it only limits the principles that need to be taken into account during its development.

"Capability" vs. "Operation" vs. "Method"

To support the on-going distinction between a service in abstract and a service implemented as a Web service, separate terms are used to refer to the functions a service can provide.

A *service capability* represents a specific function of a service through which the service can be invoked. As a result, service capabilities are expressed within the service contract. A service can have capabilities regardless of how it is implemented.

A *service operation* specifically refers to a capability within a service that is implemented as a Web service. Similarly, a *service method* represents a capability that is part of a service that exists as a component.

Note that as mentioned early on in Chapter 3, when the term "capability" is used in this book, it implicitly refers to capabilities expressed by the service contract. If there is a need to reference internal service capabilities that are not part of the contract, they will be explicitly qualified as such.

5.6 Principles and Design Granularity

The term "granularity" is most commonly used to communicate the level of (or absence of) detail associated with some aspect of software program design. Within the context of service design, we are primarily concerned with the granularity of the service contract and what it represents.

Within a service, different forms of granularity exist, all of which can be impacted by how service-orientation design principles are applied. The following sections document four specific types of design granularity, three of which are further referenced in Figure 5.3.

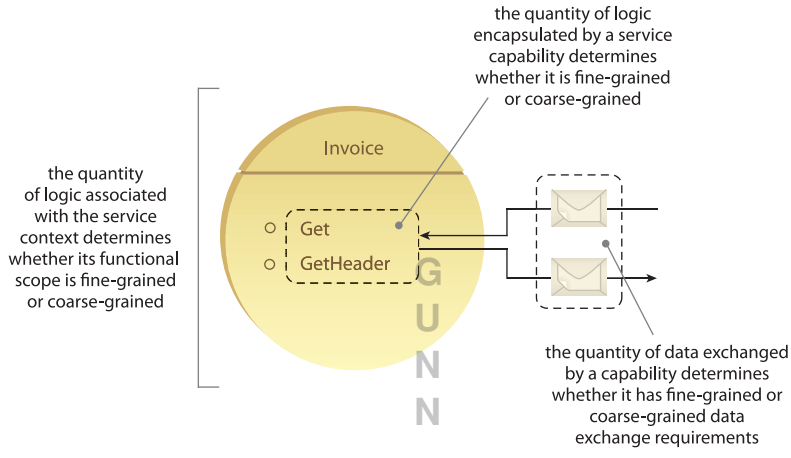

Figure 5.3

In this example, an Invoice entity service will tend to have a coarse-grained functional scope. However, it is exposing both coarse-grained (Get) and fine-grained (GetHeader) capabilities. Furthermore, because the GetHeader capability will return less data than the Get capability (which returns an entire invoice document), the GetHeader capability's data granularity is also considered fine.

Service Granularity

The granularity of the service's functional scope, as determined by its functional context, is simply referred to as *service granularity*. A service's overall granularity does not reflect the amount of logic it currently encapsulates but instead the quantity of potential logic it could encapsulate, based on its context. A coarse-grained service, for example, would have a broad functional context, regardless of whether it initially expresses one or ten capabilities.

Capability Granularity

Capability granularity represents the functional scope of a specific capability as it currently exists. As a rule of thumb, a fine-grained capability will have less work to do than a coarse-grained one.

Data Granularity

The quantity of data a capability needs to exchange in order to carry out its function represents its level of *data granularity*. There has been a tendency for services implemented as Web services to exchange document-centric messages—messages containing entire information sets or business documents. Because the quantity of data is larger, this would be classified as coarse-grained data granularity.

Document-centric messages are in sharp contrast to traditional RPC-style communication, which typically relies on the exchange of smaller (fine-grained) amounts of parameter data.

Constraint Granularity

The amount of detail with which a particular constraint is expressed is referred to as a measure of *constraint granularity*. The schema or data model representing the structure of the information being exchanged by a capability can define a series of specific validation constraints (data type, data length, data format, allowed values, etc.) for a given value. This would represent a fine-grained (detailed) constraint granularity for that value, as opposed to a coarse-grained level of constraint granularity that would permit a range of values with no predefined length or format restrictions, as represented by the first element definition in Example 5.1.

```
<xsd:element name="ProductCode" type="xsd:string"/>

<xsd:element name="ProductCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
      <xsd:maxLength value="4"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="ProductCode">
<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9]{4}"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

Example 5.1

Three variations of the same XML schema element definition. The first is clearly a coarse-grained constraint because it allows the product code to exist as an open-ended string value. The second is less coarse-grained because it restricts the product code length to one to four characters. The last element is a fine-grained constraint because it dictates that the product code must be four characters and that each character be a number between 0 and 9.

Constraint granularity can be associated with individual parameters processed by a capability or with the capability as a whole. For example, the same capability may accept

a body of input data comprised of two separate values, one of which is subject to fine-grained constraints and the other of which is validated against a coarse-grained constraint. The three code samples in Example 5.1 could alternatively exist as different types with different names but with the same constraints and as part of the same service capability (or Web service operation).

It is also important to note that constraint granularity is generally measured in relation to the validation logic present in the service contract only. This measure therefore excludes validation constraints that may be applied by the underlying service logic. Whereas a capability defined within a contract may have coarse constraint granularity, the actual capability logic may apply more fine-grained constraints after input values have been validated against the service contract.

NOTE

There are no rules about how forms of granularity can be combined. For example, it would not be uncommon for a coarse-grained service to provide fine-grained capabilities that exchange coarse-grained data validated against fine-grained constraints.

Sections on Granularity Levels

There is no one principle that dictates granularity levels for a service design. Instead, several service-orientation principles impact the various types of granularity in different ways. Those chapters that cover principles affecting design granularity typically address this issue within the standard *[Principle Name] and Service Design* section.

SUMMARY OF KEY POINTS

- Service granularity refers to the functional scope of the service as a whole, as defined by its functional context.
- Capability granularity refers to the functional scope of a specific capability.
- Data granularity refers to the volume of data exchanged by a service capability.
- Constraint granularity refers to the level of detail to which validation logic is defined for a particular parameter or capability within the service contract.