THREE

RM-ODP ARCHITECT'S PRIMER

In this introductory chapter to the Reference Model of Open Distributed Processing (RM-ODP), highlights of RM-ODP are discussed. This chapter introduces the motivation for RM-ODP, and discusses why the architect should consider its use in architecting a system. Following chapters will discuss these topics in more depth.

This chapter discusses:

- More about RM-ODP for an architect
- Overview of RM-ODP architecting techniques
- How to use RM-ODP in creating an architecture specification
- How RM-ODP relates to a distributed processing system and to an architecture
- What knowledge is prerequisite

3.1 MORE ABOUT RM-ODP

RM-ODP is an internationally agreed-upon object-based architecture standard for use in architecting distributed systems. The standard provides mechanisms to architect distributed processing software systems and distributed information, and to support the integration and interoperation of applications in a reliable and con-

sistent manner, across a heterogeneity of enterprise rules, software, protocols, programming languages, component-based frameworks, computers, and networks. RM-ODP provides a rich, precisely defined set of distributed processing concepts for architecting systems that depend upon distributed processing, as well as a rich set of techniques to use in specifying the architecture.

RM-ODP provides:

- A guide for architects to specify distributed software systems; this is based on object modeling, which is relevant to the practices of systems architects
- A set of precise concepts and structuring rules used for development of a system specification
- A set of separate interrelated viewpoint specifications used for specification of open distributed systems
- A system conformance testing framework
- A distribution transparency framework
- A set of functions specific to the system infrastructure that supports the capabilities of the distribution transparencies
- An overall framework for development of additional standards under the RM-ODP initiative that are related to open distributed processing

A system can be an enterprise-wide system of systems, a large information processing system, or a major component of another system, such as data. Any of the rules of architecting apply to any such "system."

Consider the example in Figure 3.1. With RM-ODP, all rules could apply to the Data Broker component. Once specified, it becomes an object in the Org B architecture specification. Once Org B is specified, it becomes an object in the Enterprise architecture specification. These levels of abstraction and concepts of composition and refinement are exceedingly important in RM-ODP and pervade all aspects of architecting, as will be discussed later.

Capturing the needs of the stakeholders of the information processing system in such a way as to reflect these needs into system requirements has long been a problem in technology transfer of knowledge. That is, the stakeholders[1] state their requirements in a language particular to a functional or organizational domain (e.g., finance). The system designers (system engineers, software engineers, architects, implementers, and testers) discuss system capabilities in terms of information technology language. Often these needs and capabilities are misunderstood, due to differences in the language of discourse. RM-ODP facilitates communication by capturing, in its *enterprise* model, aspects particular to an enterprise. Additionally, RM-ODP provides all of the concepts and rules needed to transform these aspects into other models applicable to the system semantics, information, and processing.

1. *Stakeholder* is a term used here to represent any customer, user, owner, administrator, acquisition authority, or program manager.

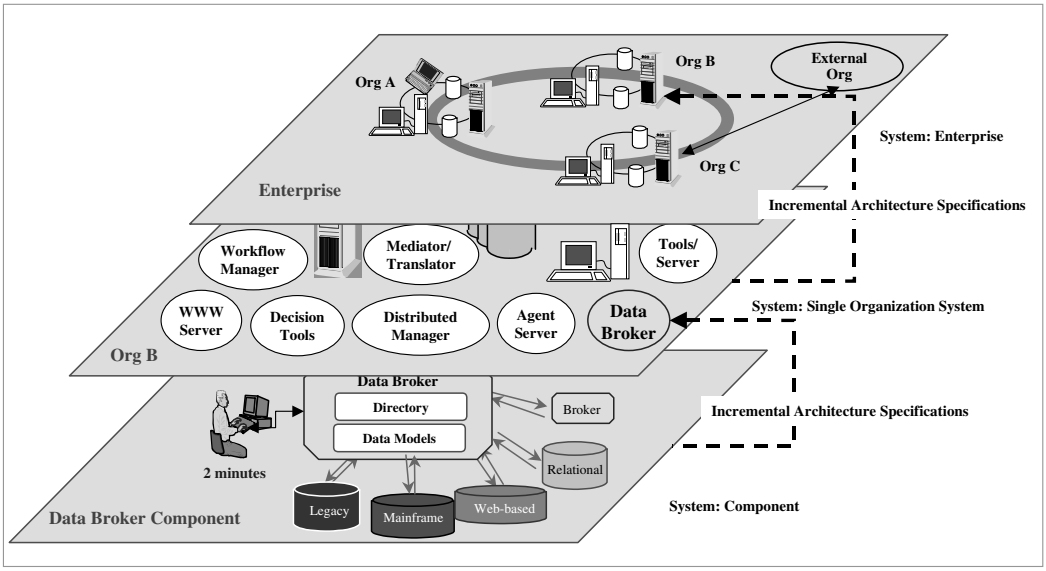

FIGURE 3.1 Different Kinds of "Systems"

RM-ODP describes how to capture the needs of the stakeholders, how to capture the information processing semantics, how to specify the components, interactions, and constraints of the system, and how to select products and technologies to realize the system. These are all different areas of focus on the system. These are all described in RM-ODP in a consistent manner so that decisions made in one area of focus are reflected in other areas.

RM-ODP uses object-based constructs to help system architects specify complex software-intensive architectures. But it goes further. It also provides precise rules to relate what the customer wants of a system, how the system should function in support of the customer needs, and how to relate all of this to the current technology and products of the day, through a set of viewpoints.

Figure 3.2 provides an overview of the concepts of RM-ODP and how they relate. These are further discussed below.

An architecture specification is precise. It clearly defines all aspects (of interest) of the distributed processing system. The specification generally results from abstract concepts, at various levels of detail, which are themselves precisely defined and constructed in accordance with well-defined rules.

RM-ODP provides terminology about distributed processing that has achieved international agreement, consistently defined throughout the development lifecycle. It also specifies the process of how to use the terms in an explicit, precise manner so that the resulting specifications will be precise, understandable, and consistent, not "intuitive."

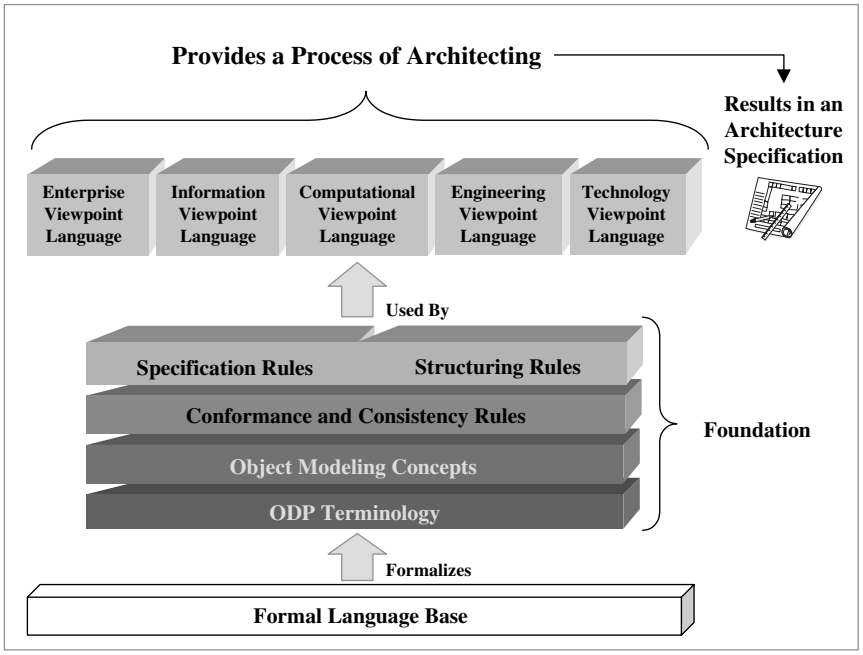

FIGURE 3.2 Overview of RM-ODP Concepts

A warm fuzzy feeling is not a specification. The precise specification that results can be refined into more and more detail, until a well-performed technical solution is realized as derived (refined) from the business specification.

The RM-ODP rules are categorized into basic rules, object modeling rules, structuring rules, specification rules, and conformance rules. The concepts and rules form the foundation of RM-ODP. The use of this foundation provides a process of architecting that comes along with precise terminology and rules that apply throughout the entire architecting process.

The *basic rules* are used throughout all the specifications. They address distributed processing, information, data, and what constitutes an open distributed processing (ODP) system. Further, these rules include abstraction, a very important concept in any architecting endeavor.

The *object model rules* are used to construct the architecture, in terms of objects, interfaces, state, and other elements. Each architecture specification is constructed in terms of an object-based model.

The *structuring rules* are used along with the object model and basic rules. They include how to address distribution, what constitutes a contract, what constitutes a policy, what is a group of objects, how to describe the behavior of a binding between interfaces, and so forth. These are the "hows" of a specification, based in object terminology and extended distributed processing terminology.

The *specification rules* are used to provide a consistent set of viewpoint specifications. The rules discuss how to compose, what a composition and component are, and how they are related. Further, these rules address how to specify behavior, interface signature, binding, and so forth.

Conformance rules discuss the conformance testing process and the points where conformance testing can occur. These apply throughout the viewpoint specifications as well.

In principle, one can create a specification of a system by taking a single view of the system. Often, though, the system is large and complex. The result can be a system description that is too complex to accomplish and too cluttered to understand. One of the key uses of RM-ODP is to address the architecture of a system from separate aspects of concern, called viewpoints. RM-ODP provides the ability to separate business application functionality from distributed system complexity, and distributed system complexity from choices of technology and products. This is accomplished by the use of the RM-ODP viewpoints, in conjunction with the rules discussed above. That is, *viewpoints* separate the areas of concern of the system into manageable parts to specify the architecture. The RM-ODP viewpoints fully capture an architecture specification of a system.

RM-ODP provides this separation of concerns of a system into five *viewpoints*: enterprise, information, computational, engineering, and technology. Each viewpoint captures certain concerns about *the entire system* from that viewpoint. Each of the viewpoints addresses certain aspects of distributed processing: how the user uses the system, how the policies of the organization affect the functioning of the system, how the designer selects current technology, how the tester ensures a correct implementation to the specification, and so forth.

The viewpoints are not layered. No viewpoint is more important than any other. They all provide a view of the same system, but each is focused on a different aspect of the system. These viewpoints are separate, but interrelated and consistent. Each viewpoint is defined in terms of a language, with defined concepts and structuring rules pertinent to that viewpoint language, all founded on an object model. An overview of the viewpoints and their purposes is shown in Figure 3.3.

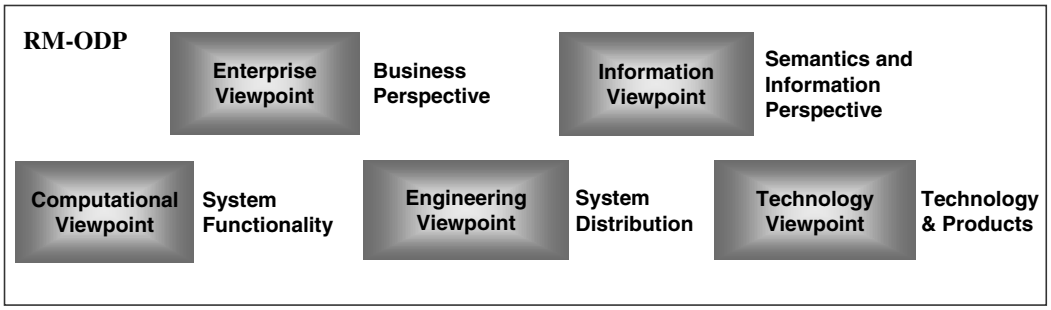

FIGURE 3.3 RM-ODP Viewpoints

RM-ODP facilitates communications with stakeholders and those involved in creating the system. The domain-specific needs are related to the elements of RM-ODP, such as a hospital (domain-specific) is a community (RM-ODP enterprise concept). The architect then uses the concepts of RM-ODP (community) and their relationships to other concepts to create an open specification that uses a well-defined distributed processing language to communicate exactly what is wanted. The problem of "what I meant was…" versus "what you understood was…" goes away. The resultant architecture specification is clear to all stakeholders. Further, because it has been so precisely defined, parts of the architecture specification can be reused as an architecture or design pattern. Parts of the working system can also be reused as plug-in components (or subsystems). All of these are doable because of the use of precisely defined "open" constructs, with well-defined terminology, and rules to explicitly define the system.

RM-ODP is one of the hallmark standards of the International Organization for Standardization (ISO) and the International Telecommunications Union (ITU). It is a foundation for any distributed processing system architecture.

Creating a standard such as RM-ODP is an international accomplishment. Developing shared terminology and ideas for what needs to be specified (such as partitioning the work of architecting into five viewpoints) is a theoretical idea that can be very powerful when used, but that use is not as simple as picking up an editing tool and "doing it." The emphasis here is on what needs to be accomplished in order to build an architecture specification. RM-ODP provides a systematic methodology for gathering knowledge from a variety of stakeholders and information technologists. This is accomplished using the viewpoints. For engineered systems, this can include users, operators, developers, and maintainers. For service industries, this can include not only operators, developers, maintainers, but domain-specific users such as patients, teachers, students, and doctors in the medical domain.

The RM-ODP concepts are not only precisely defined but also general. As such they are also abstract. They need to be understood in terms of the process of architecting and distributed processing. They do "work," but only if a talented software engineering effort (hopefully supported by good tools) is applied. The positive side is that the concepts are already defined for use, the architect does not need to redefine them first, and the concepts allow for higher precision in a specification—which is what architecting is about.

3.2 RM-ODP CONCEPTS AND TECHNIQUES FOR USE

RM-ODP is a precisely defined reference model that can be used for any distributed processing system. Its proper use can help ensure a well-formed, precise specification of an architecture, with consistent architectural concepts. Because of the rules and consistency, incremental additions to the system can be easily inserted into the architecture specification, generating technological solutions that are composable with the existing system. And this is generally a requirement by the customer of the architect: cost savings, reduced time to market, incremental evolution, and evolvability with emerging technology.

The concepts and techniques for specifying an architecture can be thought of in terms of three fundamental parts:

- Techniques of specification (that can apply to many things, not just distributed processing)
- The language of distributed processing
- The techniques of conformance testing

Architecture applies to many things, not just distributed processing systems. The specification techniques defined in terms of a set of rules are general and can actually apply to other things, such as how a heart works, or how a business is to be organized. However, in order to apply the specification techniques, a language of the area of interest is required. For example, the language of aorta valves, hypertension, arteries, veins, and so forth applies to the functioning of a heart; but the specification techniques can apply to whatever domain is of interest.

In RM-ODP, the language of distributed processing is defined. The viewpoints are defined in terms of this language, to enable focusing in on small parts of the topic (distributed processing) to create the specification. Because distributed processing pervades many business domains, relating the terms of distribution to those specific to the domain can aid the understanding of the architecture specification, though this is not required. For example, "Healthy Hospital Business Requirements" on page 58 deals with a "billing system," a "patient agent," and so forth. In RM-ODP, these are "objects that assume a role." Mapping the RM-ODP terms to the hospital terms aids the understanding of the architecture specification.

Keeping this in mind helps one to understand how all these foundational aspects of RM-ODP work together to formulate an architecture specification of a distributed processing system. Figure 3.4 provides a representation of how these aspects are related.

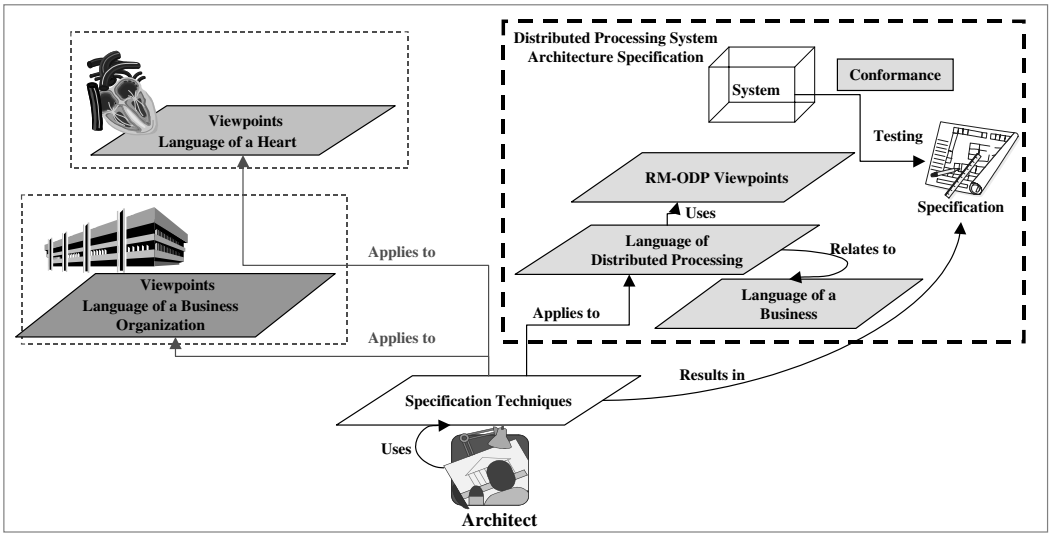

FIGURE 3.4 Relating RM-ODP Use of Concepts and Techniques

Each of the concepts and techniques defined by RM-ODP are covered in this section, at a high level. The remainder of this book covers these topics in more depth.

3.2.1 CONCEPTS

A common ontology (the set of defined concepts and rules of structure) is essential and powerful. It provides the power of clear communication. It provides the power of clear specification for what needs to be captured (such as interfaces). But using such precise terms and associated concepts for structuring are also difficult because of the generality involved, and therefore require skills in software engineering and abstract reasoning.

The terms are structured along the lines shown in Table 3.1 and are defined in the remainder of the book. Hence, this table provides all the terms without the definition, at this point, to show what each category covers and precisely defines.

TABLE 3.1 Concepts

CATEGORY	PURPOSE	CONCEPTS (TERMS)
Basic concepts	Used throughout	Distributed Processing, Open Distributed Processing, ODP System, Information, Data, and Viewpoint
Basic interpretation and linguistic concepts	Provide the concepts of interpretation of the models constructed from the RM-ODP language	Entity, Proposition, Abstraction, Atomicity, System, Architecture, Term, Sentence
Basic object model concepts	Used in building the architecture	Object, Environment, Action, Interface, Activity, Behavior, State, Communication, Location in space, Location in time, Interaction point
Specification concepts	Used across all the viewpoint languages to provide a consistent set of specifications from each viewpoint; Forms a part of the overall architecture specification	Composition, Decomposition, Composite object, Behavioral compatibility, Refinement, Trace, Type, Class, Subtype/supertype, Subclass/superclass, Template, Interface signature, Instantiation, Role, Creation, Introduction, Deletion, Instance, Template type, Template class, Derived class/base class, Invariant, Precondition, Postcondition
Structuring concepts	Used in conjunction with the basic concepts, basic interpretation concepts, object model and specification concepts to provide viewpoint-specific specifications; Used to address distribution	Group, Configuration, Domain, Subdomain, Epoch, Reference point, Conformance point, Transparencies, Contract, Quality of service, Environment contract, Obligation, Permission, Prohibition, Policy, Persistence, Isochronicity, Name, Identifier, Name space, Naming context, Naming action, Naming domain, Naming graph, Name resolution, Activity structure, Chain (of actions), Thread, Joining action, Dividing action, Forking action, Spawn action, Head action, Subactivity, Establishing behavior, Enabled behavior, Contractual context, Liaison, Terminating behavior, Causality, Binding behavior, Binding, Binding precondition, Unbinding behavior, Trading, Failure, Error, Fault, Stability, Application management, Communication management, Management information, Managed role, Managing role, Notification
Conformance	Used for testing of conformance	Conformance to ODP standards, Compliance, Consistency, Correspondence, Testing, Reference points, Programmatic reference point, Perceptual reference point, Interworking reference point, Interchange reference point, Change of configuration, Portability, Migratability, Conformance testing process, Result of testing, Implementer role, Tester role, Relation between reference points

In addition, the five viewpoints include such terms as:

- Enterprise—objects, policy, purpose, scope, action, community, process, step, and others
- Computational—objects, interaction, binding, signal, client, server, producer, consumer, interface, and others
- Engineering—objects, interface, binding, communication, transparencies, functions, channel, node, management, and others
- Information—objects, invariant schema, dynamic schema, static schema
- Technology—conformance points for testing

The terminology for use includes terms not only from RM-ODP but also additional terms from the General Relationship Model (GRM) [ISO GRM, ITU-GRM], which is another international standard. GRM provides a suite of concepts that enhance and interrelate to those of RM-ODP. An example is a further explanation and expansion of the term "invariant." In addition, the new Enterprise Viewpoint international standard [ISO-EntVP], not yet fully formalized, adds terms for specifying a business, to include such terms as community, process, step, and further expansion of other enterprise viewpoint terms.

3.2.2 SPECIFICATION RULES

The rules of specification include precision, abstraction, and composition. *Precision* means that something is well-defined, unambiguous, and consistent with other definitions. This includes the business rules, the properties of the business, the policies of the business, and the scope and objectives of the business. Precision is about how well something is defined—not how well something is detailed with attributes. At each level of detail, everything should be defined "precisely," whether the thing is the organization of a business, the details of an interface, or the specifications of a billing system used by the business.

Abstraction is a process of simplifiying. It is a key principle of specification. It provides a structure to a set of specifications that enhances understanding at each level. It allows key aspects of something to be addressed at one level, ignoring the details at a different level. Abstraction means that the customer does not need to worry about a bunch of things irrelevant to the specification of the business "language," such as what information technology terms to use.

"One of the key principles of RM-ODP is abstraction. People sometimes confuse the term abstraction with the common notion of being abstract or obtuse. In actuality, the process of abstraction is anything but obtuse—it is specifically designed to enhance, not frustrate, understanding. In RM-ODP, the purpose of abstraction is to provide a logical structure to a set of specifications by dealing first in the key aspects…and progressively pushing the rest to the lower levels of abstraction…" [Kilov-99] That is, the customer can define the business in terms

that enhance understanding. Enhanced understanding leads to a better architecture specification and system solution. The business is specified in terms understandable by the customer (such as "the hospital must submit a bill to the insurance company within 10 days"). The architect then takes the specification and further refines it to a system specification (such as a component that performs the calculations of a bill, a constraint on the interface of the component that it must meet the quality of service = 24 hours, or the interface binding to the insurance company to provide the bill).

For example, at one level of abstraction, a hospital admitting procedure is specified with respect to how it works with hospital billing and patient care. At a more detailed level, the interfaces to the database are defined, the interfaces to the patient records are defined, the manner of submitting the bill to the insurance company is defined, and so forth. Abstraction levels are progressively refined through more and more levels of detail, until the system is finally fully specified. Since the terms in RM-ODP are consistently defined, and since the rules of using the terms are consistent, abstraction coupled with further refinements results in consistent specifications.

All levels of abstraction need to be precisely defined. It is important to realize that the higher levels of abstraction need to be more precisely defined than the lower levels. The reason is that once imprecision sets in at a level of abstraction, it then generally affects all lower levels of abstraction. [Kilov-99] In the case of a business specification, it is exceedingly important to precisely define what the business rules are. Imprecision at this point will probably lead to a solution that is neither workable nor wanted by the customer.

A *composition* is a combination of entities that results in a new entity, at a different level of abstraction. That is, a composition is a grouping of entities that can be addressed as a single entity. One example of a composition is the collection of all employees in a department. Another example is the collection of all "database objects" in a "data store object." The "data store object" might be specified as an object at one level of abstraction, whereas each of the component "database objects" are specified at a lower level of abstraction. Composition is important in the business specification. A community can be a composition of other communities. They may be related in different ways, with cross-references between them. One community may be a refinement of another, or perhaps a community may be a specification for a community that is a refinement.

A relationship is "a collection of...objects together with an invariant referring to the properties of the...object." [ISO GRIM] An invariant defines what must be true during some timeframe of the relationship. For example, an invariant may state that the role constraints of a client/server relationship of the data store object are not violated. Hence the behavior of a relationship is defined by its invariants, preconditions, and postconditions.

3.2.3 OBJECT MODEL

RM-ODP defines a specific object model upon which everything else is based. This model is discussed in detail in "Essentials of the RM-ODP Object Model" on page 183. An *object* represents an entity. An *object state* is a condition of the object that determines the actions that object will next perform. This model is somewhat different from some other object models today; it is more capable. For example, in the object model for RM-ODP an object can have multiple interfaces. This model underlies the concepts "composition" and "component." An interface is part of the object model. An *interface* is an abstraction of the behavior of an object along with constraints. It includes a set of interactions.

3.2.4 VIEWPOINTS

Each of the five defined viewpoints is a form of abstraction, as discussed earlier. A viewpoint discusses the concerns of interest in specific concepts, along with structuring rules that apply to those concepts.

The following list provides the rules associated with the use of the RM-ODP viewpoints. These are discussed later in this section.

- Each viewpoint defines a set of concepts and allowable rules of structure
- Viewpoints are not layered
- Viewpoints address independent concerns
- Viewpoints are formal, founded on mathematical formal descriptions of predicate calculus
- Consistency across the viewpoints is defined
- Enterprise, information, and computational viewpoints are independent of distribution concerns
- Enterprise, information, computational, and engineering are independent of technology choices
- Conformance reference points are defined
- Viewpoints are based on the RM-ODP object model
- RM-ODP specification rules apply to the viewpoint
- RM-ODP structuring rules apply
- Viewpoints can be nested
- Viewpoints can use multiple levels of abstraction and refinement
- Viewpoints can use composition and decomposition
- Not all viewpoints are required for a given specification
- Combinations of viewpoints can provide different views of a system
- Additional viewpoints can be added

3.2.4.1 Synopsis

Enterprise Viewpoint. The enterprise viewpoint of RM-ODP takes the perspective of a business model. The enterprise models should be directly understandable by stakeholders (such as managers and end users) in the business environment. The enterprise viewpoint assures that the business needs are satisfied through the architecture and provides a specification that enables validation of these assertions with the end users.

This viewpoint is extremely useful to communicate the customer needs with the architect. It provides far more precise descriptions than, say, a Unified Modeling Language (UML) Use Case view. This viewpoint provides the customer the ability to define a policy such as, for example, "the state regulation policy for medical equipment has to be implemented in the system," or "our airline's policy is to overbook passenger seats to a maximum of 10 seats." These are "precisely" defined in the architecture. In UML, there is no concept of a "policy," though a policy can be represented as a class or a note or some other non-uniform manner.

Further, this viewpoint enables a customer to require a subsystem to be architected and implemented that will plug into an existing system. An example is "I have a legacy system that needs to work in my new system. Architect a means to do this." Or, "Architect something that will allow me to get to all the different databases in my system, and make it a part of my modernized system." Of course, it is the responsibility (and indeed the duty) of the developers to analyze and point out the probable impact of such proposed features on the rest of the architecture (e.g., greater complexity, cost).

This viewpoint can also be used for a subsystem part itself. Software tool vendors, for example, need to determine the scope and objectives of their tools. The parts that comprise a tool, coupled with their interactions (how that tool may interact with another tool), and the policies that apply are all part of an enterprise specification. This viewpoint sets the stage for further refinement of any software subsystem.

The terms used in this viewpoint include community, actor role, artefact role, purpose, scope, objectives, enterprise objects, interaction, process, task, policy, contract, environment, and environment contract.

Information Viewpoint. The information viewpoint defines the universe of discourse of the information system in two ways: the information content of the system, and the information about the processing of the system (its behavior).

The first is from the perspective a database model. The information viewpoint, in this case, is a logical representation of the data in the distributed system.

The second is from the perspective of the rules to be followed in the system, such as policies specified by the stakeholders, and then throughout the system in terms of how different components work together. For example, the information viewpoint provides constructs to define constraints of all aspects of the system

(such as behavior), constraints defined by policy, rules of allowable changes in state, things that must always be true (invariants), as well as quality attributes, and information.

Examples of constraints on the system could be the policy of overbooking airline seats, the interpretation of the policy on medical equipment, or the way two components work together in the system. The information viewpoint also captures the valid states of the objects. Finally, it captures the actions that are allowed to change the state of an object.

The information viewpoint is very useful in capturing the semantics of the operational system. It identifies schemata to do this. This viewpoint is an object-based logical model of the information assets in the business and the constraints on how these assets are processed and manipulated.

The terms used in this viewpoint include information objects, static schema, dynamic schema, and invariant schema.

Computational Viewpoint. The computational viewpoint partitions the system into functional modules that perform the capabilities of the system and are capable of being distributed throughout the enterprise. The computational viewpoint takes the perspective of a designer of application components and program interfaces. This viewpoint is similar to many architecture representation models, such as the logical model of UML, or the designer's view from Zachman [Malveau].

The computational viewpoint captures the components and interface details, without regard to distribution (which is addressed in the engineering viewpoint). Therefore, the architect need not be concerned if the application is on a particular server, or if the client must use a particular set of protocols to interact with the server. Rather, it is the viewpoint on the functioning aspects of the system, not on the distribution or implementation aspects.

In particular, this is where the software subsystem boundaries are specified in terms of application program interfaces (APIs), however those subsystems are distributed (as specified in the engineering viewpoint). Generally, these boundaries are the architectural controls that assure that the system structure will embody the qualities of interoperability, portability, scalability, and distribution in management of complexities that are appropriate to meet changing business needs, and adaptability to incorporate evolving technology.

The terms used in this viewpoint include computational objects, interface, interface signature, interaction, interaction signature, environment contract, policy, binding, operation type interface (client/server), stream interface, signal interface, client, server, producer, consumer, initiator, responder, binding object, trader, and others.

Engineering Viewpoint. The engineering viewpoint exposes the distributed nature of the system, and provides standard definitions that enable abstract descriptions of engineering constraints. These engineering objects are capable of defining the characteristics of all forms of distributed infrastructure, including

remote procedure calls, video teleconferencing, client/server communication, asynchronous interfaces for signaling, mobility of software and interfaces, multimedia, services providing fault tolerance, and so forth.

One of the engineering objects that RM-ODP defines is a binder, which forms the binding between interfaces. Another is a channel, composed of objects, one of which is the binder, and forms the full communication mechanism to tie together object interfaces. The engineering objects are capable of defining the characteristics of all forms of distributed infrastructure, including remote procedure calls, screening data interfaces, and asynchronous interfaces for signaling.

The perspective of the engineering viewpoint is similar to that of an operating system engineer or a networking engineer who is familiar with "thin clients" or "fat clients," Web servers, communication protocol stacks, and allocation issues that are necessary to define the distributed processing solutions for the distributed system.

The terms used in this viewpoint include engineering objects, interface reference, binding, channel, node, cluster, capsule, stub, binder, protocol, interceptor, relocator, migrator, checkpoint/recovery, failure, storage, node management, replicator, interface, policy, schema, and transparency.

Technology Viewpoint. The technology viewpoint serves two very important missions: to describe where to apply the technologies and products of choice, and to allow the conformance testing of the system implementation against its architectural specification. The technology viewpoint defines the mappings between the architected objects and interfaces to specific standards, technologies, product selections, and required developed code. The architecture specification provides the selection criteria for choices in this viewpoint.

The viewpoint defines four types of reference points that can be used as conformance test points. The reference point is a place where there is a set of interfaces. The four reference points for use as conformance test points are programmatic reference point, perceptual reference point, interworking reference point, and interchange reference point. Each conformance point defines the information to be observed during test, and how that information traces back through the architecture specification.

The perspective of this viewpoint is similar to that of an implementer who is familiar with the Web browser, the Web server, the communication between the two, the use of Enterprise JavaBeans™, network protocol standards, and other products to configure the information system.

3.2.4.2 Viewpoint Rules

The terms are not defined here, nor are the structuring rules that apply. Viewpoints are addressed in more depth in Chapter 6, "Separation of Concerns: Using RM-ODP Viewpoints." For now, let us say that viewpoints focus on a set of issues or concerns about a system, while reserving other concerns for later consideration.

Viewpoints are not layered, address independent concerns, are consistent, and support conformance testing. Each viewpoint is concerned with certain aspects of the system. Viewpoints are *not* layered; no one viewpoint depends directly upon any other, nor is one viewpoint used before another. They are independent in that they address different aspects of distributed processing.

However, they are also coordinated and as such do not stand alone. There are correspondences among the concepts of the different viewpoints that lead to consistency of the viewpoint specifications. An interface in one viewpoint, for example, corresponds to an interface in another viewpoint, but each addresses different considerations.

The information viewpoint, for example, captures the behavior of the system. Behavior affects everything. So the information viewpoint is coordinated with the remaining viewpoints. But it stands alone in the sense that the behavior is defined in that viewpoint alone. How the behavior is implemented, or how the behavior affects an interaction in a computational viewpoint, is where the coordination comes into play. That coordination is specified as part of the consistency rules, and correspondences among the parts of each viewpoint.

Further, all five RM-ODP viewpoints are co-equal in the sense that they each provide a complete model of the distributed system that is object based and corresponds to the other viewpoints. The RM-ODP viewpoints provide separation of concerns that divides the business and logical functionality of the system from the distributed computing and commercial technology decisions of the architecture.

With these viewpoints, a well-formed architectural specification can be created, which can result in a well-formed working system that can be tested for conformance.

Viewpoint Foundations. There are five viewpoint foundations: RM-ODP object model, specification rules, structuring rules, distribution independence, and technology independence. Each of these RM-ODP viewpoints is object based, based on the RM-ODP object model. They provide a complete model of the system from a given perspective of distributed processing (e.g., business perspective, behavioral perspective, engineering perspective).

Each viewpoint language consists of a small number of well-defined terms, based on formalisms whose detailed understanding is unnecessary. The language has no mathematical notion; English text can be used in relation to the viewpoint concepts. However, the RM-ODP concepts have been formally described in a mathematical language, to ensure a well-formed foundation. This has been accomplished, and need not be understood or used by the customer or architect. What this means is that statements of consistency and conformance are assured; they are not merely intuitive. Just knowing there is precision and a mathematical foundation is conducive to doing the analysis, consistency mapping, and conformance testing offered by RM-ODP.

All the viewpoints make use of, and are affected by, the specification and structuring rules. All the viewpoints make use of not only viewpoint-specific ontologies, but also the distributed processing ontology. That is, they are not just

terms to use in a specification; they include terms to use in conjunction with all the RM-ODP specification rules. All the viewpoints require engineering skills. Stakeholders work with the architect in using the enterprise and information viewpoints to specify the business. The stakeholder defines business objectives and the architect relates the business terms to RM-ODP concepts. For example, a stakeholder in the hospital business states a business objective "to manage patient admission," and a policy statement that "all emergency patients must be admitted." The architect relates these to RM-ODP concepts and rules of expression: an admission *role*—to manage patient admission; a *policy* statement—the admission role is *obligated* to admit all emergency patients. More on this topic is covered in Chapter 12, "Enterprise Business Specification."

The concerns of the enterprise viewpoint are from a scope and objectives perspective. This concern can apply to a business (such as the Healthy Hospital business of Chapter 2), or to a software-component specification, such as a data broker. The information viewpoint captures the behavior of the system. The computational viewpoint captures the components and their interactions, as affected by the behavior specified in the information viewpoint. Notice that none of these concerns addresses how the software is hosted on computers, how a network is used, or if all the software is co-resident on the same computer. These concerns of distribution come into focus in the engineering viewpoint, where all such considerations are specified. But the engineering viewpoint is not concerned with a choice of technologies, such as which database vendor is used (e.g., Oracle®), which data broker is used (e.g., Enterworks™ Virtual DB® [VDB]), which Web browser is used, which type of network is used, and so forth. These choices come into play in the technology viewpoint. This is represented in Figure 3.5.

The enterprise, information, computational, and engineering viewpoints are independent of specific implementations. In other words, most of the architectural specification is independent of the specific product selections used to implement the system. This characteristic of RM-ODP provides three important capabilities:

1. The independence of the business and architecture specification from technologies allows evolving technologies to be incorporated into the system without impacting the overall architectural constraints.
2. The architecture specification provides the criteria by which the product can be selected.
3. The architecture specification can be created to be visionary, in which case as products evolve and are included in the system, what has yet to be accomplished is known and defined in the architecture specification. This can yield selection criteria for products that include consideration for the future. An example of this very important point is a product that now provides visualization of stock inventory, or a product that has emerged to provide fault tolerance. The architecture specification provides the criteria of how these products must behave, how their interfaces must be used, and other criteria for how and where to compose them into the system.

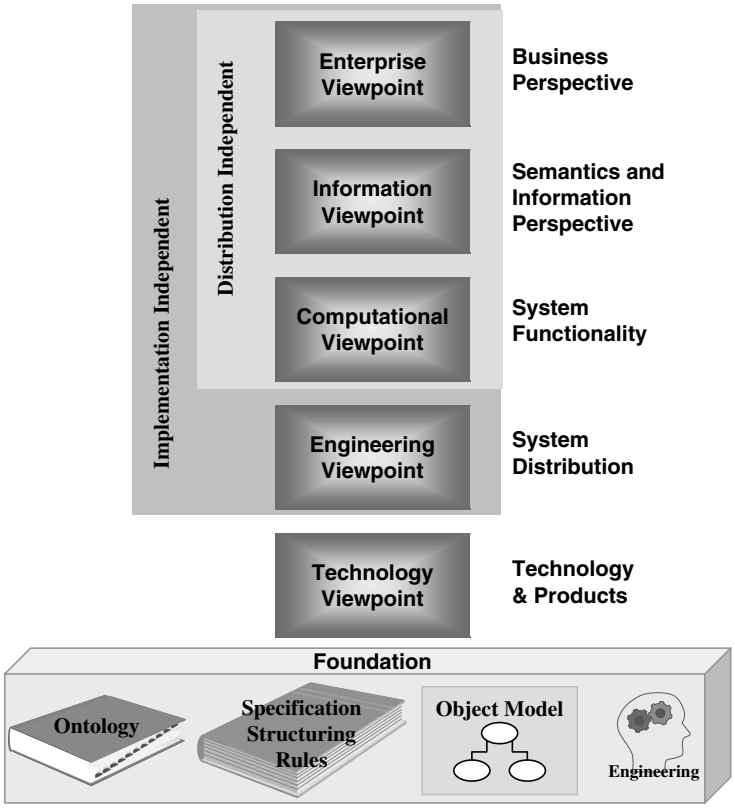

FIGURE 3.5 Viewpoint Foundation

Viewpoints can be nested, can use multiple levels of abstraction, and use composition. In the process of using the viewpoints, the architect does not need to fully use all concepts of a particular viewpoint before using another viewpoint. In fact, this would not work. For example, the use of a low-bandwidth network (determined in the engineering viewpoint) may require an additional object to manage binding of interfaces (a computational viewpoint consideration). This, in turn, may affect a quality of service statement in a policy (determined in the enterprise viewpoint). What the architect does is begin, capture as much as is known, soften constraints until more is known, and branch off to another viewpoint to perhaps learn more about a particular aspect. The use of the viewpoints is, in this sense, nested or iterative.

At any point in the use of the viewpoint, the architect can use any of the viewpoints *within* that viewpoint, to better specify something. Or the architect can use a subset of the viewpoints only sufficient to specify some part of the system. Hence, the RM-ODP viewpoints are full viewpoints on a system, that can be used

recursively within or partially, according to the needs of the architect. Through the consistency rules provided by RM-ODP, it all works together; it enables composition of architecture parts of the system, partial specification of a system, or a full specification of a system.

For example, in the process of architecting a portion of the system, it may become important to digress and define that portion of the system more fully. A synopsis of one way the viewpoints could be nested, for use in specifying a business, is shown in Table 3.2, all for a business specification that requires the use of a common message transaction service (MTS).

TABLE 3.2 Nested Viewpoints Example

VIEWPOINT	BUSINESS SPECIFICATION
Enterprise	A required message transaction service to be used, as an enterprise object, with interactions to other enterprise objects of the business
Information	Processing behavior that relates to the constraints about the MTS
Computational	Functional system configuration of objects and interactions that provide the business function using the MTS. More detail about the message transaction service to be used
Engineering	Distribution designs and distribution transparency mechanisms for the MTS and its use
Technology	Technology (e.g., OMG's Messaging service coupled with the Object Transaction Service), product (e.g., BEA's Object Transaction Server™ product)

An example of nested viewpoints may be seen in using the OMG Trader specification. In the architecting of a system, the architect may discover that a Trader is an appropriate component for use in the system. The architect does not need to fully specify the Trader component since OMG and ISO [ISO-Trading] have accomplished this. The architect need only identify its use within the context of other components, and reference the Trading specification. If there are certain behaviors that need to be addressed by the use of a particular vendor's Trader product, however, the architect needs to determine this technology viewpoint choice, and abstract the behaviors into the remaining system specifications. Again, this is both a nesting and iteration in the use of the viewpoints.

As another example, the Healthy Hospital example identified a hospital policy that affects the actions of the enterprise objects acting in their various roles (e.g., admission agent role, billing agent role). The architect wants to further refine the effect of the hospital policy on these actions. So the architect uses the computational viewpoint to better define the components and interactions, introducing a

new policy administrator object. The policy administrator object controls the appropriate use of the data associated with each of the agents. How? Perhaps the architect chooses to use a data broker that provides a single point of entry to all the data sources, and associate the policy administrator with that component. Perhaps the architect then wants to determine the distribution of the data broker and the data sources, and even the technology choice, to better understand the behavior. So the architect has drilled down on a particular aspect of the enterprise specification to better understand the interworking and semantics of the system components, and will then incorporate such knowledge into the information and enterprise viewpoints. This example is shown in Figure 3.6.

As was discussed in Chapter 2, "RM-ODP Manager's Primer," the viewpoints provide an approach for specifying the entire system, a part of the system in an incremental approach, or a visionary target system. In the latter case, the enterprise and information viewpoints will likely be used, and possibly the computational viewpoint as well. Engineering and technology viewpoints provide a

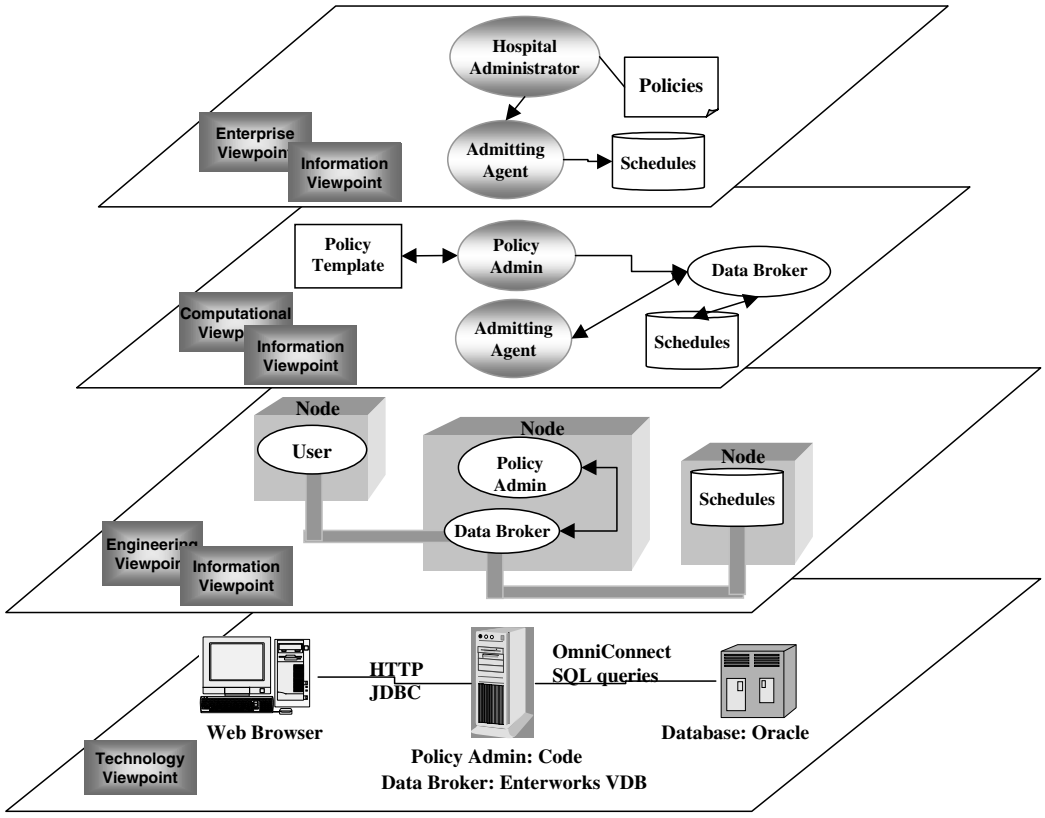

FIGURE 3.6 Example Nesting and Iterating Viewpoint Use for Healthy Hospital

virtual machine, which must be implemented by available technologies. If these are not yet in hand, the architect can still specify the visionary system, and then that portion of the system that is implementable, using the engineering and technology viewpoints. As technology emerges, the architecture is in hand to specify where that technology can be inserted with minimal impact. If the new technology is completely unanticipated, the full architecture specification may need to be revisited to ensure the technology meets the business needs, and does not adversely affect the current architecture specification and system. Cost avoidance, risk management, and schedule impact are all considerations at this point.

Not all viewpoints are required for a given specification. The architect does not need to use all the viewpoints, or even all aspects of a viewpoint. The choice is up to the architect to decide what it will take to provide a specification.

A prime example of this is the need to specify a business, the system, and the placement of the system in the business. The details of the distribution of the system and the technologies to use are perhaps left for some contractor to decide. So the organization's architect uses the enterprise, information, and parts of the computational viewpoints to clearly define what is wanted in the system. This too is a specification of the needs for a system. It is a method actually used in the creation of a service specification, or of a standard that is under the RM-ODP initiative. It is a method that can be used to construct a "reference architecture," which is popular today.

Taking a look at a reference architecture and what it means may help elucidate the use of the enterprise, information, and computational viewpoints. A *reference architecture* can be considered a high-level system specification that defines its overall target structure (components and relationships among them) in a systematic, consistent manner. A reference architecture frames, or bounds, choices, ensuring concepts and rules of structure are incorporated into all intermediate solutions, spiraling towards a target architecture. The essential difference between a reference architecture and an architecture is that an architecture is a more fully specified instance of a reference architecture. A reference architecture defines the envisioned target architecture, without addressing engineering or technology viewpoint concerns. So a reference architecture specifies:

- Objectives and scope
- Information
- Processing
- Business constraints and policies over the system composition (and components) and interactions
- Qualities of service (QoS)
- Distributed system objectives, without details of distribution
- Business objectives of incorporating previous (legacy) solutions
- Concepts and rules for business systems

- Decomposition into cooperating components and compositions
- Component relationships
- Delaying:
  - Considerations of technologies and products
  - Considerations of distribution
  - Considerations of data

In essence, a reference architecture provides the structure and rules to be used to define an incremental system solution. It is considered incremental because the current technology choices may not, in fact, achieve all that is specified in the reference architecture, and may need to incrementally evolve to the "target" or visionary solution.

A representation of a reference architecture is shown in Figure 3.7. It shows that the RM-ODP enterprise, information, and computational viewpoints can be used to provide a specification of a reference architecture.

Combinations of viewpoints provide specific views of a system. Each viewpoint either enables a viewpoint specification to be created, or contributes to a specification of multiple viewpoints. That is, sometimes several viewpoints are used to provide a particular specification, such as a business specification that includes the enterprise and information viewpoints.

Table 3.3 provides an overview of the viewpoints and the focus of concern in that viewpoint. When used in accordance with the RM-ODP rules, it becomes a viewpoint specification of the system, singly or in conjunction with another viewpoint, as exemplified in the Specification column. Some combinations of viewpoint uses result in commonplace views, such as a business specification, or a data management specification. Other combinations are possible; it all depends on the objective of the architect.

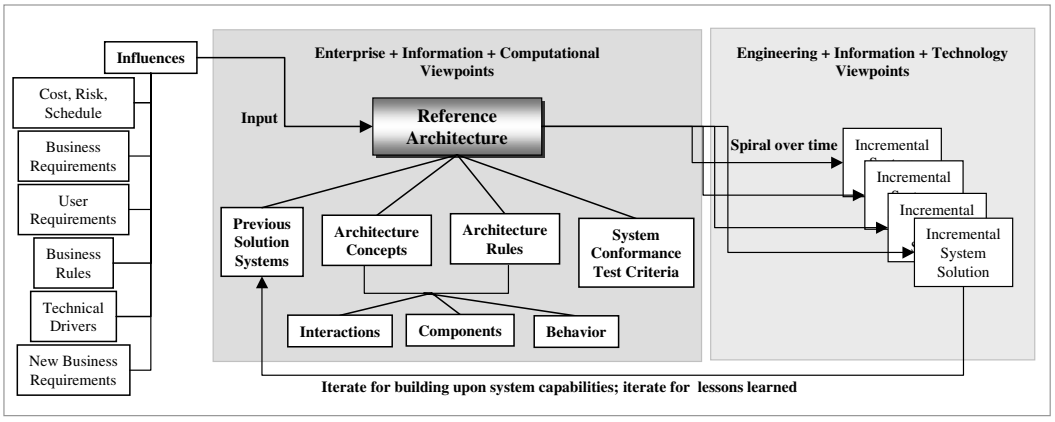

FIGURE 3.7 Reference Architecture and RM-ODP Viewpoint Use

TABLE 3.3 Viewpoint Concerns and Specifications

VIEWPOINT	CONCERN	SPECIFICATION
Enterprise	Purpose, scope, policies	Enterprise Specification
Information	Semantics of processing, and information	Information Specification
Computational	Functional decomposition into objects or components and their interactions, and their behavior and constraints	Computational Specification
Engineering	Distribution and distributed mechanisms, infrastructure, distribution transparencies	Engineering Specification
Technology	Technology, standards, products, and code choices; conformance test points	Technology Specification
	SOME COMBINATIONS	
Enterprise + Information	Business requirements along with behavior, and information	Business Specification
Computational + Information	Application "virtual machine": Application, application programming interfaces	Application Specification
Computational + Engineering + Information	Distributed processing "virtual machine": allocation of software to hardware, networks, etc.	Infrastructure and Distribution Specification
Enterprise + Information + Computational	Software component purpose, scope, policies, behavior, functional decomposition	Software Component or Standard Specification
Enterprise + Information + Computational + Engineering	Purpose, scope, policies, data, data flows, data stores allocated to computers	Data Management Specification
Engineering + Information + Technology	Implementation, Evolution to New Technologies, Incremental Implementation	Incremental Specification
All	Conformance testing	Conformance Specification

As was shown in the Healthy Hospital example of Chapter 2, "RM-ODP Manager's Primer," RM-ODP provides the means of specifying a system from a business perspective. That is, RM-ODP provides the mechanisms to capture how to realize the business requirements through a distributed processing system. This is not an easy task for any architectural framework. Using RM-ODP also provides the ability to capture business-related rules of engagement for the system. These concepts are further discussed in Chapter 12, "Enterprise Business Specification." As an example, the computational viewpoint discusses the concept of an interface. The rules associated with interface address different types of interfaces, how interactions are captured in an interface, some of the semantics of the interface, how a contract affects the interfacing (called a binding), and so forth. Perhaps this is all the architect needs to specify for some part of the system. Hence, the architect uses the computational and information viewpoints, and not all of the constructs in them.

Additional viewpoints can be added. Sometimes it's useful to view the entire system from a particular capability, such as security. Some of today's architecture frameworks (such as Zachman [Zachman] and UML [UML-1.3]) contain "views" of the system. However, the RM-ODP viewpoints are different in that they separate the concerns to stakeholder and architecture aspects, not just capability aspects. A capability is discussed in each of the RM-ODP viewpoints. For example, one common view of a system is the "data" view. This view may be depicted in a data flow diagram, representing the inputs and outputs of the data in the system. In RM-ODP, the five viewpoints define the need for the data, the elements of the data (the schema), the elements of the system that manipulate the data, the behavior of these elements (the data business rules), how the data is distributed, and how the data is managed by products or code. Another view of the system may be a security view. In this case, all aspects of an enterprise-wide security capability that exists in the system may be specified in a separate view. The RM-ODP set of viewpoints is not closed so that additional viewpoints can be added as the needs arise.

3.2.5 TRANSPARENCY

Finally, RM-ODP addresses what is called "distribution transparency." There are eight transparencies defined by RM-ODP. These eight transparencies are called access, location, migration, relocation, persistence, transaction, failure, and replication. The reason RM-ODP calls the set of transparencies "distributed" is that these transparencies hide the effects of distribution from the user, application developer, or system developer. The transparency constructs of RM-ODP are "orthogonal" in the sense that there is a separate set of defined constructs and concepts associated with each transparency, and the transparencies can be used somewhat independently.

However, once a transparency is "required," the constructs and concepts are intertwined with the mechanisms of the engineering viewpoint. That is, a transparency imposes constraints on the way the system works, focused on the engineering viewpoint (the infrastructure). These constraints are defined by RM-ODP. For example, if "failure" transparency is required, then the infrastructure needs to control what happens in the infrastructure, detect a failure, and set into motion activities that try to recover from the failure. All of this is done without knowledge of or impact to the application or the rest of the system.

The eight RM-ODP distribution transparencies are described in Table 3.4, from [RM-ODP-3].

TABLE 3.4 Distribution Transparencies

TRANSPARENCY	PURPOSE
Access	Hides the details of accessing another object, within or across heterogeneous systems. This includes hiding how the object is invoked, any data formats between the objects, and the interfaces required.
Failure	Hides the details that something has failed in the system, and the act of recovery. This transparency enables fault tolerance.
Location	Hides the details of a name and physical address used to locate some information and to interface to it.
Migration	Hides the details that an object has moved to some other location. This transparency enables other software objects in the system to continue to interact with the moved object, as though it were still in the same place.
Relocation	Hides the details that an interface has changed location, even if that interface is being used. An example is moving a server to some other location, and hiding that fact from the client. This can be used to support load balancing.
Replication	Hides the existence of copies of the software object in the system. Database replication is an example. This is often used for performance and reliability.
Persistence	Hides that an object continues to exist in the system, even if the object is deactivated and then reactivated in the system. This is used for stability and robustness of the object(s).
Transaction	Hides the coordinated activities of software objects in support of any transaction (e.g., database, messaging). A transaction is classified by the properties of atomicity, consistency, isolation, and durability (ACID). This enables a consistent transaction to occur, and establishes a more reliable system.

"Transparency" is about freeing an application program from having to provide some service or function. One has essentially "hidden" the details of where and how this service or function will be provided, and hence made the application "transparent" in regards to that function or service. The use of the term "transparent" is historical, but think of it as if having to look through an application to see where that function or service is occurring. This detail is hidden from a user. For example, locating a particular Web page is accomplished by merely accessing the local Web server, even if the Web page is geographically distant. An application is unaware of an infrastructure service that supports automatic recovery on a failure. An infrastructure service, such as something providing database services, is unaware of the management of a connection.

One typical use of transparency is to hide the complexity of some service provided by the infrastructure. In this way, for example, a complex service that supports a database replication function is hidden from any application or infrastructure service using that service.

A second use of transparency is to reduce effort of application development by relegating the complexity to the infrastructure developer, which can then be reused by all applications. This means that now the application developer does not need to develop his own version of some service or function, and can instead rely on some generalized version of that service or function in the infrastructure. This concept arises in such familiar situations as a single log-on feature that provides both the user and application the ability to log on the system once, and access anything that log-on privilege allows; mathematical functions that are usually called from libraries; or naming an object without having to know how the system refers to that object. What is different here, and perhaps unfamiliar, is how this same principle of reaching outside the application to libraries or repositories or infrastructure services can now be utilized for many other parts of the software application that may never have been considered.

Using access and location transparencies, the target location (e.g., the receiver of a message) does not need to be known. Instead, the application need only "refer to" the receiver in some way, say by some "name," and the infrastructure provides all the details to locate the target through its "name," ensure access to the receiver is valid, and establish the connection to the receiver. This frees the application developer from having to code target end details into his application (actually, into every application that would need to communicate with the target). It also reduces the number of interfaces to be developed, because the sending application can use the same interface for some data to multiple receivers, by putting in a parameter that indicates the receiver "name." The interface becomes reusable, and that reduces the cost of developing the system and maintaining it.

As another example, imagine that the user is logged on to a Windows® product. Now the user launches two applications, a vendor spreadsheet and a vendor database management system. Each one requires a log on. "Access transparency" allows the infrastructure for the user to log on each of these programs without

requiring the user to know the details of the log-on processes; they are filled in automatically. Thus access transparency has "hidden" the complexity of the log-on process and the connection process from the user, and from any other applications the user may have running as well.

A Web browser also provides access transparency. When the user clicks on a "link" on a Web page, the Web server transparently accesses the necessary servers by invoking the communication paths to the possibly heterogeneous servers, and then downloads the information from the distant server to the browser. These details are all hidden from the user by the Web server browser, and by the distributed name server that determines the location. Notice that the application developer does not need to be concerned with these details; the infrastructure services of the Web server are reused. An example of a request for a lung (versus a heart) image is shown in Figure 3.8. The "where" and "how" the lung image was obtained is a response to the user (client).

Load balancing is supported by the migration transparency. This is the ability, for example, to move a server to a new location or a different computer, without impact to the client software. Instead, the infrastructure records and manages the new location, much like a mail forwarding capability. The system keeps the forwarding address for a period of time. In addition, the system can also re-establish the connections of any interfaces being used, without knowledge of the application.

In order to achieve distribution transparency, RM-ODP offers important constructs in several of the viewpoints. The enterprise viewpoint allows the user to specify the transparency as a requirement. The information viewpoint supports this requirement in terms of specified constraints on the software and interfaces. The computational viewpoint specifies the constraint on the interfaces involved. This high-level specification of what talks to what, and the transparencies

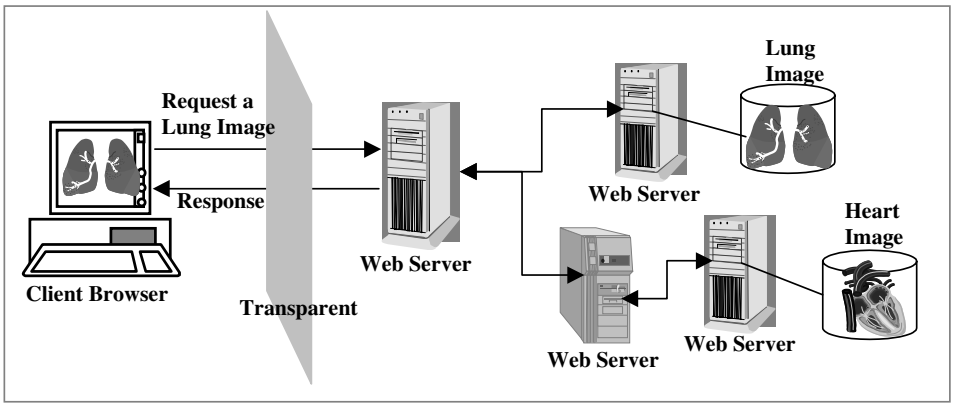

FIGURE 3.8 Web Location and Access Transparency Example

involved, allows one to identify potential services and functions that could be moved from the application level to the infrastructure level. The computational viewpoint supports distribution transparencies by abstracting the object interactions in terms of the constraints associated with the transparency. Lastly, the engineering viewpoint describes the mechanisms in the infrastructure to actually implement the transparency, and allows an application developer to see what mechanisms could be used to off-load some functions and services in the application to the infrastructure (e.g., locating a lung image). RM-ODP describes these mechanisms for any application and, as such, makes these mechanisms automatically reusable by each application. The key is to identify a transparency as a "requirement" from the customer in the enterprise viewpoint—then everything else cascades through the viewpoints.

3.2.6 CONSISTENCY

Viewpoint consistency ties together the constructs of one viewpoint to another. As an example, an interface as specified in the computational viewpoint corresponds to an interface as specified in the engineering viewpoint.

Consistency enables architecture evaluation, a common understanding, and increased completeness and consistency of the full system specification. Consistency across the viewpoints provides the ability to relate the specification of the architecture to a system implementing that specification, provides a consistent set of specifications that reflect what is wanted, and provides the ability to check conformance to the requirements.

There are several key points of correspondence that must be addressed. The computational viewpoint must support any dynamic behaviors that are specified in the information viewpoints. There is an explicit correspondence requirement between the computational and engineering viewpoints, and there is an explicit set of correspondences between the enterprise and computational, engineering, and information viewpoints. An example of a correspondence between an enterprise policy and enterprise role and constructs in the other viewpoints is shown in Figure 3.9.

There are typically more engineering objects than there are computational objects, because the engineering viewpoint exposes all the objects (which may be numerous) in the distributed infrastructure. For every computational interface defined in the computational viewpoint, there must be an explicit correspondence to an engineering interface in the engineering viewpoint. Each computational object corresponds to one or more engineering objects. In other words, the computational objects must map onto distributed engineering objects so that the distribution strategy is clarified by the architecture.

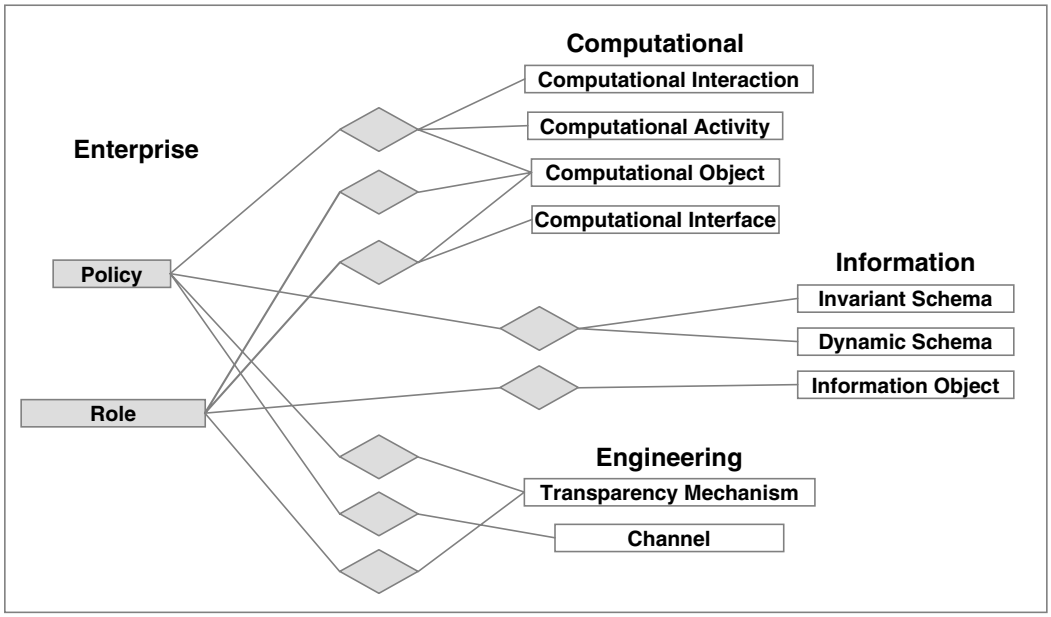

FIGURE 3.9 Sample Enterprise Viewpoint Correspondences

Another important point about consistency is that it enables geographically distant organizations to specify different parts of the architecture independently, and implement different parts of the system independently. This is achievable because of the rules of consistency across the specifications that will enable them to compose into a single specification or be realized into a single system. This is a capability provided by RM-ODP that addresses many programmatic needs in the architecture of systems.

RM-ODP defines the consistency of certain constructs between the various viewpoints. However, there are several areas not yet defined. Some of these are in the enterprise viewpoint and how the constructs of that viewpoint correspond to other viewpoints. The work in [ISO-EntVP] is furthering these consistency definitions.

Consistency checking coupled with conformance testing will increase confidence that the system will operate correctly and reliably. These activities are amenable to syntax and semantics checkers, architecture tradeoff analysis, simulation of the system based on the architecture, and other analytical capabilities.

3.2.7 CONFORMANCE

One of the most important features of RM-ODP is its concepts and rules supporting conformance assessment. Conformance assessment ensures that the implementation of the system corresponds to the architectural specification.

The RM-ODP rules of consistency across the specifications support four types of conformance reference points for testing, available for use by the tester of the system: perceptual, interchange, programmatic, and interworking. RM-ODP then proceeds to specify how conformance is achieved and represented in the architectural specification. The four types are represented in Figure 3.10.

One type of conformance test point is called *programmatic conformance reference point*. This reference point is used in the usual notion of testing the behavior of software interfaces. Many of the programmatic conformance tests address the architecture specification from the computational viewpoint specification.

A second type is *perceptual conformance reference point*. This is used in the testing at user interfaces in communications ports that represent external boundaries to the system. Usability and user interface testing, as expected by the business user, can be defined through perceptual conformance assessment.

A third type is *interworking conformance reference point*. This is used in testing between systems implementations, to make sure they interoperate. It is not sufficient for individual systems to have programmatic conformance (that is, only interface agreements) in order to guarantee interoperability. Interworking conformance includes interoperability testing between working implementations, making sure certain qualities of service are there, making sure that all the connections through all the software components work, and so forth.

The fourth type is *interchange conformance reference point*. This involves testing of the exchange of external media, such as disks and tapes. Interchange conformance assures that information stored on external media can be interpreted and incorporated in other systems that conform to the same standards.

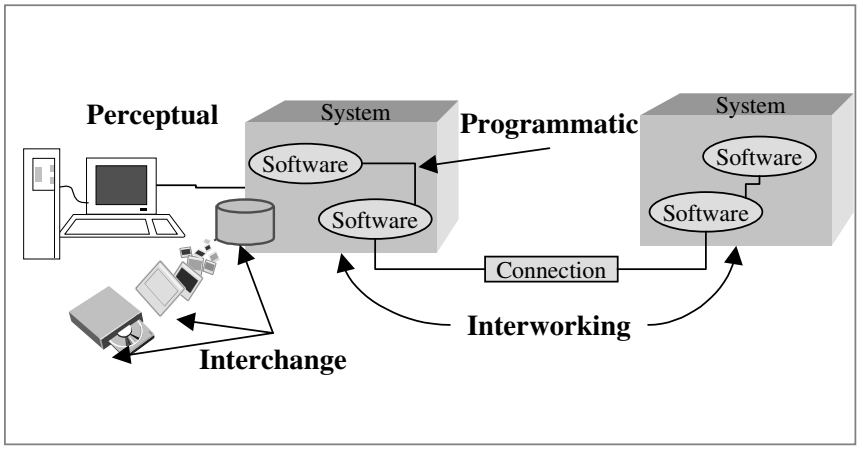

FIGURE 3.10 Conformance Reference Points for Testing

3.3 HOW RM-ODP IS USED

Thus far, the building blocks from RM-ODP have been discussed. Some use of RM-ODP has also been discussed, primarily with respect to the viewpoints. In this section, the approach to architecting, using RM-ODP, is discussed.

3.3.1 PREREQUISITE KNOWLEDGE

To specify a distributed processing system, certain knowledge is required. First, distributed processing issues need to be addressed. They occur in specifications time and again. Such issues include naming, timing, policies, administration, security, and remoteness. Understanding of the architecture specification principles of abstraction, refinement, composition, modeling, and so forth is necessary as well as knowing the difference between specification and description. This was covered in Chapter 1, "Open, Distributed Processing, Architecture, and Architecting," and more will be discussed in Chapter 7, "Architecture Specification and Representation." If this understanding is missing, training is in order. Knowing is prerequisite.

There is no value in creating a specification model per se—only in what that model allows one to do or subsequently understand. A specification model is not a physical, real, system (yet). It defines what the system is to do, precisely. Designing and building a software architecture is largely a modeling activity. The downside is that a model is abstract and takes effort to create. The upside is the ability to compare and analyze the appropriateness of an individual architecture from choices of architecture and the ability to tailor and specialize an existing architecture rapidly to new environmental circumstances, new technologies, or new functional requirements. A model founded on RM-ODP is also founded in mathematics. This enables analysis of the model, and assertions about consistency to hold true.

A tool is something that aids the architect. All of RM-ODP may be considered a tool for architecting. The architect must still use engineering skills to use the tool appropriately. In addition, graphical tools, commonplace in the market today (e.g., UML tools), are part of the architect's toolkit. Graphical development tools, also commonplace in the market today, are part of the architect's toolkit. Architecture analysis, product tradeoff analysis, testing, and even the use of a natural language are all tools for the architect's toolbox. The problem the architect has is to use the suite of tools in an integrated fashion, to generate a cohesive architecture specification, and to relay that specification to the stakeholders. There are no "silver bullets" that can provide an integrated toolkit for the architect. Hence, the architect not only needs to utilize engineering skills to architect, but must also use engineering skills to combine the use of all tools available to bring together a well-formed architectural specification.

Certain technologies today come into play in the solution. Knowledge of these technologies, what they do, and how they can be integrated in the system, are things that the architect needs to know. Examples include object modeling, abstraction, composition. Products and tools include IDL, UML, CORBA, DSSA[2], infrastructure middleware, Java™, Web, and so forth.

Tools for representing an architecture are always useful. In this book, UML models are used extensively. They provide not only a graphical representation, which is useful for communication with the stakeholders, but also some of the details about the architecture specification. They enable a better understanding of what is intended to be developed. But, as will be discussed in "Tool Support and Limitations" on page 129, UML has limitations that the architect must deal with.

Since RM-ODP and UML are generic, any domain can use them. To use them in a specific domain requires knowledge of the domain, knowledge of the appropriate language from the domain (e.g., *radiologist* in a medical domain), and someone who can relate the domain-specific language to the architecture specification language. The result of the architecture specification is focused on that specific domain and is called a DSSA (discussed in Chapter 1, "Open, Distributed Processing, Architecture, and Architecting").

3.3.2 CONTINUING WITH HEALTHY HOSPITAL EXAMPLE

Let's look at a continuation, as a refinement, of the Healthy Hospital medical example, for use in the remaining sections. The requirements for Healthy Hospital, in part, were identified in Chapter 2. The customer[3] wants the hospital to work smoothly. The customer wants the hospital admitting agent to manage patient administration. This includes admitting a patient for treatment by a radiologist, scheduling an appointment with a physician, and adhering to state regulations.

The customer is interested in a distributed system, but wants it explained in his own terminology. He wants to be sure that the kind of system he envisions will be realized. And he does not want to learn 60 years' worth of (computer) science and (computer) engineering, including 10 years of object-oriented technology, to determine whether the system will or will not work to his expectations.

An architect must also know something about the hospital business. Her understanding of this field may be abstract or minimal. She must also understand software architecture and design, to be able to communicate the architecture of the system to the implementers of the system.

The technology transfer of ideas and communication across functional and information technology domains has always been a critical issue, leading to lots of miscommunication, which often results in a system that was not what was envi-

2. IDL is Interface Definition Language; CORBA is Common Object Request Broker Architecture; and DSSA is Domain-Specific Software Architecture.

3. *Customer* here refers to the stakeholder who specifies the requirements of the hospital system.

sioned. This is largely due to a lack of a common language to explain what is wanted. RM-ODP provides the capability, through its well-defined constructs, to discuss hospital enterprise business functions and policies with the architect. This takes the form of describing the enterprise boundaries, the community of users, the system roles, the policies that constrain the system, and a high-level view of the main functional parts of the system and how they interact.

3.3.3 WHERE TO BEGIN

The objective, in the initial steps, is to capture as much as possible at a high level of abstraction. Details will come as further refinement is done. But at this early stage, some things may not yet be known, and therefore cannot be specified. So part of each step is to plan on revisiting what has already been specified. That is, the specification techniques are incremental and evolve. It is important to capture what is known, as soon as it is known, so that it is not forgotten.

Plan to refine the important parts (based on the highest risk, quickest time to market, lowest cost, or some other programmatic reason), capture behavior as known along the way, fill in the spots, and revisit. This is the essence of the initial step.

Begin the process by determining the scope of the system through the enterprise and information viewpoints. As part of the business specification, the scope, objectives, configuration of objects, roles assumed by those objects, the relation among the objects in terms of processes they perform, and activities they perform within each process are all part of the business specification.

The scope of Healthy Hospital is defined as what things are accomplished by humans, machines, or other real-world entities, as well as the system. The objectives of Healthy Hospital are specified. The objectives will define what the system is to accomplish, so they are important. Where the software system(s) is to operate within the business is defined. These are shown in Figure 3.11. There may be one software system or multiple software systems in Healthy Hospital (possibly different enterprise specifications combined into one). The software system is specified in terms of roles and objects and their relationships (interactions and processes), even with the nonsystem entities, and with the external systems.

Certain actions in a business may be assigned to certain users or software to perform, some of which are shown in Table 3.5. Each of these is a role, whether automated or a real-world entity related to a system in the enterprise. A role is a formal parameter providing an identifier for a functioning part of the community behavior. Initially, each role is very coarse-grained. As the refinement process continues, the role defined may result in additional roles in the specification. For example, the admitting agent of Healthy Hospital is a role. It may result in two roles: one that admits patients, and one that schedules appointments. The details of the refinement of the role will come later.

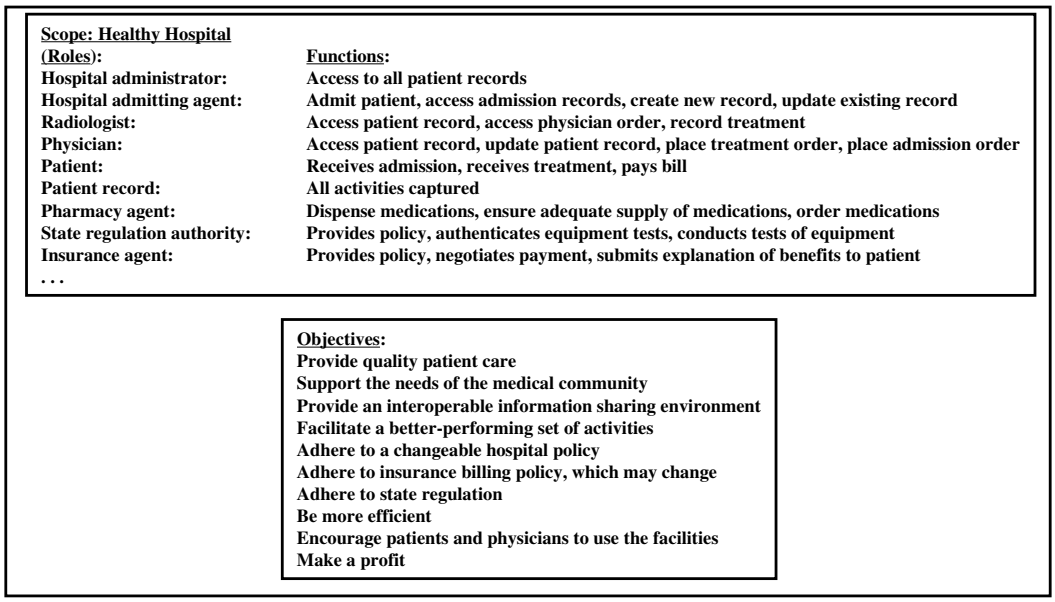

FIGURE 3.11 Scope and Objectives for Healthy Hospital

TABLE 3.5 Healthy Hospital Entities and Functions

ENTITY	KIND OF ROLE	FUNCTION
Administrator	Person	• Uses the software system
Administrator	Software system	• Defines the interfaces with the other external organizations • Ensures that the policies are adhered to • Manages the functions of the systems • Is able to access all records
Admitting	Person	• Uses software system
Admitting Agent	Software system	• Schedule appointments with hospital physicians • Schedule appointments with radiology treatments • Schedule admission to the hospital • Validate patient information • Validate that the patient can pay • Admit emergency patients • Update patient record information with schedules
Billing	Person	• Use billing system • Validate total bill

TABLE 3.5 Healthy Hospital Entities and Functions (Continued)

ENTITY	KIND OF ROLE	FUNCTION
Billing agent	Software system	• Compute a total cost of all bills • Ensure that the appropriate billing codes are associated with all billable treatments • Submit all bills to the insurance company within hospital policy timeframe • Submit bill of all balances due to the patient • Ensure the insurance company has submitted the explanation of benefits (EOB) to the hospital, along with payment, within hospital policy rules • Record all billing information in the patient record
Radiologist	Person	• Perform treatment • Perform equipment safety checks
Radiologist	Software system	• Validate patient information • Access the physician order • Validate equipment safety • Record the treatment to the patient record in accordance with hospital policy
Patient	Person	• Receive Healthy Hospital services
Patient agent	Software system	• Provide schedule of physician and radiologist treatments • Submit bill
Patient record	Software system resource	• Record patient name, address, phone number, place of employment • Record patient insurance information • Record patient outstanding balance • Record authorized physicians • Record physician order • Record radiologist treatment and results • Record all schedules • Record medications dispensed • Manage access control
Physician	Person	• Perform scheduled patient treatments • Access patient records • Achieve valid accreditation
Physician agent	Software system	[This is an external system, so the details are unknown. However, with respect to Healthy the following is provided by Healthy.] • Access patient records • Validate physician credentials • Enable updates to the patient records
Pharmacist	Person	• Manage inventory of medications • Dispense medications

TABLE 3.5 Healthy Hospital Entities and Functions (Continued)

ENTITY	KIND OF ROLE	FUNCTION
Pharmacy agent	Software system	• Provide an accurate inventory of all medications • Record all medications dispensed in the patient record • Submit an order for more medications as inventory decreases
Insurance company	External business	• Provide policies • Provide EOB to patient • Provide EOB and payment to Healthy Hospital
Insurance agent	Software system	• Provide EOB results to Healthy Hospital within amount of time specified in hospital policy • Provide the appropriate policies for use by Healthy Hospital • Ensure EOB and payment received by insurance company
State regulation	External business	• Provide state policy • Certify physicians • Certify Healthy Hospital • Certify equipment safety
State regulation authority	Software system	• Provide state regulation codes • Provide equipment safety certificates • Provide physician certifications
Notifier	Software system	• Interface with admissions for when a patient is admitted to the hospital • Notify all appropriate systems involved with admission (ill-defined in the example): nursing staff, physician, cafeteria, hospital room assignment staff, and so forth

The behavior is part of the information specification. The business and behavior information should be captured at a high level of abstraction. But initially, not everything is known. To the extent possible, the architect needs to capture:

- Cooperation—To what degree are the parts of the system to execute tasks jointly, and separately?
- Autonomy—What parts of the system are to be autonomous? That is, what parts are to retain their independence, though they may cooperate with other parts?
- Policies—What are the policies that drive the business and system needs? These should be expressed in terms of what is permitted, what is obligated, and what is prohibited.

- Quality of service (QoS)—In the sense of the enterprise viewpoint, a QoS requirement is an end-to-end user requirement. That is, a QoS may state that a bill must be submitted to the insurance company within 10 days. At this point, the QoS specification is not about real-time needs (necessarily, unless that is the subject of the enterprise specification).
- Shared environment—What are the objectives in terms of interoperability and sharing of information?
- Other "ilities," such as scalability, reliability, evolvability, flexibility, portability, dependability, availability, and security. A set of objectives about the different ilities should be specified at this point. "Ilities" drive the system specification. They are difficult, if not impossible, to add later. Hence, even if the stakeholder does not address these system properties up front, the architect needs to guide the stakeholder toward specifying what is important.
- Functionality—What the system is to accomplish, functionally, is specified. This specification will lead to the identification of enterprise roles, actions accomplished by those roles, and constraints on those actions.

The architect begins the process of refining the functionality into additional roles and relationships. An object is associated with a role, and as the role is further refined, different objects may assume different aspects of the role. That is, an object is identified to perform the functions of the role. For example, object "Guy" may assume the role of the admitting agent in Healthy Hospital. Another object "Linda" may assume that same role, at the same time or at a different time, depending on the constraints associated with that role. In each case, the functioning and behavior of the role is specified once, despite how many objects assume the role. Constraints are associated with a role. They apply to the behavior expected of any enterprise object that is to fulfil the role. For each role-based object, the interworking with another role-based object is captured, in terms of a set of actions or a single action. The architect specifies the set of actions between objects, not yet worrying about the specification of actual interfaces, what information is shared in the actions (which will be part of the information viewpoint specification), and what constraints are associated with the interworking.

The architect, then, captures the states of the objects, types of objects, and parameters about the object, to the extent known. The actions that allow the objects to transition from one state to another are captured in the dynamic schema of the information specification. The states the objects are allowed to transition to are captured in the static schema of the information viewpoint. The type and parameters of the object are captured in the information object specifications. This allows the architect to abstract the objects of the system, how they transition from state to state, and the actions that allow them to transition.

All along the way, the architect must diligently capture whatever behavior is identified. If the behavior is determined by a policy, the architect needs to also determine what the policy statement applies to, and the strength or weakness of

the policy statement. This is done in terms of what is permitted, prohibited, obligated, and associated with a role.

The architect must also begin the process of capturing environment contracts. Environment contracts define the interworkings with the external communities of the business. Contracts establish certain behavior and constraints on the business and need to be specified. These elements to be specified were addressed in the Healthy Hospital example, with the insurance company providing a policy that affects the way the hospital submits and processes bills. Some of the policies and environment contracts are shown in Figure 3.12.

The architect should fill in preconditions and postconditions as they are known, and revisit the behavior to update appropriately. (See [Wing-95] for more information about pre- and postconditions.) Preconditions in RM-ODP act as a guard to allow an object to transition, so it is important for the pre- and postconditions to be explicit at some point.

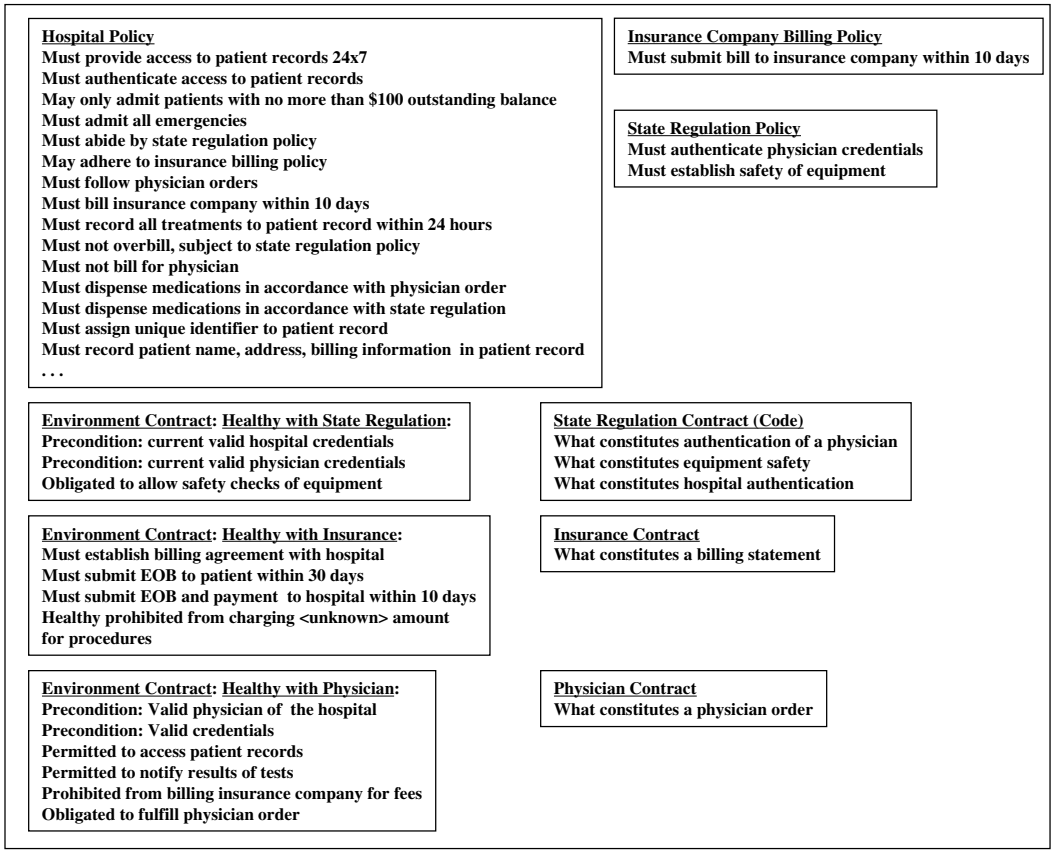

FIGURE 3.12 Initial Healthy Hospital Policies and Environment Contracts

In terms of the business, there may be patterns of business specifications for reuse. A business pattern is a structured set of concepts that reflects a specific business domain, what operations are involved, and how it can accomplish some objective. The patterns discovered for reuse help to fill out some of the specification for the business.

Some existing business patterns are founded on RM-ODP for reuse. Others are not, but can be adapted to the RM-ODP concepts. An example of the former is the general ledger specification from the Object Management Group (OMG) [GenLedger-98], wherein the enterprise, information, and computational viewpoints were used to specify the general ledger transactions, and the associated interfaces were then defined using IDL. An example of the latter is the specification of the Uniform Commercial Code (UCC) [UCC, and Kilov-98], which deals with the sale of goods costing at least $500. In this case, the specification results in a contract specification that is written down explicitly. An excellent discussion of such semantics for a business specification is [Kilov-98]. More information about enterprise business specification is provided in Chapter 11, "Enterprise Business Specification."

Specifying invariants as they become known is an important part of the process. An invariant is a property of one or more objects that remain constant (do not change) as the object goes from state to state. An invariant can be used to constrain the set of actions of an object, and in the case of RM-ODP, often leads to precondition statements. In particular, the invariants are specified in the invariant schema of the information specification, and constrain both the dynamic and static schemata. Therefore, as the architect determines the invariants, the architect needs to revisit the static and dynamic schemata to determine any possible changes. "Hard questioning of system invariants can lead to radically new designs." [Wing-95]

Another potentially critical factor is the handling of system failures. Since most systems fail on occasion, the architect needs to capture conditions of failure, and actions to take on failure. Some conditions of failure result in not adhering to a policy or violating an invariant. If certain critical actions must not result in an error, then a policy statement prohibiting such actions needs to be captured. If the system is to attempt a recovery, then fault tolerance is captured as a contract on the behavior of the system. For example, fault-tolerant systems generally have a rather complex infrastructure to support error detection, analysis, and recovery. (More about this topic is covered in Chapter 16, "RM-ODP Fault Tolerance Framework.") Sometimes error handling is addressed through an exception raised in the interface, and specified as such. This is a computational specification consideration (the specification of an interface and exception handling), which at this point, the architect may or may not choose to capture. But it's important to capture the objective that recovery from error is required in the enterprise viewpoint, to be refined later in the computational and engineering viewpoints, because the results of specifying error conditions generally lead to additional required capability in the system.

In terms of the objectives, certain critical factors may be invariant. These set the stage for how to proceed throughout the specifications, using the different viewpoints. One such factor that occurs often is the desire to use commercial prod-

ucts to the maximum extent possible. If this is required, then stating this up front as a part of the invariants is important. It affects decisions made throughout the entire architecture specification. The role the product plays in the business needs to be defined. The interworking of the product with other parts of the enterprise needs to be specified. Any known behavior, such as preconditions, postconditions, or dependencies, needs to be defined.

If the business requires the use of a product, the architect needs to position that product in the enterprise specification, and determine the effects (behavior) the product has on the business objects, and the effects the enterprise objects have on the product. Therefore, at some point, the architect may decide to find out more about the required product. The architect may then decide to elaborate the specification using the computational and possibly engineering viewpoints to determine the operations made available across the product's or object's interfaces and the behavior. This may require creating a component in the specification that represents the product, specifying the component as part of the environment of the system, and defining the constraints associated with interworking with the product. As these are exposed, the architect captures the specification in the appropriate viewpoint. The product, then, becomes part of the system, as well as part of the environment of parts of the system. The internal operation of a product is usually not made known. Capturing the specification of what is known in all the viewpoints, as applicable, better enables a system that can work together. This is represented in Figure 3.13.

In addition, through the process of architecting, criteria for any such product may be identified as constraints on the product selection. In other words, the constraints of a choice of product at this stage may later prove to be unsatisfactory to the solution. This may lead to a different choice of product or a refinement of the business rules because of the product selected. Whatever happens, the end result is captured in the architecture specification.

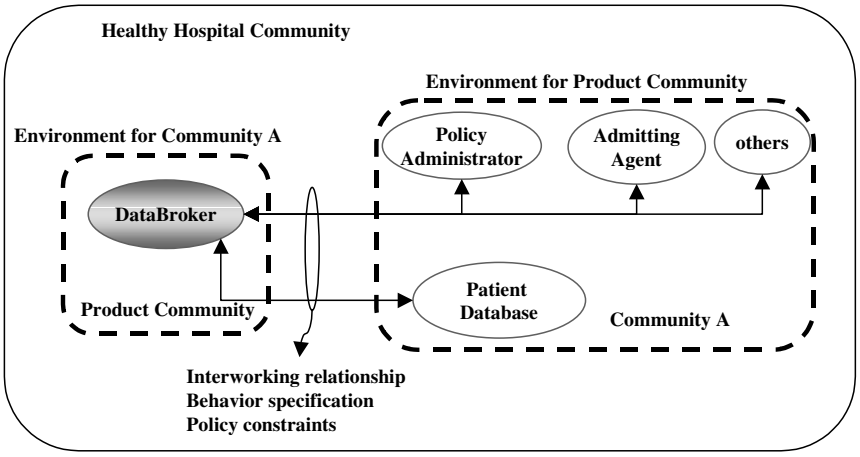

FIGURE 3.13 Specifying the Use of a Product in the Enterprise

Another critical factor that may emerge is the requirement that the solution must be delivered to market within, say, one year. This factor may limit the use of a formal specification to only those things deemed critical. These things are determined by the business specification. Typically, the things to be specified are database information specification, interapplication specifications, and the user interface, all in conjunction with the expected behavior. Functionality considerations that are "nice to have" may be delayed to an incremental release of the product. This will be easy to do with the RM-ODP specification, because where those additional capabilities fit into the solution will be clear, and can therefore expedite the delivery to market.

Another critical factor that may emerge is the need for a consistent user interface. A well-defined information specification and how that information is displayed to the user are derived from the information viewpoint specification. The definition of the information objects (the information displayed to the user), along with the allowable actions that change the state of the object (the actions to display different units of information), come from the information specification. The enterprise specification specifies the actions that take place across the objects to provide the information, and the actions the user can perform.

A point about using the enterprise viewpoint is in order here. The objects of the enterprise specification may include human users or machinery (such as in manufacturing), as discussed earlier. These objects can be modeled in the enterprise, along with their interworkings on the parts of the system, and the constraints applied to those interworkings. The physical entities are specified as a separate community, and the interworking with the system entities are specified in terms of an environment contract.

Another critical factor may be integration of the applications (components, in RM-ODP terminology) to better provide an information sharable environment. All sharable information should be defined in the information specification, as a single consistent model. The information model then drives applications and their interactions, specified in the computational and engineering viewpoints.

The architecture specification rules of composition, abstraction, refinement, precision are used throughout the enterprise specification, to get to a point of an adequately specified business. Once there, the architect can start refining the business specification to more detail towards a system solution.

All of the specifications created are founded on the RM-ODP object model and concepts. However, as was shown in the Healthy Hospital example, these concepts allow a domain-specific language: hospital, insurance, billing, physician, patient, and so forth. These are concepts from the medical domain, but are formulated in terms of the concepts from RM-ODP: objectives, environment contract, roles, policies, etc.

The information specification is the cornerstone for all the other specifications (from each of the viewpoints used). It is generally the most precise specification of the semantics of business and of the system to be developed. The behavior of the system is defined once, not threaded throughout all the remaining viewpoint specifications. The information viewpoint captures the terms of all the

other viewpoints into a single set of terms, to describe the behavior of the system. As such, throughout the architecting process the architect needs to pay close attention to the specifications in the information specification. Even though at the initial stage, preconditions, postconditions, and invariants are somewhat defined, the architect needs to constantly revisit this particular specification to ensure its correctness. As an example, date and time should be specified in the information viewpoint. Why is this important? Consider the Y2K (Year 2000) problem, for example. Had the date and time been correctly formulated in one place, it could be changed in one place, instead of the possibly thousands of places in the system.

A business specification will also define information and information processing across the enterprise, independent of actual interfaces or databases or any other implementation mechanism.

At this point, the results for Healthy Hospital might include what is represented in Figures 3.14 and 3.15. The use of UML has limitations, as noted in previous discussions, and as will be discussed in future chapters. But UML is adequate to use for some of the RM-ODP concepts, such as those shown in Figure 3.14. The limitations of UML revolve around the ability to adequately specify semantic behavior and constraints, as shown in Figure 3.15.

Figure 3.14 represents the roles and responsibilities in terms of a Use Case diagram. The "environment" notation is to represent external roles assumed. The

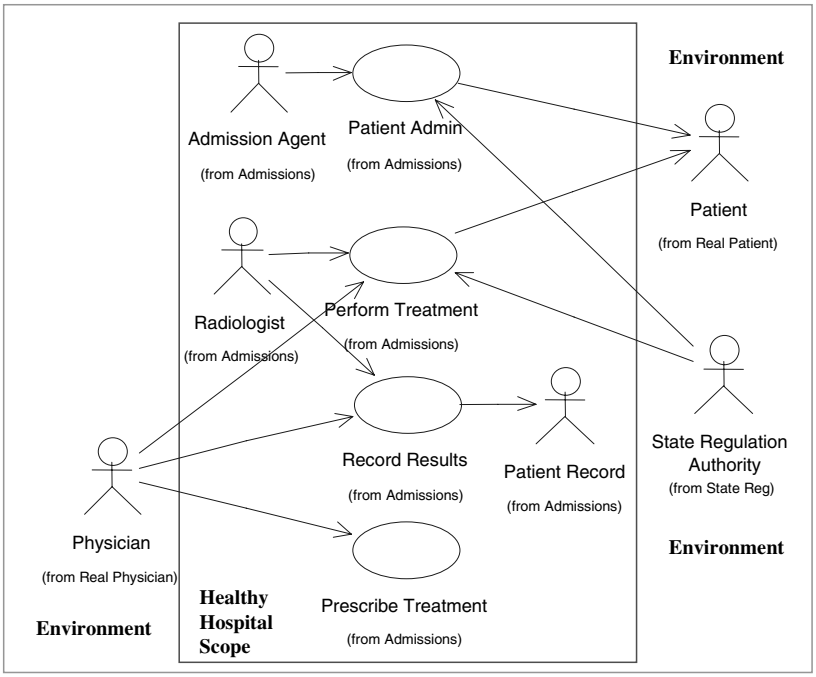

FIGURE 3.14 Use Case View of Healthy Hospital Business Scope and Objectives Example

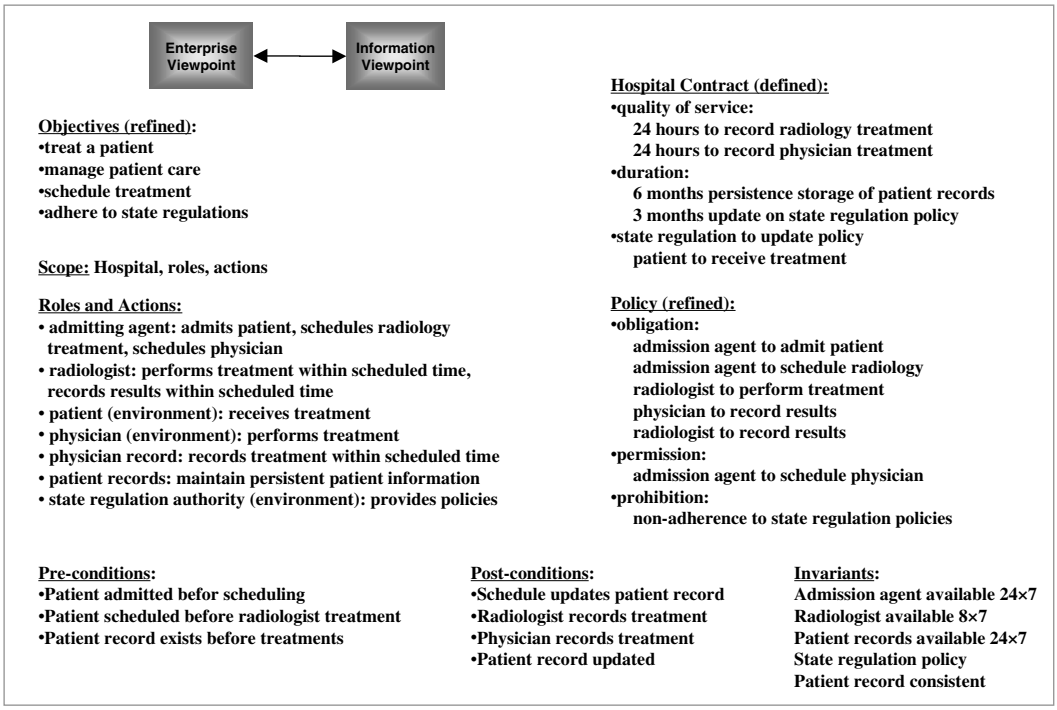

FIGURE 3.15 Some Refined and Defined Specifications of Healthy Hospital Example

scope of the system is shown as the internal set of actors and Use Cases. Figure 3.15 represents the details of the business scope, objectives, roles, policies, and behavior for the architecture specification, represented in English text.

At this point in the specification, the objectives and scope for the distributed processing system become clear. Issues related to distributed processing (discussed in Chapter 1, "Open, Distributed Processing, Architecture, and Architecting") are addressed in the next step.

3.3.4 NEXT STEP: REFINEMENT OF THE ARCHITECTURE SPECIFICATION

Once the business specification is partially defined, to include the behavior, the architect can refine it further using the computational and engineering specifications. The architect, who can relate these constructs to the RM-ODP viewpoint constructs, easily understands the language used to specify the business.

Decomposition into interacting components is captured in the computational viewpoint specification. Further refinement into infrastructure supporting objects, distribution, and communication is captured in the engineering viewpoint

specification. These two specifications provide the refinement of the business needs into a distributed processing system, constrained by the semantics captured in the information specification.

The next step is generally to use the computational viewpoint, to refine the business specification to a decomposition into objects and interfaces. Following this, the architect uses the engineering viewpoint to refine the specification into distributed software across different nodes (computers) and different networks (channels). Further, if distribution transparency is required (e.g., single log-on capability, virtual addressing, or replication), the mechanisms are defined in this viewpoint. Eventually the architect and implementer make product, standards, and technology decisions to realize the architecture and create a solution.

One might object that this resembles a waterfall approach to the use of the viewpoints, which in essence will not work! A waterfall approach would address each of the viewpoints in layers. First, the enterprise is fully defined. Next, the information, then the computational, then the engineering, then the technology, then the testing viewpoints are added. But architectures are not created this way. They are created incrementally, as more knowledge is learned about a subject. So the methodology of "specify a little, refine a little, learn a lot, and cycle through" is the approach that works. This approach was successfully used in work for Europe's Air Traffic Management infrastructure [Tyndale-Biscoe].

Again, what does work is using the viewpoints for parts of the specification *incrementally*, and folding back in the determining factors realized. This may require changes to the viewpoint specifications already accomplished. What's an example? Suppose the architecture to this point provides a single component in the system that interacts with the "physician" community, but there are hundreds of such communities associated with the hospital. Furthermore, their procedures differ, and a patient may be a patient of several physicians. Not only is there a single point of failure in the system (the physician component), but there is no concurrent access to the patient records allowed. A change to the business specification might be to allocate a separate physician component to each physician community, provide a control component that monitors the status of these components to ensure good working order and provides multiple accesses to another component in the hospital which, in turn, allows concurrent access to the patient records database. Maybe each of these components has a separate quality of service requirement: one physician wants "immediate" access on demand; one physician wants "results within 3 hours," etc. Each binding from the hospital component to the physician component would be different, based on these qualities of service. Again, this results in a change in the architecture, and these changes permeate throughout the rest of the specifications.

As can be seen, the viewpoints are not layered, nor must they be used in a waterfall approach. They are used, as needed, to provide more detail about a set of concerns in the architecture.

The architect may choose to employ a "spiral model" [Boehm-88] approach, as described in Chapter 2, "RM-ODP Manager's Primer." Perhaps there is a high-

risk area determined from the business specification, such as interoperating between the hospital and the insurance company that may have different policies or different architectural approaches. That particular interaction may be refined further throughout all of the viewpoints necessary to achieve a good, precise specification. Maybe the architect and implementer devise a solution to this high-risk area, prototype it, and iterate with the customer on how well it works. Using the conformance tests, the prototype can be validated to follow the architecture specification, so that if the customer doesn't like it, the specification changes.

An important step in this part of the specification process is to capture refined behavior in the information viewpoint. The architect should always look to the information viewpoint as the central important area of capturing the behavior of the entire system. It is not only a place to provide the behavior, but also a place to determine what behavior affects the decomposition of the system. For example, as was shown in the Healthy Hospital example, a QoS (billing must be accomplished within 10 days) affects how the interfaces are constructed, and how the binding of the interfaces works. This type of constraint may also result in additional components to make sure the binding behavior is followed, or produces a failure if it is not, followed by notifying a user of the failure.

Figure 3.16 further refines the medical example of Healthy Hospital, showing a possible distribution to client, server, and database platforms. A component named "State Regulation Procedures" is shown that reflects the state regulation

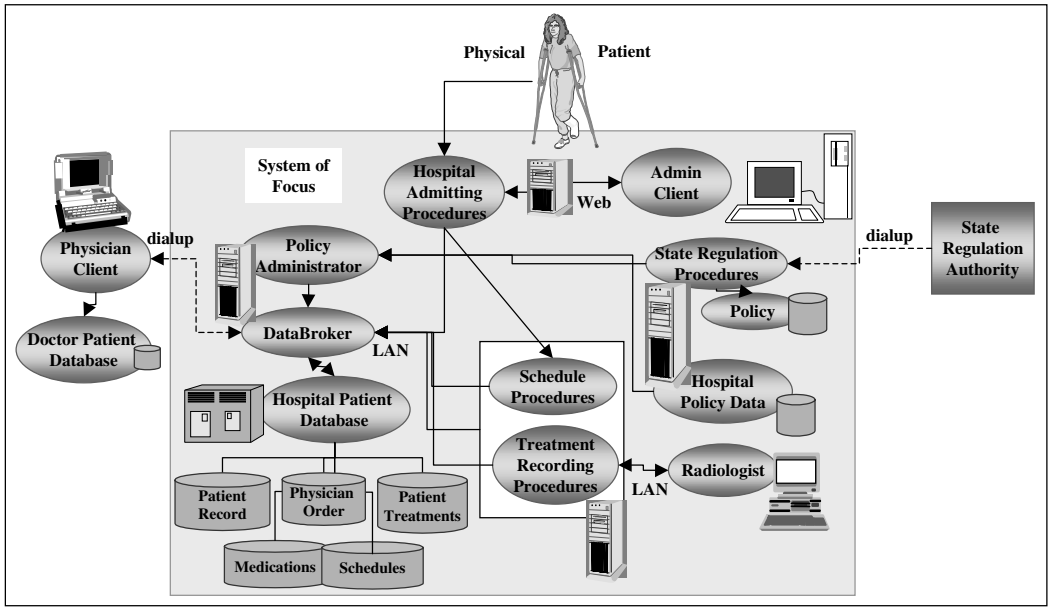

FIGURE 3.16 Example of an Architecture Using Computational, Engineering, and Technology Viewpoints

constraints identified in the previous two figures. The interfaces are shown as arrow lines, though in a true specification the details of the interface and bindings would be defined. The components "Schedule Procedures" and "Treatment Recording Procedures" are shown as residing on the same computer. Each of these components in a real architecture specification would be refined to much smaller components; e.g., the "Schedule Procedures" would define the objects and interfaces involved in its function. Also, not much is shown in the way of infrastructure service support (such as security, administration, management, and the like). The physician and state regulation authority are considered to be outside the scope of the system, but interact with the system as part of the system's environment. For each, a dialup connection is identified, and therefore must be provided. This is an example of how RM-ODP focuses on the specification of the system through the use of three separate RM-ODP models: computational, engineering, and technology. The effect of the information specification would show up as part of the true specification of each interface, binding, and even the object's behavior. This example is only notional.

Although technology today is capable of realizing many user needs through the use of the Internet, with very capable and resource-rich workstations, distributed component frameworks, and the like, the architect should still architect the distributed system. The large number of technologies, some that work alone, some that can work together, coupled with the need for designing the system to be cost effective, reliable, available, and well performing, remains a challenging task!

RM-ODP helps map the architecture onto the technologies and products of choice. The architecture specification captures the functioning and behavior of the system. With this knowledge, a technology is chosen that relates to what the function is to perform, how that function is to behave, what it must interact with, and how it must be distributed. Product selection then becomes easier, because what to look for and what to ask for become part of the criteria for selecting the product.

Part of this process also involves identifying where the specification cannot be met due to some insufficiency of the current technologies. When a new technology emerges, the architect is able to determine if it can perform the needed capabilities, within the constraints of the expected behavior of the system. That is, RM-ODP provides the architect with a set of criteria to evaluate product selection, and determine (through the specification of behavior) what properties the product must have to interact with other parts of the system.

In addition, RM-ODP provides the ability to establish conformance test points to test the system implementation to the specification. In the choice of a technology or product, then, the architect can identify where testing points need to be provided in the product and what information needs to be provided from the product. The tester can then observe the behavior of the system from these test points, and relate the information provided in terms of the architecture specification.

SIDEBAR

UNDERSTANDING SPECIFICATION AND TECHNOLOGY CHOICES

Technology today is both exceedingly capable and complex. Understanding what a given technology can provide requires the architect to understand its architecture, behavior, interfaces, and management. A common approach to the problem is to "buy commercial off-the-shelf (COTS)," to "glue COTS together," and then to convince the customer the end result is what was really wanted.

RM-ODP provides a common, consistent approach to specifying a system that is essential for a full and accurate specification. It is okay to realize the 80% solution, but have you ever wondered how you calculate the 80% when you do not know the 100%? Attempting to define the 100% is not easy. But attempting to find all the alternatives, understand them, and combine all the possible solutions from the technologies and products of today to achieve the system wanted is immensely difficult.

With limited budgets and limited time, the enterprise and information models define what 100% means. Many financial, telecommunications, and European government agencies are doing just that. Use of the rest of RM-ODP for architecting the business solution and design is a choice. The architect could define what 80% means, and proceed to architect the 80% solution. The architect could architect the 100% solution, and select technologies to solve 80%. Whatever is decided, all the business rules, processing semantics, and top-level functional needs of the system should still be specified. Then mapping the specification to the technologies and products becomes a planned decision, not an ad hoc one.

If the system requirements are informal at this stage, the architect needs to further specify them. This occurs by spiraling through the viewpoints and expressing the requirements in RM-ODP terminology. Evaluation of technical choices leads to possible refinement of the enterprise specification to allow for technical feasibility. For example, if a security technology product provides some capability that must interwork with the system administration capability, the enterprise specification needs to address this in terms of enterprise objects and their interactions, coupled with a policy statement that defines this need. Development of information and computational specifications may have security and management implications. This results in similar changes to the enterprise specification, information specification, and possibly the computational specification. The point of this discussion is that the viewpoint specifications are incrementally specified, refined, and even updated, as more information is gathered.

3.3.5 ITERATIVE REFINEMENT

The use of the viewpoints is to a certain level of abstraction. The architect has the tools of not only abstraction but also composition. So the architect may, at some point, have a precisely defined, but high level of detail about the architecture. What happens next?

The architect continues the process of refining the abstractions, refining the compositions into finer-grained objects that make up the composition, and further precisely defining their properties. The architect does this through the use of the viewpoints, always capturing more and more behavior details into the information viewpoint, and always using the information viewpoint to affect decisions on interfaces and bindings.

The architect in essence specifies a little, builds a little, tests a little, and determines if the result is what is needed, iteratively. The use of the viewpoints is represented in Figure 3.17.

When all of the coarse-grained objects (which are compositions of a group of objects) have been refined and defined, and all the abstractions have been defined in terms of implementation capabilities, the architecture specification is completed. Again, only those viewpoints necessary to get to the implementation

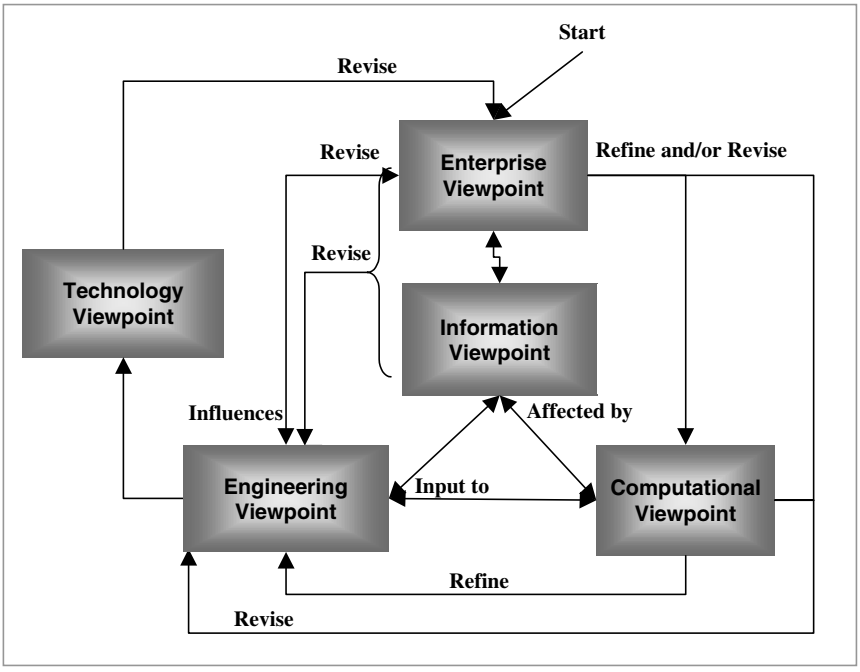

FIGURE 3.17 Use of the RM-ODP Viewpoint

point are necessary, and only concepts from each of the viewpoints that are necessary to precisely define the constructs of the specification are used.

For example, RM-ODP provides the concept of a node. This is in essence a computer and its operating system. The architect may already know, from previous iterations, that a Sun Solaris™ operating system and platform will be used. Therefore, the use of the resource management of this operating system, the threading capabilities, and how a channel is established, are specified by the operating system. The architect needs to translate the vendor-supplied information into RM-ODP terms, and either use the vendor concepts or the RM-ODP concepts. Whatever the architect chooses to do, a relationship between vendor and RM-ODP concepts is necessary to achieve the consistency and conformance checking of the viewpoints.

The process of writing specifications is similar to the processes of writing programs and writing mathematics. You need to worry about the big picture (e.g., the overall structure, organization, and meaning of concepts) as well as the fine details (e.g., syntax and special symbols). You need to learn the rules and concepts. You need to learn what rules must always be followed, and what rules you can break. As with writing programs and mathematics, writing specifications takes learning, practice, engineering, and patience. [Derived in part from Wing-95]

Even if the architecture depends on future technology becoming available, it may still be considered complete as long as the basic functionality of the system is isolated from the dependency of that technology. That is, if that technology is isolated to be part of the environment of the rest of the system, there are mechanisms in RM-ODP to address interaction with an environment. The future technology then has minimal impact on the rest of the specification. This was shown as an example in Figure 3.13. If the use of that future technology is intrinsic to the entire architecture, then the architect needs to re-specify the architecture to isolate the technology for use when it becomes available.

Once again, if the intended system solution does not need all of these, that's okay, as long as what is needed is precisely defined and specified through the architecture specification; a composition of viewpoint specifications.

3.3.6 WHEN WILL I BE DONE

The process of architecting is iterative. It iterates through the viewpoints (of interest), and then iterates back through them to make changes as more details emerge affecting the overall architecture. The rules of abstraction, composition, and object model foundation are always used in conjunction with each viewpoint.

If current technologies do not accomplish all that is wanted, then the architecting process continues until the emerging technologies and products are in hand to use.

When will you be done? When the system as wanted is in hand. But then the business stakeholder may want changes, added functionality, use of a particular new technology (e.g., voice capabilities, visualization techniques), requiring the architect to determine where and how to include these technologies. The short answer is the specification is completed when an implementation becomes feasible. The long answer is the specification is completed when the customer decides it's completed.

RM-ODP provides some patterns of reasoning for use by the architect in specifying more about the distributed processing system:

- Interactions are defined in great depth, and covered in Chapter 13, "Interaction Framework: Interoperability." The rules in RM-ODP define how constraints and behavior are associated with the interactions, how the interactions can be specified independent of distribution, and how the actual mechanisms for communication are specified.
- Support for composition is provided throughout the RM-ODP specification. In fact, just about anything in RM-ODP can exist as a composition, including behavior. It's interesting to note that a component in RM-ODP can be as small as a library or as large as a full system. What this means is that the RM-ODP rules and constructs of composition and architecture specification can apply to the full distributed processing system as well as to a fine-grained component that is part of the system.
- Heterogeneity, a problem in "plug and play" system composition, is handled by explicit specification of the behavior of the entity, associated with the RM-ODP specification of coupling heterogeneous parts through the interaction specifications. RM-ODP also addresses quality of service constraints, and how they drive the specification. More is discussed about this topic in Chapter 8, "Composition and Semantics: Quality Composition Minimizing Architectural Mismatch."
- A failure model and fault-tolerant set of mechanisms are provided. This topic is covered in Chapter 16, "RM-ODP Fault Tolerance Framework."
- Specifying QoS, and how it relates to the objects and interactions are specified in a companion RM-ODP standard [ISO-QOS, Sluman-97]. This topic is covered in Chapter 17, "Quality of Service Model."
- Federation provides the ability of multiple domains to remain autonomous, and yet share in the distributed processing. Federation is also addressed by RM-ODP, and covered in Chapter 15, "Federation."
- As was addressed in the initial step, specifying a policy determines a lot about the behavior of the system. How a policy is specified and how that specification affects the parts of the system is covered in Chapter 14, "Policy Framework."

- RM-ODP defines mechanisms for distribution transparencies, as discussed in this chapter. These are discussed in more detail in Chapter 10, "Hiding System Complexities: Distribution Transparencies."
- RM-ODP defines mechanisms for consistency and conformance testing. More is discussed in Chapter 11, "Architecture Analysis and System Conformance Evaluation."
- Certain infrastructure capabilities are supported by RM-ODP functions. An example is node management, which among other things manages the allocation of threads to objects, resources of the computer, and establishment of a connection. This is discussed further in Chapter 9, "RM-ODP Functions."

3.4 SUMMARY

The RM-ODP constructs and concepts are relevant and most important for any architecture specification. Viewpoints aid in this decision-making by separating concerns for the different stakeholders, omitting unnecessary details, and avoiding the expression of requirements in terms of solutions.

The RM-ODP patterns of reasoning provide the architect with many useful starting points.

RM-ODP is feature rich and will challenge industry to surpass existing capabilities for distributed systems architecture in the areas of behavior specification, delegation, conformance testing, and inter-domain administration: openness (portability and interworking), integration, flexibility, modularity, federation, manageability, QoS, security and transparency. Figure 3.18 provides a final overview of the parts of the viewpoint specifications, and a notional representation of the consistency across the viewpoints. The viewpoint mechanisms, coupled with a precise ontology for distributed processing, are powerful tools for architecting. The rules of specification and structuring are fundamental for any architecture endeavor. RM-ODP defines all of this.

Above all else, use of software engineering and software architecture techniques, as defined in RM-ODP, complement the domain area expertise in specifying an architecture. Rigorous software engineering is essential. Object technology and architecture technology are more than programming or a topology of computers and networks. It is essential to manage the levels of abstraction for an architecture specification of a system, defining the business rules, the interrelationships, and the semantics of the system's processing, and capturing this analysis through a model.

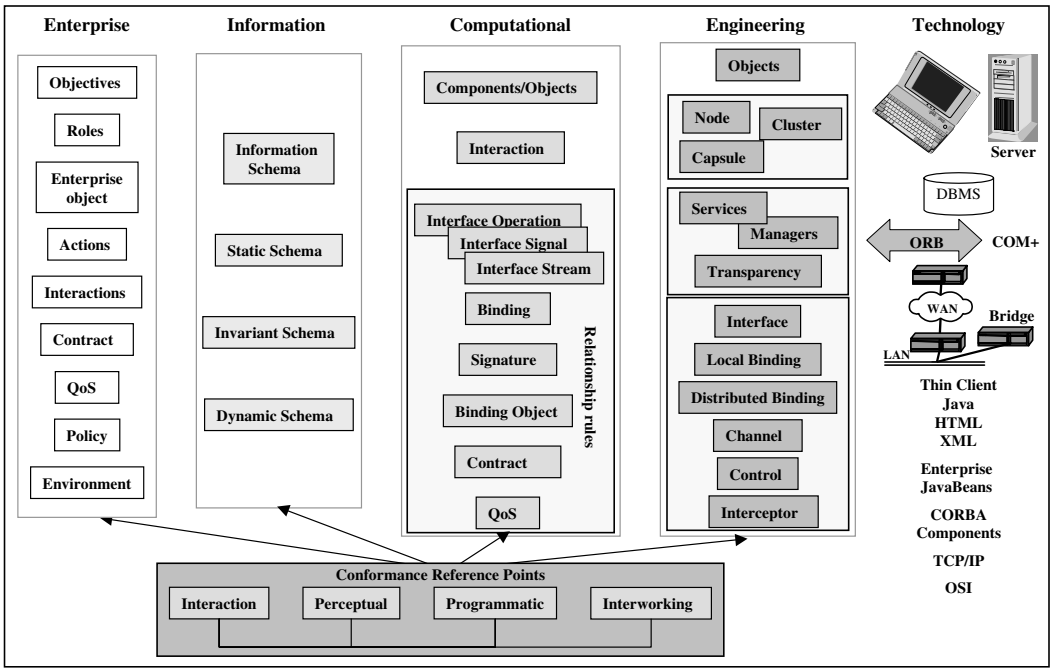

FIGURE 3.18 Overview of Viewpoint Concepts and Relationships

Some other benefits of RM-ODP are cited by [Holmes-94]:

- "A Common Language for expressing the behaviour, the problems and the requirements for heterogeneous organisations and systems, based on international standards
- A focus on externally observable behaviour rather than internal structure
- Support for 'what-if' analyses including effectiveness tradeoffs
- Support for evolutionary growth and technology interception"

The products of a standard such as RM-ODP are abstract. RM-ODP attempts to provide a general model for use in specifying any distributed processing system. It does work, but it takes a great deal of work to use the standard accurately.

Architecting is not easy. It is very easy to get it wrong, and very hard to get it right. Some automated tools exist that are claimed to help "architect." But use of a tool is only as good as the engineering and analysis that accompanies it. This chapter has attempted to elucidate the concepts of abstraction and composition as defined in RM-ODP, provide examples of their use, and guide the use of RM-ODP for specific needs in the remainder of the book.

One can be pragmatic about the use of RM-ODP. Not everything is always needed. One only needs to include those things that help plan the business objectives, the architecture, and the system implementation for the purpose at hand. One does need to use the terminology of RM-ODP, and the rules of specification and structure. However, one can elect to use a subset of the viewpoints, and a subset of the concepts and rules that make sense for the purpose at hand. However, use of RM-ODP should be correct. In this way, a cohesive system specification will result through the consistency constructs, from which analysis and conformance testing can be accomplished.

In this chapter some of the concepts and use of RM-ODP have been described. Subsequent chapters in this book provide more detail of these topics.