Enterprise Architecture
A to Z
Frameworks, Business Process Modeling,
SOA, and Infrastructure Technology

Daniel Minoli

CRC Press
Taylor & Francis Group
AN AUBERBACH BOOK
Contents

PART I: THE LOGICAL LEVEL: ENTERPRISE ARCHITECTURE A TO Z:
FRAMEWORKS, BUSINESS PROCESS MODELING, AND SOA

1 Introduction: Enterprise Architecture and Technology Trends .................. 3
  1.1 Introduction: The Criticality of Information Technology .................. 3
  1.2 IT Resources Requiring Proper Architectural Planning .................. 6
  1.3 What Is Enterprise Architecture ........................................ 9
  1.4 Trends in Enterprise Architecture and Data Center Environments ......... 19
  1.4.1 Enterprise Architecture ................................................. 19
  1.4.2 Technology Architecture .................................................. 21
  1.5 Course of Investigation ..................................................... 23
Appendix 1.1: A Formal Definition of Architecture .................................. 24
Definition .................................................................................. 24
Example .................................................................................... 25
Summary/Interpretation ............................................................... 28
Appendix 1.2: Bibliography on Enterprise Architecture ......................... 29

2 Enterprise Architecture Goals, Roles, and Mechanisms ..................... 33
  2.1 Enterprise Architecture ..................................................... 33
     2.1.1 Description of Enterprise ............................................. 34
     2.1.2 Definition of Architecture ............................................ 35
     2.1.3 Motivations for Having an Enterprise Architecture ................. 36
     2.1.4 Role Enterprise Architecture Group ................................ 41
     2.1.5 Organization-Specific Architecture Principles .................... 46
        2.1.5.1 Overview .............................................................. 46
        2.1.5.2 Additional Details ................................................ 50
     2.1.6 Instituting Enterprise Architecture Mechanism in a Firm ............. 52
  2.2 Enterprise Architecture Constructs ......................................... 53
     2.2.1 Enterprise Architecture Principles .................................. 54
     2.2.2 Enterprise Architecture Frameworks ................................ 54
        2.2.2.1 The Open Group Architecture Framework (TOGAF 8.1) ..... 55
        2.2.2.2 Zachman Framework for Enterprise Architecture ............. 57
        2.2.2.3 Extended Enterprise Architecture Framework (E2AF) ....... 59
        2.2.2.4 Department of Defense Architecture Framework (DoDAF) .. 59
        2.2.2.5 Enterprise Architecture Planning (EAP) ..................... 63
        2.2.2.6 Federal Enterprise Architecture (FEA) .......................... 64
        2.2.2.7 Federal Enterprise Architecture Framework (FEAF) .......... 67
        2.2.2.8 Treasury Enterprise Architecture Framework (TEAF) .......... 68
## 4 The Zachman Architectural Framework ................................................................. 111

4.1 Background ........................................................................................................... 111
4.2 Framework ........................................................................................................... 112
4.2.1 Principles ......................................................................................................... 112
4.2.2 Framework Structure ....................................................................................... 112
4.3 Architecture Implementation .............................................................................. 115

## 7 Business Process Modeling .................................................................................. 139

7.1 Business Process Modeling ................................................................................. 141
7.2 Business Process Modeling Standardization ...................................................... 143
7.2.1 Business Process Modeling Language .......................................................... 143
7.2.1.1 Activities .................................................................................................... 143
7.2.1.2 Activity Types ............................................................................................ 143
7.2.1.3 The Activity Context .................................................................................. 145
7.2.1.4 Simple and Complex Activities .................................................................. 146
7.2.1.5 Processes .................................................................................................... 147
7.2.2 Business Process Modeling Notation .............................................................. 147
7.2.2.1 Introduction .................................................................................................. 148
7.2.2.2 BPMN Overview ......................................................................................... 148
7.2.2.3 Business Process Diagrams ......................................................................... 153
7.2.2.4 Examples ..................................................................................................... 154
7.2.3 Business Process Query Language ................................................................. 169
7.3 Unified Modeling Language™ ............................................................................... 169
7.3.1 Overview .......................................................................................................... 170
7.3.2 Scratching the Surface of UML ....................................................................... 173
7.3.2.1 Conformance ............................................................................................... 173
7.3.2.2 Runtime Semantics of UML ....................................................................... 179
7.3.2.3 The UML Metamodel ................................................................................ 181
7.3.2.4 UML Infrastructure Specification ............................................................. 182
7.4 Model-Driven Architecture .................................................................................. 191
7.4.1 MDA Background ........................................................................................... 191
7.4.2 MDA Support .................................................................................................. 196
7.4.2.1 The Meta-Object Facility (MOF™) ............................................................. 196
7.4.2.2 UML Profiles .............................................................................................. 197
Chapter 1

Introduction: Enterprise Architecture and Technology Trends

Why is information technology (IT*) critical to organizations? What are the important trends in IT infrastructure technology? What are the technical and cost issues? How can a firm manage the infrastructure in an optimal manner that reduces costs and maximizes opportunity? Considering the answers to those questions, why are enterprise architecture (EA) methodologies important? Read on.

1.1 Introduction: The Criticality of Information Technology

Of late some have claimed that IT as applied to enterprise and institutional environments has become commoditized and it is no longer a strategic asset. Such statements originating from consultants, speakers, and magazine writers may well be off the mark. Although some commoditization has indeed occurred in the past ten years or so, in fact, it can be argued that not enough commoditization has yet taken place in IT, and more is needed. A lot of IT remains an art rather than a science: software solutions for a significant portion of a company’s functions tend to be one-of-a-kind developments rather than being off-the-shelf standardized solutions. Furthermore, business demands required to support a service/consumer economy rely even more deeply than ever on having detailed real-time market and marketing information related, but not limited, to customer trends, needs, wishes, ability to pay, history, order taking, order tracking, supply-chain, physical

* When we use the term information technology, we mean the entire corporate activity and function related to computing, networking, and storage. This includes the people, the assets, the systems, the software, the applications, the practice, and the principles related to this function.
product delivery, and distributed (if not virtual) inventories. Additionally, consumers want to be able to check product offerings, status, availability, account and billing information, to name a few, in real-time and literally from anywhere. The support of this kind of business landscape and customer experience requires a sophisticated modern IT capability that is based on architecture-driven principles. A firm that can put together such a tightly woven, information-comprehensive infrastructure is well positioned to become an industry leader. Stated differently, given the competitive environment, it is far-fetched to believe that a firm can be a leader in its space or prosper if it does not have a well-planned, well-architected, best-in-class IT apparatus.

For example, consider the case of extending credit to consumers. If a firm can successfully perform data mining that goes deep beyond the openly available credit scores to establish a multidimensional risk decisioning process in regard to extending a specific individual a credit line, whereas every other competitor relies only on the one-dimensional openly available credit score, then that firm can, perhaps, quickly become an industry leader and build a multi-billion-dollar operation. This data-intensive operation requires a well-planned, well-architected, best-in-class IT corporate capability.

With a commodity, one can open up a product catalog or an online product portal, quickly determine which product meets one’s needs, quickly undertake a feature customization, and speedily receive the product for actual use or deployment within days. The procurement of PC products is an example of a commoditized function at this time, as also the online purchase of a luxury car, boat, recreational vehicle, or modular home. In both cases one can look at a supplier’s portal, see which off-the-shelf item out of a well-defined set of items available on the “shelf” satisfies the buyer’s price/feature goal, apply some feature customization, and then immediately order the product; in the case of a PC, it typically arrives in a few days; in the case of a car, it could typically be delivered in a few weeks. That is not what happens when a large-size company needs, for example, to buy a billing system: such a project might require several hundred staff-years and cost several hundreds of millions of dollars, even when a core shell package might be available to build upon.

That IT does matter is established by the fact that there now exists a business environment that requires continual improvement and where the ultimate business goal is always a moving target that an organization must strive to achieve (but perhaps never truly attains.) First-generation (1G) IT evolved (in the 1970s) to meet 1G (batch-based) business processes. However, second-generation (2G) business processes emerged (in the 1980s and early 1990s). It is true that a firm would find it noncompetitive to try to address 2G business processes with 1G IT solutions, to the point that the firm could perhaps state that “those batch IT systems of the 1970s do not matter to the strategic needs of the 2G world”; but what a firm needed was not to declare that informatics did not matter, but to step up to develop 2G IT systems. Since the mid-1990s a third generation (3G) of business processes has emerged, based on online accessibility. Again, it is true that a firm would find it noncompetitive to try to address 3G business processes with 2G IT solutions, to the point that the firm could perhaps state that “those closed IT systems of the 1980s do not matter to the strategic needs of the 3G world”; but what a firm needs to do, then, is develop 3G IT systems. Now we are evolving to a 4G set of business processes, those based on a ubiquitous computing/location-based services/always-connected paradigm; 4G IT systems will be needed. And after that, past 2015, we will evolve to 5G business processes and 5G IT systems will be needed (what these are is anybody’s guess, but it will happen). One does not want to fall in the trap of those physicists in the 19th century who stated that all that was there to be discovered in science was already discovered by then, or of those industrialists in the 1950s that stated that a handful of computers is all that the world would ever need. Fortunately, IT does move on and great new domains remain to be
defined, appreciated, and conquered, to the financial benefit of the winners. At each stage IT is
the strategic tool to conquer the new set of business opportunities; the world is not static and the
“endgame” is not in sight.

To declare that “IT doesn’t matter” is to assume that the “world comes to a stop” and nothing
else happens hereafter in business, with the customer, with the environment, with the competi-
tion, and with regulations. So, although there were certain business expectations in the 1970s
from customers, and IT systems were developed to support these (perhaps reasonably well), the
underlying business processes have changed significantly since then. For example, in the 1970s
one could generally only do banking Monday to Friday, 9:00 AM to 3:00 PM; the transaction
had to be done face to face and in a bricks-and-mortar building. The customer’s expectation was
simply that if he or she walked into the bank, the bank would be able to reliably complete a trans-
action (e.g., a withdrawal) by having access to an up-to-date ledger. If the “world were to stop”
then and nothing else ever changed, and a bank had developed an optimized IT system to support
that paradigm, then in the early 1980s the bank could declare that there was no strategic value in
spending more money in “tweaking” the 1G system. However, automated teller machines (ATMs)
revolutionized the banking scene in the 1980s. Banks that could quickly and reliably bring this
service to customers (by updating their IT systems) were the ones that thrived. Banks needed to
make major investments in IT to serve the new customers’ expectations. If the “world were to
stop” in the late 1980s and nothing else ever changed, and a bank had developed an optimized IT
system to support that ATM paradigm, then in the early 1990s the bank could declare that there
was no strategic value in spending more money in “tweaking” the 2G system. However, the online
approach to doing business (shopping, banking, paying, etc.) burst onto the scene in the 1990s.
The same observations just made about necessary (new) IT investments apply. At this juncture,
or soon hereafter, there may evolve an expectation for location-based services, such that when a
consumer enters a space, say a mall, the consumer’s PDA, iPod, cell phone, etc., starts telling the
customer what services, sales, specials, etc., are available at that precise spot; if the customer walks
200 yards farther along, new information is then delivered to the customer. The customer may
wish to make an on-the-spot purchase with a wireless financial transaction; then the customer
may want the order shipped, again via a wireless interaction; then the customer may want to check
how much money he or she has in the account; then the customer may want to see if money can
be transferred on the spot for seven days to another bank that gives the customer a better interest
rate—or transfer credit card balances to another credit card company with better rates. If that is
the paradigm of the near future, then firms are well advised to start making major investments in
IT now at all levels, to upgrade their 2.5G/3G systems to the 4G environment.

Perhaps what has worked against IT in large environments and has fed the recent chorus
that “IT doesn’t matter” is the fact that it continues to be difficult for decision makers to be able
to get a good sense of what corporate IT resources (machine cycles, storage space, intranet use,
etc.) really cost on an allocated run-time basis, and how chargebacks can be equitably passed
back to the various business users across the company. The old saying that “a shoemaker’s son
always goes barefoot” seems, unfortunately, applicable here: IT runs all sorts of systems for the
firm, but seems to be having a hard time designing a system to run “on itself” to collect appro-
priate measures of usage of resources and do a true cost-based allocation (which would have the
benefit of providing true costs to business users and also be useful to drive new user behavior).
Two symptomatic predicaments could be as follows: (1) having to collect and deal with several
hundreds of IT usage measurements; or (2) being asked to develop a model for mainframe
usage and work four months to “reinvent the wheel,” when good measurements for mainframe
usage were already developed four decades ago. There is no need to collect several hundreds of
measurements: the IT industry could learn a lesson from the telecom industry in this regard. The telecom industry can charge back a user for the use of a broadband (say Asynchronous Transfer Mode) link between two distant cities by knowing just three things about the service that the user is employing: (1) access port speed, (2) average input rate, and (3) type of service, out of four possible service types. Why cannot the IT industry do something similar? The telecom carrier does not measure, for example, the instantaneous input rate, the specific kind of facility used to connect the user at site A, the specific kind of facility used to connect the user at site B, the amount of switch memory used at time $t$ along the way, the amount of electrical power used by the switches along the way, the BTUs generated by the switches along the way (which needs to be then conditioned via HVAC systems), the length of the cabling within each central office along the way between the digital cross-connect system and the broadband switch, the number of retransmits that the user does at layer 2, the number of retransmits that the user does at layer 4, etc. The telecom industry does not collect or deal with several hundreds of parameters to compute a reasonably accurate bill for broadband transmission. Why do some firms need to collect several hundred parameters to provide a reasonably accurate chargeback to the business users? This only serves to obfuscate the value that IT can provide to an enterprise. Instead, a few well-chosen well-understood parameters should be used. A versatile enterprise architecture framework will go a long way to address this issue.

However, enough said about the dubious position that IT does not matter. Because it does matter, the fundamental question is then, “How do Fortune 5000 companies deal with it in the most versatile and profitable manner?” Earlier, we stated that a firm requires a well-woven information-comprehensive infrastructure to meet contemporary business demands. The problem is that many companies have hundreds, thousands, and even tens of thousands of applications to accomplish this, along with physical infrastructure assets valued at hundreds of millions of dollars. In turn, this creates an environment that is difficult to manage, optimize, and migrate to a (future) target state. What firms need is a well-planned, well-designed, best-in-class IT capability. To that end, it is the enterprise architecture plan that defines the organization’s blueprint for optimal allocation of assets. This topic is the focus of this book.

1.2 IT Resources Requiring Proper Architectural Planning

Firms have made major investments in their IT resources in recent years. This text deals with optimizing the IT assets that a firm has deployed to run its business. This is accomplished, it is argued herewith, by properly architecting the IT environment, namely, by developing an enterprise architecture for all relevant assets, and then by implementing a roadmap for the proper realization of the architecture blueprint. At a broad level, enterprise architecture relates to understanding the universe of the distinct elements that comprise the enterprise and how those elements interrelate.

An enterprise, therefore, is any collection of departments or organizations that have a common set of goals/principles or a single bottom line. In that sense, an enterprise can be a whole corporation, a division of a corporation, a government agency, a single department, or a network of geographically distant organizations linked together by common objectives. Elements in this context encompass people, processes, business, and technology. Examples of elements include strategies, business drivers, principles, stakeholders, units, locations, budgets, domains, functions,
processes, services, information, communications, applications, systems, infrastructure,* and so on [IEA200501]. Hence, enterprise architecture is the collection of business processes, applications, technologies, and data that supports the business strategies of an enterprise.

We begin this subsection by providing some heuristic ballpark estimates on the “size of the opportunity scope.” This is information we have synthesized over the years in the context of first-pass quantification and modeling of IT assets. Although this information is approximate, it does show some of the macro-level trends.

Generally, we have found that for large companies the following applies at a broad level:

- For manufacturing, education, legal, and transportation, companies tend, on average, to spend 2–3% of their gross yearly revenues on the (yearly) IT budget.
- For marketing/service-based operations (publishing, reservation companies, hotels, average corporate establishment, etc.), companies tend, on average, to spend 3–6% of their gross yearly revenues on the (yearly) IT budget.
- For information-intensive operations (banks, brokerage, insurance, credit cards, etc.), companies tend, on average, to spend 6–10% of their gross yearly revenues on the (yearly) IT budget.†

As a first pass, and for the typical firm, about 40% of this figure is for run-the-engine (RTE) costs and 60% for development of new applications (sometimes called business process investments) and for internal business user liaison (for many firms “internal selling/business user liaison” equates to 20% of this last figure or less; for some firms the “internal selling/business user liaison” equates—abnormally—to 40–50% of this last figure.)

Generally, for the RTE portion, 27–33% (30% as average) tends to be for staff payroll, 21–25% (23% as average) for equipment (yearly amortized figures), 15–19% (17% as average) for software licenses, 10–14% (12% as average) for communication, 10–16% (13% as average) for external support services (not including carriers), 4–6% (5% as average) for facilities (e.g., physical data center, backup site, offices, etc.) and miscellaneous.

As noted, communication services tend to be in the 12% range of the IT budget, split as follows: 83–87% (85% as average) voice services; 13–17% (15% as average) data/Internet services.

Finally, the embedded hardware investment of a firm can be taken to be 3–5 times the yearly application development figures just quoted. The embedded applications investment can be taken to be 5–8 times the yearly application development figures just quoted.

All of this adds up to a lot of money. Table 1.1 provides an illustrative example. Again, this information is approximate, but it points to some general budgetary allocations. Because of the large budget figures involved, it makes sense to seek to optimize the environment and the architecture. Basically, all line items, including the staff line, can be addressed by the information contained in this book. Even a modest 5% improvement could save tens of millions of dollars in RTE expenses (e.g., $12, $16, and $20 million for lower-end, midrange, and higher-end, operation, respectively; see Table 1.1.)

* In this text, unless otherwise noted, the term infrastructure refers to the entire set of IT assets (software, hardware, data, etc.). The term physical infrastructure refers specifically to the physical infrastructure (the servers, the networks, the storage devices, etc.)
† Some researchers claim even higher spend rates.
<table>
<thead>
<tr>
<th>Table 1.1 Example of an Information-Intensive Firm (e.g., a typical financial firm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firm’s annual revenue</td>
</tr>
<tr>
<td><strong>Lower-end operation</strong></td>
</tr>
<tr>
<td>IT budget (yearly)</td>
</tr>
<tr>
<td>RTE</td>
</tr>
<tr>
<td>Staff</td>
</tr>
<tr>
<td>Amortized IT hardware</td>
</tr>
<tr>
<td>Software licenses</td>
</tr>
<tr>
<td>Communications</td>
</tr>
<tr>
<td>Voice</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>External services</td>
</tr>
<tr>
<td>Facilities, etc.</td>
</tr>
<tr>
<td>Development</td>
</tr>
<tr>
<td>Embedded hardware, range(^a)</td>
</tr>
<tr>
<td>Embedded applications, range(^b)</td>
</tr>
<tr>
<td><strong>Midrange operation</strong></td>
</tr>
<tr>
<td>IT budget (yearly)</td>
</tr>
<tr>
<td>RTE</td>
</tr>
<tr>
<td>Staff</td>
</tr>
<tr>
<td>Amortized IT hardware</td>
</tr>
<tr>
<td>Software licenses</td>
</tr>
<tr>
<td>Communications</td>
</tr>
<tr>
<td>Voice</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>External services</td>
</tr>
<tr>
<td>Facilities, etc.</td>
</tr>
<tr>
<td>Development</td>
</tr>
<tr>
<td>Embedded hardware, range(^a)</td>
</tr>
<tr>
<td>Embedded applications, range(^b)</td>
</tr>
<tr>
<td><strong>Higher-end operation</strong></td>
</tr>
<tr>
<td>IT budget (yearly)</td>
</tr>
<tr>
<td>RTE</td>
</tr>
<tr>
<td>Staff</td>
</tr>
<tr>
<td>Amortized IT hardware</td>
</tr>
<tr>
<td>Software licenses</td>
</tr>
<tr>
<td>Communications</td>
</tr>
<tr>
<td>Voice</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>External Services</td>
</tr>
<tr>
<td>Facilities, etc.</td>
</tr>
<tr>
<td>Development</td>
</tr>
<tr>
<td>Embedded hardware, range(^a)</td>
</tr>
<tr>
<td>Embedded applications, range(^b)</td>
</tr>
</tbody>
</table>

*Note:* Top—lower-end operation; Center—midrange operation; Bottom—higher-end operation.  
\(^a\) 3–5 times yearly amortizes IT hardware  
\(^b\) 5–8 times yearly development budget
What organizations should seek to achieve is optimal management of the environment by developing an architecture that supports industry best practices; this allows the organization to reduce RTE costs, and often, enhances the firm’s ability to meet evolving business needs and maximize functional enablements. IT assets, as noted earlier, encompass logical resources (applications, databases, etc.) and physical resources (processors, storage, networks, desktops, etc.).

1.3 What Is Enterprise Architecture?

We discuss more formally what an enterprise architecture is later in the book. For the time being, architecture can be seen as a blueprint for the optimal and target-conformant placement of resources in the IT environment for the ultimate support of the business function. As described in American National Standards Institute/Institute of Electrical and Electronics Engineers (ANSI/IEEE) Std 1471-2000, an architecture is “the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.” A metaphor can be drawn by thinking of a corporate/IT blueprint for the planning of a city or a large development. Specifically, then, the blueprint provides the macro view of how elements (roads, lots, utilities—read: platforms, networks, applications, applications’ logical components) fit, particularly in relation with one another. In this book we make use of terms such as “enterprise architecture, the current blueprint” (or “enterprise architecture,” or “enterprise architecture description”), and “enterprise architecture blueprint for the target state” (or “target state enterprise architecture” or “target state enterprise architecture description”). See Appendix 1.1 for a more formal definition.

The goal of enterprise architecture is to create a unified IT environment (standardized hardware and software systems) across the firm or all of the firm’s business units, with tight symbiotic links to the business side of the organization (which typically is 90% of the firm as seen earlier, at least by way of budget) and its strategy. More specifically, the goals are to promote alignment, standardization, reuse of existing IT assets, and the sharing of common methods for project management and software development across the organization. The end result, theoretically, is that the enterprise architecture will make IT cheaper, more strategic, and more responsive [KOC200502].

The purpose of enterprise architecture is to create a map of IT assets and business processes and a set of governance principles that drive an ongoing discussion about business strategy and how it can be expressed through IT. There are many different suggested frameworks to develop an enterprise architecture, as discussed later on. However, most frameworks contain four basic domains, as follows: (1) business architecture: documentation that outlines the company’s most important business processes; (2) information architecture: identifies where important blocks of information, such as a customer record, are kept and how one typically accesses them; (3) application system architecture: a map of the relationships of software applications to one another; and (4) the infrastructure technology architecture: a blueprint for the gamut of hardware, storage systems, and networks. The business architecture is the most critical, but also the most difficult to implement, according to industry practitioners [KOC200502].

Figure 1.1 depicts the macro view of the environment. On the left-hand side of the figure one can see external entities that may drive a firm. These include the customers, the market, the industry the firm is in, the opportunities that may exist or may develop, competitors, regulators, and investors, among others. A firm has an existing or newly developed business strategy. The firm also has an existing set of business assets. The goal is to develop the IT infrastructure to support an
end-state IT environment that enables, supports, and facilitates the business strategy. To this end, the enterprise may have developed an enterprise architecture, which is a blueprint of its information, systems, and technology environment. The blueprint also specifies the standards as related to these three categories (e.g., equipment standards, protocols standards, interface standards, etc.)

The firm may have developed the architecture using the industry mechanisms shown in the lower end of Figure 1.1. These include IT industry techniques and methods to develop an enterprise architecture; architecture principles; enterprise architecture IT industry standards; IT industry enterprise architecture frameworks and models; and architecture development tools.

As a new business strategy is developed by the firm, a new or modified enterprise architecture may be needed (this could be determined by a gap analysis). This enterprise architecture needs to take into account (as seen in the figure) the existing embedded base of IT assets, the existing enterprise architecture, the existing enterprise architecture standards, the firm’s principles and practices, the desired business strategy, and the available frameworks/tools to develop a new enterprise architecture or modify the existing one.

The output of this synthesis will be a set of derived IT strategies, a new/modified enterprise architecture, a new/modified set of enterprise architecture standards, a roadmap describing the IT
projects needed to effectuate (implement) the new architecture and achieve the target state, and a development/deployment plan. As the figure shows, there also are governance and effectiveness-assessment capabilities as well as an environment-monitoring function.

Part 1 of this book provides a survey of the architectural constructs that span the entire infrastructure, physical and logical; Part 2 focuses on actual technologies that can be used to develop an optimal physical infrastructure environment.

Layered frameworks and models for enterprise architecture have proved useful because layering has the advantage of defining contained, nonoverlapping partitions of the environment. There is a number of models/modeling techniques, for example, The Open Group Architecture Framework (TOGAF), the Federal Enterprise Architecture Framework (FEAF), and so on. However, there is at this time no complete industrywide consensus on what an architectural layered model should be, therefore various models exist or can be used. One case where standardization in the layered model has been accomplished is in the case of the Open Systems Interconnection Reference Model (OSIRM) published in 1984 by the International Organization for Standardization (ISO)*† (this model, however, only applies to communications). In the chapters that follow we discuss some formal models advanced or used in the industry to develop architecture planning tools. In the context of architecture, an important recent development in IT architecture practice has been the emergence of standards for architecture description, principally through the adoption by ANSI and the IEEE of ANSI/IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems; one of the aims of this standard is to promote a more consistent, more systematic approach to the creation of views (a view is a representation of a whole system from the perspective of a related set of concerns) [TOG200502]. However, the adoption of this model is still far from being universal.

As noted, there are about a dozen-and-a-half enterprise architecture frameworks and new ones are being added over time (see Table 1.2). There is even a book with the title How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating or Choosing an Enterprise Architecture Framework [SCH200501]. The most commonly used framework today, based on industry surveys, was the Zachman Framework, followed by an organization’s own locally developed frameworks, followed by TOGAF, and commercial-level Department of Defense Technical Reference Model (DoD TRM) (this covers about two-thirds of all enterprises.)

---

* The OSIRM is a set of internationally accepted standards that define a protocol model comprising seven hierarchically dependent layers. It is the foundation of protocol development work by the various standards agencies. A joint International Organization for Standardization (ISO)/International Telecommunications Union (ITU) standard for a seven-layer, architectural communication framework for interconnection of computers in networks. OSIRM-based standards include communication protocols that are mostly (but not totally) compatible with the Internet Protocol Suite, but also include security models, such as X.509, that are used in the Internet. The OSIRM layers, from highest to lowest, are (7) Application, (6) Presentation, (5) Session, (4) Transport, (3) Network, (2) Data Link, and (1) Physical. The model, originally developed in the 1980s, is defined in the following four documents: (1) ISO/IEC 7498-1:1994: Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model; (2) ISO 7498-2:1989: Information processing systems—Open Systems Interconnection—Basic Reference Model—Part 2: Security Architecture. (3) ISO/IEC 7498-3:1997: Information technology—Open Systems Interconnection—Basic Reference Model: Naming and addressing. (4) ISO/IEC 7498-4:1989: Information processing systems—Open Systems Interconnection—Basic Reference Model—Part 4: Management framework.

† Open Systems Interconnection (Networking) standards are defined by ISO to support the OSIRM architecture.
Fundamentally, all models seek in some way to make use of the concept of a generic service/object-oriented architecture\(^*\) (GSOA) in which sets of like functions are grouped into reusable service modules that can be described as objects; more complex capabilities are then built from appropriate assembly of these basic modules (just as, by analogy, matter is made up of various combinations of atoms of the elements). Although the idea of the service view is generally accepted at the “higher layers” of the model (e.g., the logical, computational, application, data, and business logic), we have advocated in recent years using the GSOA concepts also at the lower layers, where the actual IT technology resides (e.g., servers, networks, storage devices). This view can drive a move to a grid computing paradigm where physical IT assets are virtualized and distributed. Some of the enterprise architecture models have their genesis in the client-server model developed in the late 1980s and early 1990s. However, some of the concepts have been modernized; for example, the client could be a browser-based access device, the server could be a Web server, and the exchange protocol could be Hypertext Transfer Protocol (HTTP); or both entities could be a server running some service-providing/service-requesting Web Service (WS) and the exchange protocol be Simple Object Access Protocol (SOAP).

Practitioners, however, need to have a pragmatic rather than academic view of all of these models; otherwise, one could end up spending an inordinate amount of time over several years developing a framework model (e.g., with principles, strategies, decisions, guidelines, standards, alternatives, justifications, etc.) and have little concrete to show (some firms have spent in the range of 100 staff-years to develop such a model with relatively little to show in the end). An analogy here with the well-established OSIRM mentioned earlier is useful. First, it should be noted that

\(^*\) We employ the term GSOA to describe the general concept of service-oriented modeling of the enterprise architecture. We use the term SOA to refer specifically to the commercially available products to implement a GSOA paradigm, as discussed later in the book.
the model was quickly developed in the early 1980s. Second, the model was simple (a very thin standard). Third, the model has been standardized and universally accepted. Having said that, however, it needs to be noted that the raw OSIRM model by itself is of limited value; it would not have any major importance or consequence if it were not for the hundreds of supportive standards developed for Layer 1, Layer 2, Layer 3, etc., by all the standards-making organizations (not only ISO but also Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T), IEEE, ANSI, Internet Engineering Task Force (IETF), etc.) What is useful is the full complement (the apparatus created by all) of the supportive standards, not the model itself.

A related observation in this context is as follows: the ultrarigorous attempt to model an entire enterprise (business/informatics) function with any one of the available models may be an extremely tedious effort and may become a moot point when the business climate changes every eighteen months, or even every six months. Because of the effort involved in the rigorous application of the modeling languages, an enterprise architecture organization may invest large amounts of time in undertaking a pedantic exercise that produces out-of-date results by the time it is completed. Results from the world of fuzzy set theory show that sometimes it is better to be in the “ballpark” rather than being “ultraprecise.” We do agree that large-scale software development with fixed, well-established requirements (e.g., for things such as the next-generation jumbo jet, a nuclear power plant, a military system, or a space exploration platform) requires such rigor, but garden-variety business systems (e.g., order tracking, inventory system, customer records, etc.) may do fine with a somewhat more (but not totally) relaxed view. This is because requirements typically change even over a short period of time (new kinds of orders may be needed within six months, new kinds of items may be inventoried next year, new customer billing/information may be collected in nine months), and so an ultrarigorous effort at requirements documentation may not be worth it. Additionally, change control mechanisms may be difficult to implement over frequent variations over a short period of time. We are not suggesting not to go through the modeling effort; we are recommending not to spend hundreds of staff-years to develop a partial framework for the enterprise architecture.

A firm may have developed a full suite of architectures for the various framework layers or may only have a partially developed architecture, as illustrated in Figure 1.2. Figure 1.3 illustrates

Figure 1.2 Maturity of enterprise architecture development at a firm.
graphically the motivation for having an enterprise architecture: the top portion shows a rather simple application at a firm, where an architecture may be optional; the middle portion illustrates that over time the system and interrelations may grow more complex, and so an architecture blueprint is recommended; the bottom portion of the figure shows a full-grown application environment where an architecture blueprint is indispensable. Fortune 500 firms may have several dozens (if not hundreds) of applications with this type of complexity; trying to position oneself
Introduction

Strategically in this environment without an enterprise architecture plan is completely futile. At this juncture it is not just the large organizations that have adopted enterprise architecture: smaller organizations are also adopting this approach (however, the architecture maturity is at a higher level in larger firms than in smaller firms) [IEA200501]. Every organization that seeks to manage its IT complexity in a cost-effective manner for rapid system deployment should consider making the appropriate investments in enterprise architecture.

Figure 1.4 shows some basic events that trigger a refresh of an enterprise architecture.

A final observation: any enterprise architecture must be seen (designed, delivered, and internally sold) as a deliverable product, something that can be “touched and used” not just an abstract conceptualization. In the IT context, an architecture needs to be perceived (seen) by users and stakeholders almost like another IT system application: it must have inputs, outputs, functionality, built-in data, etc. A simple conceptualization is difficult to be seen as adding value. If one is to consider the enterprise architecture as a set of “blueprint guidelines on how to build things, particularly showing the relationship of one IT entity with another,” then the architecture should be perceived by the corporate user/developer to be like any other industry standard artifact (except that this applies internally to a firm rather than across the industry.) Such a standard is, in fact, a product: one can purchase a standard from ITU-T that is the definitive statement that a developer, say, of the Automatic Switched Optical Network (ASON), can use to develop globally conformant products. One can purchase a standard from the IEEE that is the definitive statement that a developer, say, of WiMax/802.16, can use to develop globally conformant products. So, we argue, the enterprise architecture description could well have the look-and-feel of an ISO, ITU-T, IEEE, ANSI, or IETF document with mandatory/optional capabilities, PICS Proforma, etc. If such ISO, IEEE, ANSI, IETF mechanisms are good enough to standardize products across an industry, or across several industries, or even across the world, why are these mechanism not good enough to standardize products inside a firm? Why does one need to reinvent, perhaps over several iterations, the set of architecture-supporting artifacts? This is what we meant earlier when we stated that there is, as of yet, not enough commoditization in IT: it is because often firms think that they are so different from one another that they have to reinvent how they undertake a common function, rather than use a standardized approach.

![Figure 1.4](image-url)  
**Figure 1.4** Some basic events that trigger a refresh of an enterprise architecture.
Next, we define a simple enterprise architecture model that we have used in recent years, which is depicted in Figure 1.5. This decomposition of the enterprise is modeled after TOGAF and is as follows:

- **Business Function**: This is a description of all business elements and structures that are covered by the enterprise.
- **Business Architecture**: An architectural formulation of the Business Function.
- **Information Function**: This is a comprehensive identification of the data, the data flows, and the data interrelations required to support the Business Function. The identification, systematization, categorization, and inventory/storage of information are always necessary to run a business, but these are essential if the data-handling functions are to be automated.
- **Information Architecture**: An architectural formulation of the Information Function via a data model.
- **(Systems/Application) Solution Function**: This is the function that aims at delivering/supplying computerized IT systems required to support the plethora of specific functions needed by the Business Function.
- **(Systems/Application) Solution Architecture**: An architectural definition of the (Systems/Application) Solution Function.
- **Technology Infrastructure Function**: The complete technology environment required to support the Information Function and the (Systems/Application) Solution Function.
- **Technology Infrastructure Architecture**: An architectural formulation (description) of the Technology Infrastructure Function.

![Diagram of enterprise architecture model, also showing architecture artifacts.](image-url)
These architecture sublayers are clearly related to one another via well-definable relations; integration of these sublayers is a necessity for a cohesive and effective enterprise architecture design. These layers are hierarchical only in the weak sense; hence, they can also be seen as domains (rather than layers per se.)

IT/networking security is also important, and firms need to have well-developed, comprehensive security architectures in place; this topic, however, is too extensive to be covered in this text.

Figure 1.6 partitions the IT space from an architectural perspective into logical resources, physical resources, and management resources. Physical resources in the Technology Layer provide the environment and services for executing applications; these resources encompass platforms (mainframe and midrange processors) along with hardware and operating system (OS) classifications; storage; desktops; and, networks (covering eight subcomponents). Notice the virtualization middleware, which we discuss later in the book. The Operations and Management Layer is a combination of processes and tools required to support the entire IT environment. It covers detection of faults and outages, configuration, administrative accounting, performance, and security.

As mentioned earlier, there are many models that can be used. The model of Figure 1.5 is loosely based on the Reference Model of Open Distributed Processing (ISO-RM-ODP) (ITU-T Rec. X.901-aka) ISO/IEC 10746-1 through ITU-T Rec. X.904 (aka) ISO/IEC 10746-4), which provides a framework to support the development of standards to support distributed processing in heterogeneous environments. RM-ODP uses an object-modeling approach to describe distributed systems. Two structuring approaches are used to simplify the problems of design in large complex systems: (1) the “viewpoints” provide a way of describing the system; and (2) the “transparencies” identify specific problems unique to distributed systems. Each viewpoint is associated with a language that can be used to describe systems from that viewpoint. The five viewpoints in RM-ODP are the following:

- Business Services Layer
- Information Layer
- (Systems) Solution Layer
- Technology Layer
- Operations/Management Layer

These architecture sublayers are clearly related to one another via well-definable relations; integration of these sublayers is a necessity for a cohesive and effective enterprise architecture design. These layers are hierarchical only in the weak sense; hence, they can also be seen as domains (rather than layers per se.)

IT/networking security is also important, and firms need to have well-developed, comprehensive security architectures in place; this topic, however, is too extensive to be covered in this text.

Figure 1.6 partitions the IT space from an architectural perspective into logical resources, physical resources, and management resources. Physical resources in the Technology Layer provide the environment and services for executing applications; these resources encompass platforms (mainframe and midrange processors) along with hardware and operating system (OS) classifications; storage; desktops; and, networks (covering eight subcomponents). Notice the virtualization middleware, which we discuss later in the book. The Operations and Management Layer is a combination of processes and tools required to support the entire IT environment. It covers detection of faults and outages, configuration, administrative accounting, performance, and security.

As mentioned earlier, there are many models that can be used. The model of Figure 1.5 is loosely based on the Reference Model of Open Distributed Processing (ISO-RM-ODP) (ITU-T Rec. X.901-aka) ISO/IEC 10746-1 through ITU-T Rec. X.904 (aka) ISO/IEC 10746-4), which provides a framework to support the development of standards to support distributed processing in heterogeneous environments. RM-ODP uses an object-modeling approach to describe distributed systems. Two structuring approaches are used to simplify the problems of design in large complex systems: (1) the “viewpoints” provide a way of describing the system; and (2) the “transparencies” identify specific problems unique to distributed systems. Each viewpoint is associated with a language that can be used to describe systems from that viewpoint. The five viewpoints in RM-ODP are the following:
1. The enterprise viewpoint, which examines the system and its environment in the context of the business requirements on the system, its purpose, scope, and policies. It deals with aspects of the enterprise, such as its organizational structure, that affect the system.

2. The information viewpoint, which focuses on the information in the system. How the information is structured, how it changes, information flows, and the logical divisions between independent functions within the system are all dealt with in the information viewpoint.

3. The computational viewpoint, which focuses on functional decomposition of the system into objects that interact at interfaces.

4. The engineering viewpoint, which focuses on how distributed interaction between system objects is supported.

5. The technology viewpoint, which concentrates on the individual hardware and software components that make up the system.

Having discussed this model, we alert the reader that the framework implied by Figures 1.5 and 1.6 is the one used in this text. Figure 1.7 shows how the key components of an architecture-enabled environment relate to one another.

Figure 1.8 illustrates some dos and don’ts in the process of developing or maintaining the enterprise architecture. Among other observations, the following are pertinent. Enterprise architecture must be more than just pretty pictures. Often, one sees lots of elaborate figures, charts, and presentations emerge from early enterprise architecture efforts at a firm, but unless the concepts are translated into actual decisions, migrations, and governance, such intriguing graphics will not lead to any concrete advancements and ameliorations. Enterprise architecture must help firms manage IT costs; it must help the organization decide where to make new IT investments, namely, where to “retain,” “retire,” or “rebuild” applications or infrastructure. Also, the architecture framework (just by) itself does not intrinsically save money for a firm: all of the follow-on artifacts must be developed, and, then, in turn applied to the environment, that is to say, implemented through funded efforts.
1.4 Trends in Enterprise Architecture and Data Center Environments

In this section we highlight briefly some of the industry trends that are perceivable in the area of enterprise architecture and technology architecture (specifically, data center environments) at press time.

1.4.1 Enterprise Architecture

The following press time observations from CIO magazine do a good job of describing the environment and trends that impact enterprise architecture [KOC200501,KOC200502]:

Enterprise architecture has been around since the mid-1980s, but it has only recently begun to transform from an IT-centric exercise in mapping, controlling, standardizing and consolidating into something new—a function entrusted with creating a permanent process for alignment between IT and the business. This mandate is new, and forward-looking firms are just beginning to move beyond the IT architecture umbrella (which usually includes the data and infrastructure architectures).

However, practitioners believe that expanded role is not going to be successful if the Chief Information Officer (CIO) has not established institutionalized, repeatable governance processes that promote IT and business alignment at the highest level. The CIO has to report to the Chief Executive Officer (CEO) for enterprise architecture to be an alignment force. If CIOs cannot get access to strategic discussions, they cannot follow through on the next governance prerequisite to the new enterprise architecture: IT investment prioritization. If IT does not get access to the business at the highest level, IT is not going to get a clear idea of what the business really wants from IT. With a reasonable IT investment process in place at the highest level, enterprise architecture has a chance to move beyond IT architecture. But only if those who are part of the highest-level investment process are comfortable ceding some power and control to the architects. They need to make architecture review a condition for investment approval.
The executive committee holds the carrot (money) and they wave the stick, which is that before you come to us asking for money, you need to go through the architecture review process first. It is the only way to get midlevel business people to care about enterprise architecture. They do not care about IT architecture efforts that consolidate infrastructure or standardize applications because the savings do not impact them personally. They are focused on process. If the architects control investment and project review, they will start to care. Then it is up to the CIO to make sure that business people do not wind up hating the architects.

And enterprise architecture cannot become yet another effort to gain credibility for IT with the business. … (one may) focus the enterprise architecture effort on IT architecture first before tackling the business and application architectures (i.e., before getting the business involved) because the business wanted proof that IT could deliver before giving up business people’s time to the effort.

… Enterprise architecture has to offer more products and benefits to be accepted by the business. Architecture alone is not a viable sell. Service Oriented Architecture (SOA) is a great way to build enthusiasm for architecture among business people because it reduces IT to a vocabulary they can understand. ERP, CRM, legacy systems and integration are hidden beneath composite applications with names like “get credit rating” and “get customer record.” Reuse becomes real and starts to impact midlevel business people directly because they can see how much faster they get new functionality from IT since “get customer” became a service in a repository that any developer can access when the need arises.

Advances in integration technology—primarily intelligent and flexible middleware and Web Services—are providing new ways for designing more agile, more responsive enterprise architectures that provide the kind of value the business has been seeking. With these new architectures, IT can build new business capabilities faster, cheaper and in a vocabulary the business can understand. These advances are giving new life to a couple of old concepts that could inspire new enterprise architecture efforts and revive failing ones. The first concept is services, better known today as SOA. SOA provides the value to the business that in the old enterprise architecture was rarely more than a vague promise. The idea behind services is simple: Technology should be expressed as a chunk of the business rather than as an arcane application… CIOs at large companies estimate that they may have on the order of 200 services sitting in a repository on their intranets, ranging from things like “credit check” to “customer record.” Businesspeople can call for a service in a language they can understand, and IT can quickly link these with other services to form a workflow or, if need be, build a new application. These applications can be built quickly because complex, carefully designed interfaces allow developers to connect to the services without having to link directly to the code inside them. They do not even have to know how the service was built or in which type of language it was written. The second concept currently driving enterprise architecture is events. Pioneered by telecommunications carriers and financial services companies, this involves using IT systems to monitor a business process for events that matter—a stock-out in the warehouse, for example, or an especially
large charge on a consumer’s credit card—and automatically alert the people best equipped to do something about it. Together, services and events are revolutionizing the design of enterprise architecture, providing the kind of flexibility and value that CIOs, CFOs and CEOs have been looking for (but rarely have found) all these years. Services and events do not define enterprise architecture, but they can form its new core. CIOs who build enterprise architectures without a services-and-events approach will miss an opportunity to address the two most enduring—and accurate—complaints that the business has leveled at IT: slowness and inflexibility.

...In this evolving paradigm, enterprise architects have the opportunity to become the face of IT to the organization, consulting with business people to help them better articulate functional specifications. Technologies such as middleware, grid computing, Web Services (WS), enterprise architecture tools, and Business Process Management (BPM) tools are maturing to the point where architects can have a real strategic impact on the business and on IT. But not if governance has not been tackled first. CIOs need to do more to support their architects—the people who represent the future of the IT.

1.4.2 Technology Architecture

Although the entire IT environment greatly benefits from an architecture-based approach, as we will see in the next seven chapters, the second half of the book focuses principally on the technology infrastructure portion of the IT portfolio (the technology infrastructure architecture). This is relevant because as we saw earlier, a lot of the actual budget line-items are tied to this layer of the architecture. A press time survey of companies shows that having a technology architecture is as common as having the enterprise architecture as a whole* † [IEA200501].

In the early 1990s we published a fairly comprehensive book entitled Enterprise Networking—From Fractional T1 to SONET, Frame Relay to BISDN [MIN199301]. The text examined the plethora of evolving networking technologies that could be employed by large institutions to build out the (intranet) networking portion of the technology architecture. This present text extends the scope by examining late-decade networking, platform, and storage technologies. Also, this text extends a discussion of the enterprise architecture topic that was started by the author in a chapter entitled “Service Oriented Architecture Modeling: An Integrated Approach to Enterprise Architecture Definition That Spans the Business Architecture, the Information Architecture, the Solution Architecture, and the Technology Architecture,” contained in the recently published Handbook of IT and Finance [MIN200701].

* The cited study obtained the following results: 15% of the surveyed companies had an enterprise architecture; 15% of the companies had a technology infrastructure architecture; 15% of the companies had a security architecture; 14% of the companies had an information-systems (solution, application) architecture; 13% of the companies had an information architecture/data model; 11% of the companies had a software architecture; and 10% of the companies had a business architecture.

† The cited study obtained the following results: 17% of the surveyed companies had an enterprise architecture; 14% of the companies had a technology infrastructure architecture; 10% of the companies had a security architecture; 14% of the companies had an information-systems (solution, application) architecture; 15% of the companies had an information architecture; 13% of the companies had a software architecture; and 12% of the companies had a business architecture.
New data center technologies such as blade servers, grid computing, IP storage, storage virtualization, tiered storage, Internet Fibre Channel Protocol (iFCP), Internet Small Computer System Interface (iSCSI), 10 Gigabit Ethernet (10GbE), storage attachment (only) via Storage Area Networks (SANs), MultiProtocol Label Switching backbone services, metro Ethernet services, and Voice-Over-IP (VoIP)/integrated networks are now become prominent for the technology layer of the enterprise architecture [MAC200501].

The following press time observations from a column for NetworkWorld by the author describes the technology environment and trends that likely will impact the technology architecture of an enterprise in the near future [MIN200601]:

Virtualization is a well known concept in networking, from Virtual Channels in Asynchronous Transfer Mode, to Virtual Private Networks, to Virtual LANs, and Virtual IP Addresses. However, an even more fundamental type of virtualization is achievable with today’s ubiquitous networks: machine cycle and storage virtualization through the auspices of Grid Computing and IP storage. Grid Computing is also known as utility computing, what IBM calls on-demand computing.

Grid computing is a virtualization technology that was talked about in the 1980s and 1990s and entered the scientific computing field in the past ten years. The technology is now beginning to make its presence felt in the commercial computing environment. In the past couple of years there has been a lot of press and market activity, and a number of proponents see major penetration in the immediate future. IBM, Sun, Oracle, AT&T, and others are major players in this space.

Grid computing cannot really exist without networks (the “grid”), since the user is requesting computing or storage resources that are located miles or continents away. The user needs not be concerned about the specific technology used in delivering the computing or storage power: all the user wants and gets is the requisite “service.” One can think of grid computing as a middleware that shields the user from the raw technology itself. The network delivers the job requests anywhere in the world and returns the results, based on an established service level agreement.

The advantages of grid computing are the fact that there can be a mix-and-match of different hardware in the network; the cost is lower because there is a better, statistically-averaged, utilization of the underlying resources; also, there is higher availability because if a processor were to fail, another processor is automatically switched in service. Think of an environment of a Redundant Array of Inexpensive Computers (RAICs), similar to the concept of Redundant Array of Inexpensive Drives (RAIDs).

Grid Computing is intrinsically network-based: resources are distributed all over an intranet, an extranet, or the Internet. Users can also get locally-based virtualization by using middleware such as VMware that allows a multitude of servers right in the corporate Data Center to be utilized more efficiently. Typically corporate servers are utilized for less than 30–40% of their available computing power. Using a virtualization mechanism the firm can improve utilization, increase availability, reduce costs,
and make use of a plethora of mix-and-match processors; at a minimum this drives to server consolidation.

Security is a key consideration in grid computing. The user wants to get its services in a trustworthy and confidential manner. Then there is the desire for guaranteed levels of service and predictable, reduced costs. Finally, there is the need for standardization, so that a user with an appropriate middleware client software can transparently reach any registered resource in the network. Grid Computing supports the concept of the Service Oriented Architecture, where clients obtain services from loosely-coupled service-provider resources in the network. Web Services based on SOAP and Universal Description, Discovery and Integration (UDDI) protocols are now key building blocks of a grid environment.
Summary/Interpretation

An enterprise architecture at any instance \( n \) is an exact description of the functionality, the interfaces, the data, and the interface protocols supported by the (partitioned) set of functional elements in the environment, as of time instance \( n \). Call that simply “enterprise architecture, the current blueprint,” or just “enterprise architecture.” One is then able to provide a description of a desired target state at time \( n + 1 \) or \( n + j \) in terms of possibly new functionalities, new interfaces, new data, new interface protocols, and possibly new partitioning of the set of functional elements. Call that the “enterprise architecture blueprint for the target state,” or just “target state enterprise architecture.”

Notice once again that (1) the strategy decision for wanting to define a certain “target state enterprise architecture” is actually external to the architecture itself (the architecture simply states how all the parts work together); and (b) the strategy steps needed to actually transition to the “target state enterprise architecture,” the roadmap and the development plan, are also external to the architecture itself.
However, for practical considerations the following will be considered part of the enterprise architecture:

- The enterprise architecture description (of the current or target state)
- The enterprise standards set
- The enterprise approved equipment list
- The roadmap, along with (migration) strategies
Chapter 2

Enterprise Architecture Goals, Roles, and Mechanisms

What does a corporate or institutional enterprise architecture corporate function address? What are the available architecture frameworks? What can tools do?

This chapter covers, at a macro level, the following: (1) techniques for developing an enterprise architecture by using an architecture framework; (2) tools for using the framework to instantiate the architecture; and (3) mechanisms for actually building out the architecture. We begin in Section 2.1 by reviewing some motivations as to why enterprise architectures are needed and some general approaches to the development effort, e.g., guiding principles and architecture techniques (see the lower portion of Figure 2.1). We then focus generically in Section 2.2 on architectural frameworks (e.g., Federal Enterprise Architecture Framework, Zachman Framework; see center-lower portion of Figure 2.1). In Section 2.3 we briefly discuss some governance approaches.

This discussion is completed over the next six chapters. Chapters 3 and 4 focus more directly on The Open Group Architecture Framework (TOGAF) and the Zachman Framework, respectively. Chapter 5 looks at what official architectural standards are available. Chapter 6 looks at tools (e.g., ARIS). Chapter 7 looks at business process modeling (BPM), relational/process modeling (UML), and requirements gathering and analysis. Chapter 8 looks at architecture fulfillment via service-oriented architecture (SOA) modeling; SOA is a method (but not the only one) to actually implement a portion or the entire enterprise architecture.

2.1 Enterprise Architecture

In this section we review some motivations as to why enterprise architectures are needed and some general approaches to the development effort, e.g., guiding principles and architecture techniques.
2.1.1 Description of Enterprise

For the purpose of this book enterprise is any collection of corporate or institutional task-supporting functional entities that have a set of common goals or a single mandate. In this context, an enterprise is, but is not limited to, an entire corporation, a division or department of a corporation, a group of geographically dispersed organizations linked together by common administrative ownership, a government agency (or set of agencies) at any level of jurisdiction, a group of government agencies, and so on. This also encompasses the concept of an extended enterprise, which is a logical aggregation that includes internal business units of a firm along with partners and suppliers (sometimes customers are also considered part of an extended enterprise).

Large organizations and government entities may comprise multiple enterprises; however, there is often a lot in common about overall mission, and, hence, the ensuing need for (minimally) interoperable information systems, consistent data representations/extracts; in turn, this drives the desire for a common architecture framework. One of the many examples that could be given deals with having a set of common radio frequencies (and supportive infrastructure, whether that be physical, logical, or IT system level) that could be used by first responders. A common architecture supports cohesive multi-organizational operation. A common architectural framework, in turn,
can provide a basis for the development of an architecture repository for the integration and reuse of models, designs, and baseline data.

In this book, in conformance with common parlance in the enterprise architecture context, the term enterprise is taken to denote the enterprise itself, as just defined, as well as all of its information systems and operational data; furthermore, the term is also employed to discuss a specific business domain within the enterprise at large, along with all the subtending information systems and data to support that specific function. In both cases, the architecture crosses multiple systems, and multiple functional groups within the enterprise.

### 2.1.2 Definition of Architecture

An enterprise architecture is a plan of record, a blueprint of the permitted structure, arrangement, configuration, functional groupings/partitioning, interfaces, data, protocols, logical functionality, integration, technology, of IT resources needed to support a corporate or organizational business function or mission. Typically, resources that need architectural formulations include applications, security subsystems, data structures, networks, hardware platforms, storage, desktop systems, to name just a few. The term enterprise is taken to be consistent with that given in Chapter 1. As described in ANSI/IEEE Std 1471-2000, an architecture is “the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.”

As we discussed in Appendix 1.1 of Chapter 1, the following will be considered here as being part of the enterprise architecture:

- The enterprise architecture description (of the current or target state)
- The enterprise architecture standards set
- The enterprise approved equipment list
- The roadmap along with (migration) strategies

Areas where firms may look to develop an architecture include the following:

- **Business function**: This is a description of the all business elements and structures that are covered by the enterprise. These aspects capture the business logic functionality, specifically business processes. (In the language of Appendix 1.1 this would be called Business Functional Group.)
- **Information function**: This is a comprehensive identification of the data, the data flows, and the data interrelations required to support the business function. The identification, systematization, categorization, and inventory/storage of information are always necessary to run a business, but these are essential if the data-handling functions are to be automated. (In the language of Appendix 1.1 this would be called Information Functional Group.)
- **(Systems/application) Solution function**: This is the function that aims at delivering/supplying computerized IT systems required to support the plethora of specific functions needed by the business function. (In the language of Appendix 1.1 this would be called (Systems/Application) Solution Functional Group.)
- **Technology infrastructure function**: The complete technology environment required to support the information function and the (systems/application) solution function. (In the language of Appendix 1.1 this would be called Technology Functional Group.)
Subareas where firms may look to develop an architecture include, but are not limited to, the following:

- Presentation aspects of IT applications: These aspects define the logical interfaces between applications, and between applications and user systems.
- Application integration aspects: These aspects describe the interconnection mechanism of the business logic as implemented in IT applications (this supports the specification of interfaces, and messaging).
- Information management aspects: These aspects deal with the design and construction of data structures; data entities and relationships between data entities are defined via a data model.
- Technology infrastructure aspects: networks, platforms, storage, desktops, etc.

Architectures include the following (potentially along with other architectures to support the aspects just described):

- Business architecture: an architectural formulation of the business function.
- Information architecture: an architectural formulation of the Information Function via a data model.
- (Systems/application) solution architecture: an architectural definition of the (systems/application) solution function.
- Technology infrastructure architecture: an architectural formulation (description) of the technology infrastructure function.

The preceding paragraph points out that a firm will typically have several interlocking architectures that cover various aspects of the IT/enterprise function. For example, a presentation services architecture, an application integration architecture, an information/data architecture, a network architecture, a hardware platform architecture, etc. It is not possible or desirable to develop a massive architecture that covers everything. Figure 2.2 depicts the concept of multiple subarchitectures pictorially.

2.1.3 Motivations for Having an Enterprise Architecture

Enterprise architecture drives to standardization, which in turn drives to commoditization, as discussed in Chapter 1. Standardization results in (1) lower run-the-engine (RTE) costs and (2) faster rollout of a function, whether this is an atomic function such as “configure a server,” “configure a storage partition,” or a more complexly integrated function, such as a new software application. Figure 2.3 shows pictorially that although a completely rigid rollout of a standard may not always be optimal, a position where there is a large degree of standardization may be the overall optimal strategy. Architecture work is important for at least three reasons: It (1) enables communication among stakeholders, (2) facilitates early design decisions, and (3) creates a transferable abstraction of a system/environment description [FER200401]. The enterprise architecture will help firms manage their IT costs; it helps the organization decide where to make new IT investments, namely, where to retain, retire, or rebuild applications or infrastructure.

As information technology extends its reach into every facet of the enterprise and as the panoply of systems, applications, databases, servers, networks, and storage appliances become highly
Applications use a small set of unified solutions, defined in a "corporate standard" e.g., a single device with modularized interfaces.

Each application/user community selects the equipment/solution that minimizes their local/departmental cost.

All applications/user communities use the same "corporate standard" solution (one single/unique solution).

Figure 2.2 Collection of subarchitectures.

Figure 2.3 Optimality locus.
interconnected and highly interdependent at the logical and at the physical level, IT practitioners need to recognize the growing importance of enterprise architecture to the continued success and growth of the firm [ZIF200501]. Enterprise architecture work, when done correctly, provides a systematic assessment and description of how the business function operates at the current time; it provides a “blueprint” of how it should operate in the future, and, it provides a roadmap for getting to the target state. For example, for the U.S. Department of the Navy (DON) Enterprise Architecture development has been driven by a technology change, the implementation of the Navy Marine Corps Intranet (NMCI), and a mandate to reduce its application portfolio by 95%. The Application Rationalization Review conducted under the auspice of enterprise architecture initiatives indeed reduced the Navy and Marine Corps application portfolio from 100,000 applications to 60,000 applications; the final phase of these reviews will reduce the application portfolio to approximately 5000 supported applications [FEA200501].

Figure 2.4 shows how the IT budget maps (approximately) to the layers of the architecture and the supported functions. As can be seen, large fractions of the budget are allocated to activities related to the technology layer and to the development layer. Given the size of these allocations, it is useful to have a planned approach to the management of these IT resources.

Table 2.1 depicts the actual perceived value for the development of enterprise architecture, based on an extensive press time survey of large institutions around the globe and in several industry segments [IEA200501]. Table 2.2 identifies issues that are expected to be resolved by the development of enterprise architectures.

![Figure 2.4 Mapping of IT budget to IT layers.](image-url)
E-Government initiatives are driving the introduction of enterprise architecture. At press time about 1/4 of government agencies sampled were undertaking IT architectural efforts and about 1/6 of financial companies were also doing so; business segments such as insurance, telecom, utilities appear to be lagging, with only 1/50 of the establishments in each of these segments engaged in such efforts [IEA200501]. In the industry the function typically reports to the IT director/manager (in about a third of the companies that do have an architecture group); to the CIO (in about another third of the companies); and to a business manager, the CTO, the CEO, or the board of directors (in the rest of the companies). However, some see a recent shift of responsibilities from the CIO to a higher level, such as board members, or a shift to a federated environment that directly involves business managers.

What are the risks of not developing an enterprise architecture? Without enterprise architecture work, the following risks exist, among others:

- Locally optimal, rather than globally optimal solutions
- Expensive, nonshared, RTE-intensive solutions
- Closed vendor/proprietary environments—little leverage

### Table 2.1  Reasons Given for the Perceived Value of the Development of Enterprise Architecture in Large Corporations

<table>
<thead>
<tr>
<th>Reason</th>
<th>Percentage of Companies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports decision making</td>
<td>16</td>
</tr>
<tr>
<td>Helps manage IT portfolio</td>
<td>14</td>
</tr>
<tr>
<td>Delivers blueprints for change</td>
<td>14</td>
</tr>
<tr>
<td>Helps manage complexity</td>
<td>12</td>
</tr>
<tr>
<td>Supports systems development</td>
<td>12</td>
</tr>
<tr>
<td>Delivers insight and overview of business and IT</td>
<td>11</td>
</tr>
<tr>
<td>Supports business and IT budget prioritization</td>
<td>11</td>
</tr>
<tr>
<td>Supports (out/in) sourcing</td>
<td>5</td>
</tr>
<tr>
<td>Helps affecting mergers and acquisitions</td>
<td>3</td>
</tr>
<tr>
<td>Other</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 2.2  Issues That Firms Expect Enterprise Architecture to Help Tackle On

<table>
<thead>
<tr>
<th>Specific Goals</th>
<th>Percentage of Companies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business–IT alignment</td>
<td>20</td>
</tr>
<tr>
<td>Business change</td>
<td>15</td>
</tr>
<tr>
<td>Transformation roadmap</td>
<td>15</td>
</tr>
<tr>
<td>Infrastructure renewal</td>
<td>12</td>
</tr>
<tr>
<td>Legacy transformation</td>
<td>11</td>
</tr>
<tr>
<td>ERP implementation</td>
<td>11</td>
</tr>
<tr>
<td>Application renewal</td>
<td>10</td>
</tr>
<tr>
<td>Mergers/acquisition</td>
<td>4</td>
</tr>
<tr>
<td>Other</td>
<td>2</td>
</tr>
</tbody>
</table>
Solution band-aiding for the short term, but constraining in the long term
- Nonadherence to standards; complex solutions; plethora of one-off designs

Even if an organization outsources some (or all) of its IT functions and runs a set of these functions with an application service provider (ASP)-like arrangement, without architecture work the following risks exist:

- Vendor enslavement
- Suboptimal service from a cost, feature, and portability perspective
- Excessive, unhealthy reliance on vendor, which may not have a firm’s best interests at heart
- Failure to follow industry standards or regulatory compliance
- Not achieving best-in-class/best-in-breed solutions
- Inappropriate visibility due to nonoptimal service level agreement (SLA) metrics

Even if an organization outsources some (or all) of its IT functions and runs a set of these functions with an asset-retention arrangement (in which the sources provide the personnel to run the function but the firm still owns the IT assets), without architecture work the following risks exist:

- Infrastructure becomes outdated and expensive
- Expensive, nonshared, RTE-intensive solutions
- Excessive, unhealthy reliance on vendor, which may not have firm’s best interests at heart
- Failure to follow industry standards or regulatory compliance
- Not achieving best-in-class/best-in-breed solutions

In cases involving outsourcing, the planners need to take into account the end-to-end view and have the synthesis of an “extended enterprise” in mind. Here, certain portions of the environment will be directly controllable by the firm, whereas others will only be indirectly controllable. It is important to note, however, that in many (if not most) instances it is undesirable, in the long term, to not exercise any control even on the indirectly controllable portion of the operation.

The following are often-cited [ZIF200501] motivations/observations about enterprise architecture:

- Enterprise architecture is fundamental for successful participation in the global interaction of modern enterprises.
- Enterprise architecture is the principal structural mechanism for:
  - Establishing a basis for assimilating high rates of change
  - Advancing the state-of-the-art in enterprise design
  - Managing the knowledge base of the enterprise
  - Integrating the technology (automated or nonautomated) into the fabric of the enterprise
- Enterprise architecture is universal—that is, every enterprise of any substance and any expectation of longevity needs to employ architectural concepts.
- Enterprise architecture is cross-disciplinary, requiring integration of diverse skills, methods, and tools, within and beyond the technology community.
2.1.4 Role Enterprise Architecture Group

There are two polar-opposite approaches to the organizational role of the enterprise architecture function, along with something in between:

1. Mandate that absolutely no project can be approved for deployment if it does not conform to an architectural standard or go through an architectural assessment. Companies in this arena are either companies that have several decades of process-development history (but not limited to large manufacturing establishments), or companies that have a large installed base and so it make sense to test and validate a system for conformance before it is deployed in the field (e.g., traditional telecom carriers).

2. Strictly run the enterprise architecture function on a “pure advocacy basis”: the group has no budgetary control and no (or very weak) enforcement control. Here, in effect, adherence to the architecture blueprint by the larger IT organization or by the business users, is pretty much voluntary; clearly, in this environment the function of the enterprise architecture group, unfortunately, often has relatively little overall impact. Companies in this arena tend to be younger companies that have grown to the ranks of Fortune 500 rather quickly and may only have a corporate history going back a decade or two.

The “in-between” formulation between these two extremes would say that firm placed IT subareas in two categories. For subareas in Category A the firm mandates that no project can be approved for deployment if it does not conform to an architectural standard or go through an architectural assessment. For subareas in Category B the firm runs the enterprise architecture function on an advocacy basis. It should be noted that the same kind of diagram as Figure 2.3 is applicable here: it would be optimal for the firm if a major portion (but not necessarily all) of the subareas were classified as Category A. We call this environment a Well Thought-Out Environment (WTE).

There is a fine balance between having an enterprise architecture group that “has no teeth” and one that is perceived to “have too much bite”; however, if the architecture group is to be successful, such a balance must be achieved. In some companies a developer or planner who wants to spend more than a quarter-of-a-million dollars on a project must get approval from the architecture group: to secure the funds, project leaders have to submit to a review process where the architects decide whether the project’s goals and technology fit into the firm’s overall business and technology strategy [KOC200502].

IT organizations need to take more leadership within the business context and achieve the “proper level of respect/confidence” within the organization. Respect and confidence are achieved by delivering products and services on a cost-effective, timely, and uncomplicated manner. IT cannot be “just taking orders” from the business side and constantly be in a “begging mode” where all they do is advocating this or advocating that; instead, IT must develop a corporate posture and positioning, based on sustained credibility and success, to be able to mandate various (architectural) solutions or approaches, and not just be in the advocacy role.

At first brush, standardizing, mapping, and controlling IT assets does not make the business more flexible, capable, or profitable for the immediate next quarter; as a result, IT architecture efforts at some firms fail or become completely IT-centric [KOC200502]. However, an analysis of (all sorts of) industry standardization efforts over the past 20 years shows unambiguously that not only do these efforts come to fruition in the sense of reducing costs over a period of perhaps three to five years, but entirely new functional enablements are fostered and triggered by the introduction of broadly available standardized techniques, platforms, data formats, protocols, and so on.
In a well-thought-out environment, the enterprise architecture function develops and then “promulgates” high-level architectural requirements to in-house IT-services or to sourcing vendors to ensure best-in-class, industry-grade, reliable, open, measurable, portable, and replicable service environments. These guidelines aim at maximizing a firm’s business/IT flexibility at any future point in time.

As a point of reference, Figure 2.5 depicts a synthesis of the enterprise architecture function as described in the FEA F [FEA200503].

In a well-thought-out organizational environment, when a new business imperative presents itself, a (high-level) enterprise architecture target is developed and architecture specifications (standards, or at least guidelines) are published; these specifications are then used by internal developers or by vendor management and sourcing vendors (when operating in an outsourcing mode) to deliver business service. The architecture is documented and published formally, with the goal of providing a “self-service” capability to consumers of these technical plans (e.g., developers).

The enterprise architecture function also develops SLA criteria. Enterprise architecture focuses on the form/syntax/scope of SLA (the business side of the firm would likely select actual values.) Enterprise Architecture will typically work closely with Vendor Management to implement solutions that support business operations in a symbiotic manner; it may assist in development of engineering specifications, RFPs, procurement plans, but in a consultative/support role.

In summary, enterprise architecture bears the fiduciary responsibility of helping the firm achieve the following:

1. Business and technology alignment:
   - Drive to provide a clear connection between the firm’s business strategy and architecture work/deliverables
   - Maximize business opportunity/capabilities via open/non-vendor-proprietary solutions
2. Recommend and publish medium-term and longer-term strategic directions based on business imperatives
− Undertake all pertinent technical due diligences at the enterprise architectural level

2. Develop a self-service architecture function (based on automated PICS Proforma mechanisms):
− Recommend and publish medium-term and longer-term strategic directions that can be accessed and assessed by stakeholders in a simple and direct manner
− Support adherence to enterprise standards
− Manage internal governance of introduction of new services

3. Achieve optimality:
− Secure optimal use of IT assets in the enterprise
− Maximize business opportunity/capabilities via open/non-vendor-proprietary solutions
− Achieve immediate/ongoing RTE savings via deployment of state-of-the-art/virtualized/shared assets and technologies

Some firms wonder if there is a need for enterprise architecture if the firm has outsourced (a certain portion of) the IT environment (either as an ASP model or as an asset-retention model). Indeed, enterprise architecture continues to have a role because from the perspective of the ultimate customer of the firm, it is the firm that provides them a service, even if a supplier handles some portion of the business process. In other words, the firm in question owns the entire business process end-to-end in the eyes of the customers even though some parts of the process are executed by a supplier. Therefore, a pragmatic architected end-to-end solution is required.

For example, if a customer of Bank X calls the bank’s customer service number to review his or her account, and the telephone connection is noisy because the bank has outsourced the Contact Center to a large Contact Center provider Y, which in turn uses a hub-and-spoke Voice-over-IP (VoIP) service from carrier Z to connect remote or international sites where the provider's agents are located, the customer of Bank X will perceive Bank X as providing poor service, not the unknown suppliers Y or Z.

It follows that the firm’s business processes and the underlying technology must be architected for end-to-end integrity. Some call this an “extended enterprise” environment. The roles of the enterprise architecture sourcing environment include the following:

- The enterprise architecture provides a holistic expression of the enterprise’s key strategies and their impact on business functions and processes, taking the firm’s sourcing goals into explicit consideration.
- The enterprise architecture assists the firm to establish technical guidelines of how the “service delivery function” (whether in-sourced, partially sourced, or fully sourced) needs to operate to deliver reliable, effective business services:
  - Establishes risk-monitoring mechanisms/flags (for example, over the past 15 years many sourcing deals have gone sour; how does the firm track/avoid predicament?)
  - Generates technical guidelines of how the “service delivery function” makes optimal use of IT assets, thereby maximizing cost-effectiveness, flexibility, and availability (for example, what happens if the supplier is only interested in profitability and fails to have a technically current/transferable technology?)
  - Generates technical guidelines on how to maximize business opportunities though open/non-vendor-proprietary solutions (for example, what happens if the supplier locks the firm in with a completely closed technology that nobody else has or is familiar with?)
Develops high-level architectures that foster and engender best-in-class/best-in-breed service environments at sourcing partners (or in-sourced departments) to retain technically sound/nonobsolescent environments (for example, what happens if the supplier uses questionable/speculative technology such as a nonsecure connectivity service, an unreliable carrier, or outdated equipment?)

Development of SLA schema/taxonomies/syntaxes to monitor that the service is achieved. Also, identify adequate reporting metrics (what happens if we fail to identify/enumerate the proper atomic components required for service unambiguous visibility?)

Helps develop new technical business processes (for example, existing processes will become outdated/inefficient as a firm moves to a sourced environment. Who best to develop them but Enterprise Architecture?)

Establishes a success-oriented technical environment (for example, how do we know that the vendor “is pulling wool over the firm’s eyes” by promising something that in fact cannot be delivered?)

Helps keep the vendor “technically honest” (for example, since time immemorial vendors have been self-serving; can we afford to give them carte blanche?)

Enterprise Architecture assists the firm to establish technical governance principles of how the “service delivery function” needs to design the service environment (at a high level) to meet technical and regulatory thresholds:

Establish pertinent technical due-diligences mechanisms at the architectural level (consider an anecdotal example of wanting to source a truck-driving function. The firm would want to establish rules such as: drivers must have more than ten years’ experience; drivers must not have been given more than two traffic tickets in ten years; drivers must not have had more than one accident in ten years; the firm shall carry business insurance; the trucks shall not pollute the environment and give bad publicity to the firm; etc.)

The business side of the firm clearly understands the kind of solutions required to meet business/financial imperatives. Hence, the role of Enterprise Architecture is to (1) determine which architectures are able to maximize their flexibility, positioning, and competitiveness; (2) articulate the value of these architectural visions; (3) develop appropriate technical approaches; and (4) socialize technical work through the IT and user organizations to secure sponsorship (and funds) to implement the roadmap.

In developing appropriate technical approaches, Enterprise Architecture should develop: (1) high-level architecture descriptions; along with (2) roadmaps and implementation plans. For environments where outsourcing is a factor, Enterprise Architecture should develop (1) service definition guidelines; (2) service acceptance principles; (3) service integrity criteria; (4) SLA criteria; (5) risk flags; and (6) metrics criteria/recommendations. Enterprise Architecture should also develop internal and external governance means/metrics/guidelines and socialize these with other teams.

With reference to the implementation of the architecture roadmap, typically, there will be an engineering/development organization that develops/builds systems and an operations organization that runs the IT systems and manages the data center and network. Enterprise Architecture should assist these other groups develop service-specific specifications, service acceptance specifications, service-reporting specifications, and request for proposal (RFP) generation. However, Enterprise Architecture should not be as involved with the development of budgets and implementation timetables. Enterprise Architecture may provide consultative assistance for RFP generation/vendor selection. The group also provides standards governance (products) and architecture governance (architecture).
Typically, there will be a Supply-Chain Management (purchasing) group at the firm: Enterprise Architecture may provide consultative assistance for RFP generation/vendor selection (done in conjunction with the Engineering/Development and Operations groups.)

Figure 2.6 depicts the four layer of the architecture partitions used in this book (Business Architecture, Information Architecture, Systems/Solution Architecture, and Technology Architecture) along with artifacts generated by the Enterprise Architecture group.

In cases where ASP suppliers or asset-retention suppliers have been engaged by the firm, Enterprise Architecture should help ensure that the high-level architecture plans, service definition guidelines, service acceptance principles, service integrity criteria, SLA criteria, risk flags, and metrics criteria/recommendations are properly addressed and implemented. This is done to mitigate the risks identified earlier, specifically, the goal of maintaining vendor independence/portability (see the left side of Figure 2.6).

Here, Enterprise Architecture should be able to cite demonstrated savings to the firm in using a recommended Architecture Approach A versus Architecture Approach B or no architecture at all. Also, it should be able to document improved flexibility/time-to-market/service availability in using recommended Architecture Approach A versus Approach B or no architecture at all.

Sometimes the following question is posed, “Why do I need enterprise architecture when I can get what I may need via an SLA with an outsourcer?” The answer to this question includes the following observations.

Typically, firms have outsourced the more routine, back-office, operations-oriented functions. They have not outsourced the more strategic-planning-oriented functions that support

![Figure 2.6 Functions and responsibilities.](image-url)
the aforementioned activities. These more strategic functions are properly synthesized through an architectural capability that has the long-term technical integrity of the corporation in mind. SLAs do not address this issue; SLAs relate to a current capability; architecture work relates to future capabilities/positioning.

Architecture provides the “high-altitude/strategic view” of where the corporation should want to go. Outsourcers require proactive management to make sure that they do not deploy a set of technologies (on behalf of the enterprise) that may be constraining, suboptimal, short-sighted, self-serving, risky, premature, untested, etc. Many outsourcing deals have not delivered the intended value; hence, proactive management is important. This is done by specifying high-level technical/architecture directions and making sure they follow suit. Furthermore, the firm’s target state is best developed by personnel who are intimately familiar with the firm’s business and IT environment.

Besides the strategic function, Enterprise Architecture can assist the development of the “correct” set of SLAs. Specifically, the Enterprise Architecture group will work on the type, form, syntax, parameterization, and interdependence of SLA. For example, what would be the correct set of SLAs to guarantee that a host-based VoIP service has sufficient quality, reliability, security, etc.? Hence, the group will develop high-level architecture plans, service definition guidelines, service acceptance principles, service integrity criteria, SLA criteria, risk flags, and metrics criteria/recommendations. Enterprise Architecture also develops internal and external governance means/metrics/guidelines.

### 2.1.5 Organization-Specific Architecture Principles

#### 2.1.5.1 Overview

It may be of some value to develop a set of organization-specific architectural principles that can be used to develop specific subarchitectures. These principles are philosophical positions of the firm with regard to particular issues of specific importance to a given organization. Architectural principles represent fundamental perspectives and practices believed to be valuable for the organization. Architectural principles can serve to provide a basis for decision making in regard to potential selections of attributes of the architecture (as we discussed in the appendix of Chapter 1, the attributes of the architecture $A(n)$ represent the set of function groupings, the constituent functions, the interfaces, and the protocols utilized over the interfaces). In other words, they form the basis for making IT decisions related to the future (state). After defining and documenting the principles, an effort must be undertaken to define policies and procedures to support the proper application of the principles when an architecture or subarchitecture is developed.

Principles are general rules and guidelines that are stable and rarely amended. Principles describe and support the way that an organization undertakes its mission. Architectural principles must enjoy a level of consensus among the various elements of the enterprise. Each architectural principle needs to relate to the business objectives of the firm. However, principles are not expected to exhaustively define in an algorithmic manner the processes employed by the organization to carry out its work, but rather, they are part of a structured set of concepts, approaches, and philosophical ideas that collectively point the organization toward the goal.

Firm-specific principles are often defined in hierarchal tiers (or layers), although this is not mandatory. Suppose principles are organized as Tier 1 (e.g., Enterprise), Tier 2 (e.g., IT), and Tier 3 (Enterprise Architecture). Then, Tier 2 (e.g., IT) principles will be guided by, and elaborate on, the principles at the Tier 1 (e.g., Enterprise). Tier 3 (Enterprise Architecture) principles will be
guided by the principles at the two higher levels. For example, a firm can have the following tier arrangement [TOG200501]:

- **Enterprise (business) principles** provide guidance for decision making throughout an enterprise, and impact how the organization sets about fulfilling its mission. **Enterprise principles** are used as a means of harmonizing decision making across a distributed organization. Included in this category are the business desiderata of the enterprise, e.g., business charter, business goals, strategic business directions, and some degree of service-delivery style/approach. Such enterprise-level principles are commonly found in governmental and not-for-profit organizations, but are also used in commercial organizations. In particular, they are a key element in a successful governance strategy related to architectural adherence by the firm. Generally, the definitional process for the generation and institutionalization of business principles is outside the scope of the architecture function.

- **IT principles** provide guidance on the use and deployment of all IT resources and assets across the enterprise. They are developed with the goal of making the information environment as productive and cost-effective as possible.

- **Architecture principles** provide guidance that relates to architecture efforts. These principles reflect a level of consensus across the enterprise, and embody the spirit and thinking of the enterprise architecture. The set of architecture principles focus on architecture-level issues but they can to also restate, or cross-refer, or embody a portion of the set of business principles. In any case, they must support the business principles; i.e., within an architecture project, the architect will normally need to ensure that the definitions of these business principles, goals, and strategic drivers are current, and clarify any areas of ambiguity. Architecture principles can be further partitioned as follows:
  - **Principles that govern the architecture process**, affecting the development, maintenance, and use of the enterprise architecture
  - **Principles that govern the implementation of the architecture**, establishing the first tenets and related guidance for designing and developing information systems.

Architectural principles are generally developed by the chief architect, in collaboration with the enterprise CIO, architecture board, and business stakeholders. Having too many principles can negatively impact the flexibility of the architecture development and maintenance process. Organizations should, therefore, define only high-level principles and limit the number to a dozen or at most two dozen. In addition to a definition statement, each principle could include information related to the rationale for choosing such a principle and also observations related to the possible downstream implications of the principle. This ancillary information serves to facilitate the acceptance of the principle and to support its application by explaining and justifying the rationale behind the guidelines embodied in the principle.

Two important observations here are that (1) principles are, basically, philosophical ideals/positions of the firm (specifically, of the firm’s IT positions), and (2) the firm should not spend an inordinate amount of time to develop these. Certainly, it should not take three years and ten FTE (full time equivalents) to develop these. For example, it took this author one day to develop the following 16 enterprise architecture principles used in some applications (only the statement portion of the principle is shown):
1. The enterprise architecture shall be **business-enabling**. It should help the company conduct its business in a reliable, expansive manner.

2. The enterprise architecture shall be **value-enhancing**. It should help the company achieve increased value, either through improved cost, new functionality, or new flexibility, etc.

3. The enterprise architecture shall be **brand-name advancing**. Clearly, it should help the company achieve added customer satisfaction and brand recognition.

4. The enterprise architecture shall be **market-share expanding**. It should help the company open up new markets and reach a market faster in a more comprehensive manner.

5. The enterprise architecture shall be **time-to-market facilitating**. It should help the company introduce new products faster, respond to changing consumer needs faster, etc.

6. The enterprise architecture shall be **productivity-enhancing**. It should help the company conduct its business with fewer or less expensive resources.

7. The enterprise architecture shall be **sourcing-enabling**. It should help the company rapidly and flexibly bring up a new outsourcing provider.

8. The enterprise architecture shall be **process-simplifying**. It should help the company conduct its business in a simpler, more direct fashion.

9. The enterprise architecture shall be **cost-effective**. It should help the company support transactions at a lower cost per transaction (for the same or better level of functionality).

10. The enterprise architecture shall be **optimized**. It should help the company avail itself of ever-decreasing costs in technology to its benefit. It does so in a way that is the best compared to other approaches.

11. The enterprise architecture shall be **global**. It should help the company reach/connect providers on multiple continents.

12. The enterprise architecture shall be **multi-service**. It is often cheaper to have, say, one network that supports voice, video, and data, from a facilities, management, and planning/engineering point of view.

13. The enterprise architecture shall be **scalable**. It should allow the company to rapidly add specific functionality in a forward-compatible, cost-effective, and rapid manner.

14. The enterprise architecture shall offer multiple levels of **quality of service (QoS)**. It should help the company provide different grades of service to different transactions, products, or clientele.

15. The enterprise architecture shall be **reconfigurable**. It should help the company quickly (and cost-effectively) redesign/reprovision the topology or relationship matrix to support rapidly evolving business needs.

16. The enterprise architecture shall be **unified**. It should support a cohesive, consistent, well-architected, and well-planned networking infrastructure.

This is just an example. There are no fixed rules that limit the philosophical principles; they generally depend on the firm’s mandate. For example, for a public agency the principles could relate to enhancing public safety. For a nonprofit the principles could relate to maximizing its value to the community. For a medical-oriented organization the principles could relate to optimizing healthcare delivery. And so on. Again, these principles are nothing more than are philosophical positions of the firm and are to be used simply as a catalyst for architectural development, and the firm should not spend an inordinate amount of time to develop them. Also, the principle should neither be totally abstract or unrealistically ethereal nor ultraspecific or ultratechnical (see Figure 2.7).
It is useful to have a systematic approach to defining principles. Using computer science terminology, one may define a principle, if the drafter of the principles so chooses, as an object with the following body parts:

\[
\text{PR}_x: = \text{Name}, \text{Statement\_body-part}, \text{Rationale\_body-part}, \text{Implications\_body-part}
\]

where [TOG200501]:

- **Name**: Identity information (that distinguishes it from other principles/objects) that represents the essence of the rule. Additional observations: The name should be easy to remember. Specific technology platforms should not be mentioned in the name or statement of a principle. One should not use ambiguous words in the Name (and in the Statement\_body-part) such as *support* and *open*. Avoid using unnecessary adjectives and adverbs.

- **Statement\_body-part**: Description that succinctly and unambiguously communicates the fundamental rule. For the most part, the statements for managing information are similar from one organization to the next. It is important that the statement be unambiguous.

- **Rationale\_body-part**: Description that highlights the business benefits of adhering to the principle, using business terminology. The drafter of the principle should point to the similarity of information and technology principles to the principles governing business operations. Also, the drafter of the principle should describe how it is related to other principles, and guidelines for a balanced interpretation. One should describe situations where one principle would be given precedence or carry more weight than another for making a decision.
Implications: Description that highlights the requirements, both for the business and IT, for implementing the principle—in terms of resources, costs, and activities/tasks. It will often be apparent that current systems, standards, or practices would be incongruent with the principle upon adoption. The impact to the business and consequences of adopting a principle should be clearly stated. The reader of the principle should readily discern the answer to “How does this affect me?” It is important not to oversimplify, trivialize, or judge the merit of the impact. Some of the implications will be identified as potential impacts only, and may be speculative rather than fully analyzed.

On the other hand, it is not obligatory that architectural principles exist at a firm; in this case the decisions related to attributes can be made based on the basis of some other mechanism (e.g., specific information about a particular subarea, specific financial business case as applicable to the subarea in question, specific information about a technology or solution, etc.) The fact that there is an absence of principle does not have to be taken as “an alarm bell”—it may simply be that given the relatively abstract formulation of the principles, the firm decides it is not worth the effort to define, maintain, and apply them to the daily decision-making process.

2.1.5.2 Additional Details

Architectural principles are intended to capture the fundamental truths about how the enterprise uses and deploys information technology resources and assets. Some useful guidance is provided in the TOGAF documentation about principles, which is applicable independently of the architectural framework model that is chosen by the enterprise (because principles are exogenous to the framework itself.) We include such guidance in this section [TOG200501].

Architectural principles need to be driven by overall IT principles and principles at the enterprise level, if they exist. Architectural principles are chosen to ensure alignment of IT strategies with business strategies and visions. Principles are often interrelated and must be developed and utilized as a cohesive set. Specifically, the development of architectural principles is typically influenced by the following:

- Enterprise mission and plans: The mission, plans, and organizational infrastructure of the enterprise
- Enterprise strategic initiatives: The characteristics of the enterprise—its strengths, weaknesses, opportunities, and threats—and its current enterprisewide initiatives (such as process improvement, quality management)
- External constraints: Market factors (time-to-market imperatives, customer expectations, etc.); existing and potential legislation
- Current systems and technology: The set of information resources deployed within the enterprise, including systems documentation, equipment inventories, network configuration diagrams, policies, and procedures
- Computer industry trends: Predictions about the usage, availability, and cost of computer and communication technologies, referenced from credible sources along with associated best practices presently in use
Simply having a set of written statements labeled as principles does not mean that the principles are good, even if people in the organization agree with it. A good set of principles is founded on the beliefs and values of the organization and expressed in language that the business side of the company understands and uses. This is driven by the fact that IT is a large line item in the budget, as high as 10% (as discussed in Chapter 1); therefore, it is important that the business side understand what they are getting for the money they allocate. Unfortunately, all too often conversation about IT effort is replete with an intolerably thick acronym alphabet soup.

Architectural principles (or even more generally, enterprise principles) need to drive behavior within the IT organization at the firm. Principles should be few in number, be forward-looking, and be endorsed and championed by senior IT and business management. They provide a foundation for making architecture and planning decisions, framing policies, procedures, and standards, and supporting resolution of contradictory situations. A poor set of principles will quickly fall into disuse, and the resultant architectures, policies, and standards will appear arbitrary or self-serving, and thus lack credibility. There are several criteria that characterize a good set of principles:

- **Understandability**: The underlying tenets can be quickly grasped and understood by individuals throughout the organization. The intention of the principle is clear and unambiguous, so that violations, whether intentional or not, are minimized.

- **Robustness**: Enables good-quality decisions about architectures and plans to be made, and enforceable policies and standards to be created. Each principle should be sufficiently definitive and precise to support consistent decision making in complex, potentially controversial situations.

- **Completeness**: Every potentially important principle governing the management of information and technology for the organization is defined. The principles cover every perceived situation.

- **Consistency**: Strict adherence to one principle may require a loose interpretation of another principle. The set of principles must be expressed in such a way that a balanced interpretation is possible. Principles should not be contradictory to the point where adhering to one principle would violate the spirit of another. Every word in a principle statement should be carefully chosen to allow consistent yet flexible interpretation.

- **Stability**: Principles should be enduring, yet able to accommodate changes. An amendment process should be established for adding, removing, or altering principles after they are ratified initially.

The architectural principles are used in a number of ways:

1. To provide a baseline within which the enterprise can start to make conscious decisions about IT.
2. As a guide to establishing relevant evaluation criteria, thus exerting strong influence on the selection of products or product architectures in the later stages of managing compliance with the IT architecture.
3. As drivers for defining the functional requirements of the architecture.
4. As an input to assessing both existing IT systems and the future strategic portfolio, for compliance with the defined architectures. These assessments will provide valuable insights into the transition activities needed to implement an architecture, in support of business goals and priorities.
5. The Rationale statements (discussed earlier) highlight the value of the architecture to the enterprise, and therefore provide a basis for justifying architectural activities.

6. The Implications statements (discussed earlier) provide an outline of the key tasks, resources and potential costs to the enterprise of following the principle. They also provide valuable inputs to future transition initiative and planning activities.

7. Support the architectural governance activities in terms of:
   - Providing a “back-stop” for the standard compliance assessments when some interpretation is allowed or required
   - Supporting the decision to initiate a dispensation request when the implications of a particular architecture amendment cannot be resolved within local operating procedures

From a pragmatic perspective, however, we strongly believe that just having a “nice set” of ideal target statements does not lead anywhere, unless the organization is willing to put “its money where the mouth is” and also runs the architecture function as a mandated compliance operation rather than just a “debate society to determine who has the best advocacy oratory and salesmanship skills.” Architects in a firm need not have to be the best orators, advocacy mouthpieces, or salespeople; although these skills help, if 95% of the architects’ time is “selling” and 5% is in the technical work, then such a function is really dysfunctional. We believe that the effort ratio allocation between “selling” and “doing the technical work” in an architecture organization needs to be more like “20% selling, 80% technical work.” Therefore, the reader needs to be careful not to be oversold on the idea of generating an aesthetically beautiful set of principles because by itself this is only an academic exercise.

Appendix 2.1 shows a set of principles discussed under the TOGAF Model. Another example of architecture principles is contained in the U.S. government’s Federal Enterprise Architecture Framework.

### 2.1.6 Instituting Enterprise Architecture Mechanism in a Firm

In consideration of the advantages to be gained by having an architected approach to IT, especially considering the relatively high budget figures involved in running an IT operation, as discussed in Chapter 1, the CIO may seek to establish an enterprise architecture capability within the firm. Generally, only a few senior individuals who have had previous experience in architecture are needed: a network architect, a platform architect, a security architect, a data architect, an integration architect, etc.

In recent years many firms have set out to install an enterprise architecture function. Some have attempted a centralized function with stringent governance powers. Others have tried a federated structure of architects from the different business units, working together to share best practices and coordinate architecture decisions across the firm. Yet others have tried a hybrid approach. Some argue that the hybrid approach is best [KOC200502]. In any event, because the IT department seems to be chronically tackling backlogs and is unable to meet requests from the business side in a timely manner (part of the reason being, perhaps, that IT often seeks to reinvent everything to perhaps get 95% of the users’ needs over a long development cycle rather than using a commercial off-the-shelf (COTS) and getting 85% of the users’ needs over a short development cycle), the Enterprise Architecture group should not be perceived as adding any (additional) delay or obstructing the IT initiatives.
Besides the dos and don’ts that we discussed in Chapter 1, what is needed is a relatively quick and effective exercise to achieve the following:

- Develop and establish a set of organization-specific architectural principles (guidelines) (say, 0.5 staff-years of effort).
- Select a framework from the set of available industry-defined frameworks (say, 1.0 staff-years of effort).
- Identify the IT areas that are subject to architectural standardization (this is the partitioning/Functional Block discussion alluded to earlier, but it is also driven by the framework chosen—perhaps a dozen or so areas are identified) (say, 0.5 staff-years of effort).
- Identify applicable industry standards (say, 0.5 staff-years of effort).
- Select architectural description tools and install them (say, 1.0 staff-years of effort).
- Establish governance mechanisms (and tools) and empower an Architecture Review Committee (say, 0.5 staff-years of effort).
- Start the process of identifying in the current environment for the various areas identified earlier, but do not spend an inordinate amount of time documenting the “as is” environment (say, 2.4 staff-years for dozen or so areas identified).
- Start the process of identifying the target environment for the various areas identified earlier, and publish target architectures (the “to be” environment) and architectural standards in such a manner that conformance by the developed is self-directed (namely, the architectures and standards contain unambiguous PICS Proforma) (say, 6 staff-years for dozen or so areas identified).

Notice that the effort discussed so far can be undertaken in about 13 staff-years of effort. Market research at the time of this writing found that nine out of ten companies surveyed had centralized enterprise architecture groups of fewer than ten people, regardless of company size [KOC200502]. This means that for the typical level of investment a firm can get the function established and running in one calendar year.

The refresh activities should be accomplishable for about 0.25–0.5 of a staff-year per architecture subarea. That implies that for a dozen or so areas identified, it will take about three to six staff-years. In turn this means that an organization of about ten people should be able to support this function. Large organizations have IT staffs that typically number in the several hundreds upward to a couple of thousand; ten people within such a pool should be financially acceptable in terms of the required investment.

Sometimes firms rely on consultative help from the outside to undertake the tasks of setting up an enterprise architecture initiative (from organizations such as Gartner, IBM, Forrester/Giga Research, Delloite & Touche, Accenture, Capgemini); what is important is that the function be established.

2.2 Enterprise Architecture Constructs

This section starts the discussion of the available industry machinery that can be used by firms to document their enterprise architecture work. We focus on architectural frameworks. A framework is intended to be a “language” to enable communication, research, and implementation of enterprise architecture constructs. Although some selection effort is needed, particularly by developing requirements for the framework that take into account the ultimate goal of the enterprise
architecture function, the firm should not devote an inordinate amount of time to selecting the framework. Earlier we stated that a firm should try to select a framework from the set of available industry-defined frameworks with about 1.0 staff-years of effort. We noted in Chapter 1 that enterprise architecture must be more than just pretty pictures: firms need more than elaborate figures, charts, and presentations; unless the concepts are translated into actual decisions, migrations, and governance, such intriguing pictorials will not lead to any concrete results. Also, it needs to be reiterated that the architectural framework (just by) itself does not intrinsically save money for a firm: all of the follow-on artifacts must be developed, and then in turn applied to the environment, that is to say, implemented through funded efforts.

Leading organizations are seeking to organize architecture information using industry-proven frameworks and tools. This reduces replication of architecture documentation and improves the accessibility to architecture documents (these being captured under a system of record). Additionally, the use of tools reduces the cycle time for creating new artifacts, while at the same time establishing common nomenclature to foster consistency. At the same time, recent surveys have shown that a nontrivial fraction of organizations (about one-fifth) are defining their own enterprise architecture frameworks instead of adopting existing frameworks, and the trend seemed to be accelerating. The use of enterprise architecture repository tools is growing; however, again, at press time most organizations were still using Microsoft’s Office and Visio products for capturing their enterprise architecture. On the other hand, the usage of standard business modeling techniques for modeling the results is broadly accepted, and Business Process Modeling Language (BPML) is today the standard in this area. OMG’s Model-Driven Architecture and Unified Modeling Language (UML) are broadly accepted for modeling information systems [IEA200501].

### 2.2.1 Enterprise Architecture Principles

We discussed earlier that a firm may have firm-specific architectural principles that may depend on, among others, the industry the firm is in, the long-term mission of the firm, a short-term imperative of the firm, or a particular predicament the firm is in. In addition to the firm-specific architectural principles, the architecture framework may itself have (or may have been developed using) architectural principles in conjunction with subtending general business principles. Framework architectural principles provide a basis for decision making in regard to potential selections of attributes of the framework itself. These principles are not always explicitly stated in the frameworks, but they will have been used in the development and construction of the framework.

Framework architectural principles typically are similar to a core subset of firm-specific principles: e.g., “the framework shall be simple to use,” “the framework shall be unambiguous,” “the framework will facilitate (not hinder) the development of enterprise architectures for a wide range of enterprises,” “the framework shall be stable,” “the framework shall stand the test of time, and not be outdated within a couple of years as the underlying technology advances.” Again, these principles are not necessarily explicitly stated, but they will have been used to derive the framework itself.

### 2.2.2 Enterprise Architecture Frameworks

There are many enterprise architecture frameworks, and new ones are being added every day. Some frameworks are given in the following list:
Enterprise Architecture Goals, Roles, and Mechanisms

The Open Group Architecture Framework (TOGAF)

Zachman Enterprise Architecture Framework

Extended Enterprise Architecture Framework (E2AF)

Enterprise Architecture Planning (EAP)

Federal Enterprise Architecture Framework (FEAF)

Treasury Enterprise Architecture Framework (TEAF)

Capgemini’s Integrated Architecture Framework (IAF)

Joint Technical Architecture (JTA)

Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) and DoDAF

Department of Defense Technical Reference Model (DoD TRM)

Technical Architecture Framework for Information Management (TAFIM)

Computer Integrated Manufacturing Open System Architecture (CIMOSA)

Purdue Enterprise Reference Architecture (PERA)

Standards and Architecture for eGovernment Applications (SAGA)

European Union—IDABC & European Interoperability Framework

ISO/IEC 14252 (IEEE Std 1003.0)

IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description

Recent work shows that the most commonly used framework is the Zachman Framework, followed by the organization’s own frameworks, followed by TOGAF, U.S. DoD (this covers about two-thirds of all enterprises).* Generally, a framework is a detailed method and a set of supporting tools. Frameworks provide guidance on how to describe architectures; they typically do not provide guidance on how to construct or implement a specific architecture or how to develop and acquire systems or systems of systems [SYS200502].

Some see the architecture framework situation as a quagmire because of the overabundance of models and proposals [SYS200501]. Figure 2.8, loosely based on [SYS200501], depicts some of the relationships among the various models. In the material that follows we provide a brief overview of some of the more popular models.

2.2.2.1 The Open Group Architecture Framework (TOGAF 8.1)

The Open Group is a vendor-neutral and technology-neutral consortium seeking to enable access to integrated information, within and among enterprises, based on open standards and global interoperability. The Open Group had developed an architectural framework known as The Open Group Architecture Framework (TOGAF), Version 8.1 at time of book’s publication. It is described in a set of documentation published by The Open Group on its public Web server, and may be used freely by any organization wishing to develop an enterprise architecture for use within that organization.

TOGAF Version 8.1 partitions the architecture into the Business Architecture, Data Architecture, Application Architecture, and Technical Architecture. These four domains track directly with the four domains that we are using in this text.

TOGAF was developed by The Open Group’s own members. The original development of TOGAF Version 1 in 1995 was based on the Technical Architecture Framework for Information

*A press time survey showed the following statistics [IEA200501]: Zachman Framework: 25%; Organization’s own: 22%; TOGAF: 11%; U.S., DoD Architecture Framework: 11%; E2AF: 9%; FEAF: 9%; IAF: 3%; TAFIM: 2%; TEAF: 0%; ISO/IEC 14252 (IEEE Std 1003.0): 0%; Other: 9%.
The Open Group gave explicit permission and encouragement to create TOGAF by building on the TAFIM, which itself was the result of many years of development effort and many millions of dollars of U.S. government investment. Starting from this foundation, the members of The Open Group’s Architecture Forum have developed successive versions of TOGAF each year and published each one on The Open Group’s public Web site.

There are four main parts to the TOGAF document:

- **Part I: Introduction** provides a high-level introduction to some of the key concepts behind enterprise architecture and, in particular, the TOGAF approach.
- **Part II: Architecture Development Method** is the core of TOGAF. It describes the *TOGAF Architecture Development Method*—a step-by-step approach to developing an enterprise architecture.
- **Part III: Enterprise Continuum** describes the TOGAF Enterprise Continuum, a virtual repository of architecture assets, which includes the TOGAF Foundation Architecture, and the Integrated Information Infrastructure Reference Model.
- **Part IV: Resources** comprise the TOGAF Resource Base—a set of tools and techniques available for use in applying TOGAF and the TOGAF ADM.

TOGAF Version 8 is a superset of the well-established framework TOGAF Version 7. Version 8 uses the same underlying method for developing IT architectures that was evolved, with a particular focus on technical architectures, in the versions of TOGAF up to and including Version 7. However,
Version 8 applies that architecture development method to the other domains of an overall enterprise architecture—the business architecture, data architecture, and application architecture, as well as the technical architecture. The following significant additions have been made in Version 8.1:

1. Part II has a new section describing the requirements management process at the center of the ADM life cycle.
2. Part IV has a new structured section on architecture governance, comprising three subsections: (a) Introduction to Architecture Governance; (b) Architecture Governance Framework; and (c) Architecture Governance in Practice. Also, this part has a new section on architecture maturity models as well as a new section on TOGAF Architecture Skills Framework.

As noted earlier, TOGAF is the second most widely used framework. Chapter 3 covers this framework in detail.

2.2.2.2 Zachman Framework for Enterprise Architecture

The Zachman Framework for Enterprise Architecture (in short, the Zachman Framework) is a widely used approach for developing or documenting an enterprisewide architecture. John Zachman based his framework on practices in traditional architecture and engineering. The framework is a logical structure for classifying and organizing those elements of an enterprise that are significant to both the management of the enterprise and the development of its information systems. Figure 2.9 depicts the framework pictorially. The vertical axis provides multiple perspectives of the overall architecture, and on the horizontal axis a classification of the various artifacts of the architecture. Similar to other frameworks, its purpose is to provide a basic structure that supports the organization, access, integration, interpretation, development, management, and transformation of a set of architectural representations of the organization’s information systems; such objects or descriptions of architectural representations are usually referred to as artifacts. The Zachman Framework is the most widely used framework today.

In 1987, John Zachman wrote, “To keep the business from disintegrating, the concept of information systems architecture is becoming less of an option and more of a necessity.” From that assertion over 20 years ago, the Zachman Framework has evolved and become the model through which major organizations view and communicate their enterprise information infrastructure. The Zachman Framework draws upon the discipline of classical architecture to establish a common vocabulary and set of perspectives, a framework, for defining and describing today’s complex enterprise systems. Enterprise architecture provides the blueprint, or architecture, for the organization’s information infrastructure [ZIF200501].

The framework contains global plans as well as technical details, lists and charts as well as natural language statements. Any appropriate approach, standard, role, method, technique, or tool may be placed in it. In fact, the framework can be viewed as a tool to organize any form of metadata for the enterprise. The contents of some of the cells are well understood in the industry today. In fact, it is easy to purchase, off the shelf, application development tools and methodologies that support building the model [ZIF200501].

Chapter 3 covers this framework in detail.
Figure 2.9 The Zachman Framework.
2.2.2.3 Extended Enterprise Architecture Framework (E2AF)

The Extended Enterprise Architecture℠ (E2A) and Extended Enterprise Architecture Framework (E2AF) are developed by The Institute For Enterprise Architecture Developments (IFEAD). E2AF was in version 1.4 (at press time). E2AF addresses three major elements in a holistic way: the element of construction, the element of function, and the element of style. Style reflects the culture, values, norms, and principles of an organization.

Often, the term enterprise architecture deals with construction and function, without due consideration of the stylistic aspect. The stylistic aspect reflects the cultural behavior, values, norms, and principles of an organization in such a way that it reflects its corporate values. At the same time, the enterprise architecture addresses the aspects of business, information, information systems, and technology infrastructure in a holistic way covering the organization and its environment [IEA200501]. These domains are similar to the ones used in this text (see Figure 2.10). In this context enterprise architecture relates to understanding all of the different elements that go to make up the enterprise and how those elements interrelate; this is basically the definition we provide in Appendix 1.1.

Organizations can use this framework for their own purposes or in conjunction with their customers, by including a reference notice to IFEAD’s copyrights. Organizations that want to use the framework for commercial purposes can get a license from IFEAD (http://www.enterprise-architecture.info/).

E2AF is based on the ideas described in IEEE 1471-2000 regarding views and viewpoints, and a transformation of these concepts into the enterprise architecture domain enables another perspective of viewpoints and views. Looking from the outside world at an enterprise, several groups of (extended) enterprise stakeholders influence the goals, objectives, and behavior of the enterprise. Even so, these groups of enterprise stakeholders have different concerns, and therefore different sets of viewpoints, when we analyze these extended enterprise stakeholders. The model clusters their concerns into four generic categories (Business, Information, Systems, and Infrastructure Technology), also as seen in Figure 2.10. The framework shows the drivers of the enterprise and makes possible an understanding of what motivates the (extended) enterprise stakeholders [IEA200501].

2.2.2.4 Department of Defense Architecture Framework (DoDAF)

In the mid-1990s, in response to requirements related to joint multi-service and multinational military operations, the DoD discerned the need for a standard architectural formulation to ensure that its military systems could interoperate. The Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework, Version 1.0, was published in 1996 and it was reissued a year later in Version 2.0. On the basis of the experience with these frameworks, the DoD began work on a new version and published the DoD Architecture Framework (DoDAF), Version 1.0, in 2003.

The goal of DoDAF is to ascertain that the architectural descriptions developed by the various commands, services, and agencies are compatible and interrelatable and that the technical architecture views are usable and integrable across organizational domains. This framework addresses the military domain and is used primarily by the DoD.

Similar to any other architecture framework, it provides rules and guidance for developing and presenting architecture descriptions, including artifacts. It provides input on how to describe architectures, but it does not provide mechanisms in how to construct or implement a specific
Extended Enterprise Architecture Framework (E2AF)

IFEAD is an independent research and information exchange organization working on the future state of Enterprise Architecture.

**Figure 2.10** Extended Enterprise Architecture Framework (E2AF).
architecture or how to develop and acquire systems or systems of systems. DoDAF is organized into three parts. Volume I provides general guidance on the need for and use of architecture descriptions in DoD. Volume II provides definitions of the 26 products contained in the three views of the model (Operational View, Systems View, and Technical Standards View). Volume III is a deskbook that provides examples of architectures that are compliant, approaches to undertaking architecture development, and other support information. To comply with the framework, architecture descriptions must include the appropriate set of products and use the common terms and definitions specified in the framework.

As noted, DoDAF’s architectural descriptions require the use of multiple views, each of which conveys different aspects of the architecture in several products (artifacts or models). DoDAF’s integrated architecture comprises a number of views and the interrelationships between them. DoDAF defines the following views [SYS200502,DOD200301] (see Figure 2.11 [FEA200503]):

- **Operational View** depicts what is going on in the real world that is to be supported or enabled by systems represented in the architecture. Activities performed as parts of DoD missions and the associated information exchanges among personnel or organizations are the primary items modeled in operational views. The operational view reveals requirements for capabilities and interoperability.

- **Systems View** describes existing and future systems and physical interconnections that support the DoD needs documented in the operational view.

- **Technical Standards View** catalogs standard (commercial off-the-shelf [COTS], government off-the-shelf [GOTS]) system parts or components and their interconnections. This view augments the systems view with technical detail and forecasts of standard technology evolution.

- **All View** augments the other views by providing context, summary, or overview-level information, and an integrated dictionary to define terms.

The DoDAF comprises the types of guidance listed in Table 2.3.

The **Core Architecture Data Model** (CADM) (originally developed in the context of C4ISR) is a formal model of architecture products (artifacts), structures, and their respective interrelationships. It provides a common schema (database) for repositories of architectural information. A

---

**Figure 2.11** Views in DoDAF.
A repository based on the CADM is able to store in a common manner architecture products (artifacts) from multiple framework-based architecture projects. This allows products from different projects, organizations, or services to be analyzed and compared. Furthermore, with a CADM-based repository, the extraction and presentation of an appropriate subset of the architectural information can be (partially) automated.

The DoDAF defines (Volume II) 26 different architecture products that are organized per the views (All, Operational, Systems, and Technical standards), as follows [SYS200502]:

- **All View**
  - **Overview and Summary Information**: Details scope, purpose, environment, and other summary-level formation for an architectural description.
  - **Integrated Dictionary**: Provides definitions of all terms used in all products.

- **Operational View**
  - **Operational Concept Graphic**: Provides a graphical and textual description of the operational concept.
  - **Operational Node Connectivity Description**: Lists the operational nodes, activities performed at each node, connectivity, and information needs between nodes.
  - **Operational Information Exchange Matrix**: Lists and describes information exchanged between nodes.
  - **Organizational Relationships Chart**: Lists organizations, roles, and relationships among organizations.
  - **Operational Activity Model**: Details the activities performed and their interrelationships, including input/output relationships.
  - **Operational Rules Model**: Identifies business rules that govern or constrain operations.
  - **Operational State Transition**: Identifies sequencing and timing of activities.
  - **Operational Event Trace**: Traces actions in a scenario or sequence of events.
  - **Logical Data Model**: Identifies data requirements of the operational view.

- **Systems View**
  - **Systems Interface Description**: Lists systems, system components, and their interconnections.
  - **Systems Communications Description**: Identifies system communications.
  - **Systems–Systems Matrix**: Lists connections between individual systems in a group.
  - **Systems Functionality Description**: Lists functions performed by individual systems and the related information flow.
- **Operational Activity-to-Systems Function Traceability Matrix**: Maps systems information back to the operational view.
- **Systems Data Exchange Matrix**: Provides detail of data moving between systems.
- **Systems Performance Parameters Matrix**: Lists performance characteristics of individual systems.
- **Systems Evolution Description**: Lists migration plans for systems.
- **Systems Technology Forecast**: Lists technologies and products that are expected to affect systems.
- **Systems Rules Model**: Describes constraints on system operation imposed by design or implementation.
- **Systems State Transition Description**: Describes system activity sequencing and timing.
- **Systems Event Trace Description**: Describes system-specific requirements on critical event sequences.
- **Physical Schema**: Describes the physical implementation of the logical data model from the operational view.

### Technical Standards View
- **Technical Standards Profile**: Lists technical standards that apply to the architecture.
- **Technical Standards Forecast**: Describes emerging or evolving standards that might apply to the architecture.

#### 2.2.2.5 Enterprise Architecture Planning (EAP)

EAP defines a process that emphasizes techniques for organizing and directing enterprise architecture projects, obtaining stakeholder commitment, presenting the plan to stakeholders, and leading the organization through the transition from planning to implementation. EAP was originally published in the early 1990s by S. Spewak [SPE199201]. EAP takes a business-data-driven approach aimed at ensuring quality information systems. EAP emphasizes (1) the definition of a stable business model, (2) data dependencies defined before system implementation, and (3) the order of implementation activities based on the data dependencies. EAP is a specific attempt to provide methodologies for specifying the top two rows of the Zachman Framework: Scope (Planner) and Business Model (Owner). Consequently, only the BPM is pursued, and no effort is devoted to technical design or implementation. EAP has been primarily used in business and industrial market segments.

The major principles that guide the application of the EAP framework include the following [SYS200502,SPE199201]:

- Enterprise data should be accessible whenever and wherever it is needed.
- Information systems should adapt to meet the needs of changing business needs.
- High data integrity and standards should exist across the enterprise.
- All enterprise data systems should be integrated.
- These critical success factors should be obtainable cost-effectively.

The synthesis of EAP is shown in Figure 2.12 [SYS200502]. Each block represents a phase of the process that focuses on how to define the associated architectures and development plans. The framework defines the roles and responsibilities for each phase along with descriptions of the artifacts that
need to be produced. Additional capabilities include mechanisms for carrying out the process steps and for undertaking cost estimates for each phase.

2.2.2.6 **Federal Enterprise Architecture (FEA)**

The goal of FEA is to improve interoperability within U.S. government agencies by creating one enterprise architecture for the entire federal government. The Clinger–Cohen Act (1996) legislated a chief information officer (CIO) for all federal agencies and makes him or her responsible for developing, maintaining, and facilitating the implementation of sound and integrated information technology and/or systems. Additional legislation that followed (Executive Order 13011 and the E-Government Act of 2002) established the Chief Information Officers Council as the principal interagency entity to improve agency practices for the management of IT. As part of its work the CIO Council authored the FEAF with the publication of “Federal Enterprise Architecture Framework, Version 1.1” in 1999 and “A Practical Guide to Federal Enterprise Architecture, Version 1.1” in 2001. The Office of Management and Budget (OMB) now reviews and approves the business cases for all major IT investments. The mandated applicability of FEA covers all organizations in the federal government.

Architectural principles include the following [FEA200101, SYS200502]:

- Architectures must be appropriately scoped, planned, and defined based on the intended use.
- Architectures must be compliant with the law as expressed in legislative mandates, executive orders, federal regulations, and other federal guidelines.
- Architectures should facilitate change.
- Architectures set the interoperability standard.
- Architectures provide access to information but must secure the organization against unauthorized access.
- Architectures must comply with the Privacy Act of 1974.
- Enterprise architectures must reflect the agency’s strategic plan.
- Enterprise architectures coordinate technical investments and encourage the selection of proven technologies.
Architectures continuously change and require transition.
Architectures provide standardized business processes and common operating environments.
Architecture products are only as good as the data collected from subject matter experts and domain owners.
Architectures minimize the burden of data collection, streamline data storage, and enhance data access.
Target architectures should be used to control the growth of technical diversity.

The Federal Enterprise Architecture is a collection of interrelated reference models designed to facilitate cross-agency analysis. It is intended for use in analyzing and optimizing IT operations (see Figure 2.13). The objectives of the FEAF are to enable the federal government and agencies to achieve the following [FEA200503]:

- Leverage technology and reduce redundant IT expenditures across the federal government
- Facilitate cross-agency IT integration and sharing of data
- Apply common architecture practices
- Assist agencies to meet their EA legislative mandates

The intent of the FEAF is to enable the federal government to define and align its business functions and supporting IT systems through a common set of reference models. These models are defined as follows [FEA200503]:

- Performance Reference Model (PRM): The PRM is a standardized framework to measure the performance of major IT investments and their contribution to program performance.
- Business Reference Model (BRM): The BRM is a function-driven framework for describing business operations of the federal government independent of the agencies that perform them.
- Service Component Reference Model (SRM): The SRM is a business- and performance-driven functional framework that classifies service components according to how they support business or performance objectives.

|-----------------------|--------------------------------|----------------------------------------|-------------------------------------------|-------------------------------|

Figure 2.13 The Federal Enterprise Architecture.
Data Reference Model (DRM): The DRM is a model describing, at an aggregate level, the data and information that support program and business line operations.

Technical Reference Model (TRM): The TRM is a component-driven, technical framework used to identify the standards, specifications, and technologies that support and enable the delivery of service components and capabilities.

The Performance Reference Model (see Figure 2.14) is a framework to measure the performance of IT investments. The PRM aims at producing enhanced performance information to improve strategic and daily decision making. Its purpose is to improve the alignment (and better articulate the contribution) of inputs with outputs and outcomes, thereby creating a clear “line of sight” to desired results. It identifies performance improvement opportunities that span traditional organizational structures and boundaries. Agencies were required to use the PRM in their FY2005 IT initiatives.

The Business Reference Model is a function-driven agency-independent framework for describing the business operations of the federal government. The BRM Version 2.0 provides an organized, hierarchical construct for describing the day-to-day business operations of the government. The BRM is the first layer of the Federal Enterprise Architecture, and it is the main reference point for the analysis of data, service components, and technology. The BRM lists four Business Areas that provide a high-level view of the operations the federal government performs. The four business areas comprise a total of 39 external and internal Lines of Business and 153 SubFunctions. Starting with the FY2005 budget, agencies are required to map their major IT investments to BRM business lines.

The Service Component Reference Model is a business- and performance-driven, functional framework that classifies Service Components from the perspective of how they support business

![Performance Reference Model](image.png)

**Figure 2.14** Performance Reference Model.
or performance objectives. It is intended to support the discovery of governmentwide business and application service components in IT investments and assets. The SRM is structured across horizontal and vertical service domains that, independent of the business functions, can provide a foundation to support the reuse of applications, application capabilities, components, and business services. The SRM defines “component” as a self-contained business process or service with predetermined functionality that may be exposed through a business technology or interface. There are four levels in SRM [FEA200101,SYS200502]:

- Federated: A set of cooperating system-level components that resolve the business need of multiple end users, frequently from different organizations.
- Business Component System: A set of cooperating business components assembled to deliver a solution to a business problem.
- Business Component: The implementation of an autonomous business concept or business process.
- Distributed Component: A software element that can be called at run-time with a clear interface and a separation between interface and implementation. The most detailed level of granularity defined in the SRM.

The Technical Reference Model is a component-driven, technical framework employed to identify the standards, specifications, and technologies that support and enable the delivery of service components and capabilities. The TRM unifies existing capabilities by providing a foundation to advance the reuse of technology and component services from a governmentwide perspective. TRM has four tiers:

- Service Area: a technical tier that supports the secure construction, exchange, and delivery of business or service components. Each service area groups the requirements of component-based architectures within the federal government into functional areas.
- Service Category: a sub-tier of the service area to classify lower levels of technologies, standards, and specifications with respect to the business or technology function they serve.
- Standard: hardware, software, or specifications that are widely used and accepted (de facto) or are sanctioned by a standards organization (de jure).
- Specification: a formal layout/blueprint/design of an application development model for developing distributed component-based architectures.

2.2.2.7 Federal Enterprise Architecture Framework (FEAF)

As seen in a previous subsection, FEA seeks integrate the separate architectures of the various federal agencies. To support this goal, the government needs a collaboration tool for collecting and storing common architecture information; FEAF is such a tool. FEAF partitions a given architecture into business, data, applications, and technology architectures. FEAF includes the first three columns of the Zachman Framework and the EAP. FEAF allows the government to organize federal information for the entire federal government; promote information sharing among federal organizations; help federal organizations develop their architectures; help Federal organizations quickly develop their IT investment processes; and serve customer and taxpayer needs better, faster, and more cost-effectively.
The major components of the FEAF are (also see Figure 2.15) [SYS200501, FEA200102]:

- **Architecture Drivers:** Represent external stimuli that cause the FEA to change.
- **Strategic Direction:** Ensures that changes are consistent with the overall federal direction.
- **Current Architecture:** Represents the current state of the enterprise. Full characterization may be significantly beyond its worth and maintenance.
- **Target Architecture:** Represents the target state for the enterprise within the context of the strategic direction.
- **Transitional Processes:** Apply the changes from the current architecture to the target architecture in compliance with the architecture standards, such as various decision-making or governance procedures, migration planning, budgeting, and configuration management and engineering change control.
- **Architectural Segments:** Focus on a subset or a smaller enterprise within the total federal enterprise.
- **Architectural Models:** Provide the documentation and the basis for managing and implementing changes in the federal enterprise.
- **Standards:** Include standards (some of which may be made mandatory), voluntary guidelines, and best practices, all of which focus on promoting interoperability.

2.2.2.8 *Treasury Enterprise Architecture Framework (TEAF)*

TEAF aims at guiding the planning and development of enterprise architectures in all bureaus and offices of the Treasury Department. TEAF (Version 1.0, July 2000) is an elaboration of an earlier (1997) Treasury model, TISAF; it also draws from FEAF. The purpose of the framework is to achieve the following:

- Provide guidance for Treasury Enterprise Architecture development and management
- Satisfy OMB and other federal requirements
Support Treasury bureaus and offices with the implementation of their architectures based on strategic planning.

Show the benefits of incorporating enterprise architecture disciplines and tools into normal business operations.

Provide a structure for producing an EA and managing EA assets.

The key principles in TEAF include [SYS200501,TEA200001]:

Compliance with applicable laws, orders, and regulations is required.

Business objectives must be defined before building IT solutions.

Total business value is the primary goal that drives IT decisions.

EA is an integral part of the Investment Management Process.

Architectural decisions shall maximize interoperability and reusability.

Standardization will be used to fulfill common requirements and provide common functions.

Collaboration among Treasury IT organizations will facilitate sharing the information, data, and infrastructure required by the business units.

COTS technology will be used, where appropriate, rather than customized or in-house solutions.

Information and infrastructure are vital assets that must be managed, controlled, and secured.

EA must be consistent with departmental guidance and strategic goals.

TEAF has three basic parts:

1. A definition of the framework
2. A set of activities that guide architecture planning and implementation
3. A set of guidelines that support strategic planning, EA management, EA implementation approach, and building a repository for EA products

The framework contains resources and work products that guide architecture development. The EA description must depict various perspectives of the Treasury from several different views. For instance, the Planner perspective must contain models that describe the enterprise functions, information, organization, and infrastructure from the perspective of the executives responsible for planning the work of the Treasury bureaus and offices. Similar models must be created for the perspectives of the Owner, Designer, and Builder. See Figure 2.16 for the reference model.

The activities within the EA development process include (1) defining an EA strategy; (2) defining an EA management process; (3) defining an EA approach; and (4) developing the EA repository. Although specific guidance is given for what should be in an EA, including strategy, work products, roles, and responsibilities, the TEAF leaves to each bureau the responsibility for choosing the how, when, and why. The TEAF provides guidance for the following: creating an enterprise architecture strategy, defining a road map for development, defining roles and responsibilities of participants, creating policies for configuration management, managing investments, creating an enterprise repository, and creating specific work products.
2.2.2.9 ISO/IEC 14252 (IEEE Std 1003.0)

This 1996 ISO standard, “Guide to the POSIX Open System Environment,” was the initial basis for a number of frameworks (as shown in Figure 2.8). For example, TOGAF was originally based on the U.S. DoD Technical Architecture Framework for Information Management (TAFIM), which was itself a development of ISO/IEC TR 14252. At a general level, ISO/IEC 14252, TAFIM and TOGAF have a similar reference model. ISO/IEC TR 14252, however, does not supply a diagram that provides a detailed partitioning of the application software, application platform, or external environment entities; the document does partition the application platform into a number of similar, though not identical, areas. ISO/IEC TR 14252 includes details related to service category definition and in the recommended family of standards and specifications. The ISO/IEC TR 14252 reference model implies no structure among the service categories, leaving that to individual system architectures and actual implementations.

2.3 Enterprise Architecture Governance

We have already hinted at the criticality of the governance process at the end of Chapter 1 and earlier in this chapter. This topic could well cover a chapter, and perhaps even a book. In the following text we simply scratch the surface.

Governance relates to the publication of the architecture (enterprise architecture description) of the current or target state, the enterprise standards set, the enterprise approved equipment list; and the roadmap along with (migration) strategies, and then having the mechanisms, the support, and the management commitment to effect enforcement or consistency with the enterprise architecture.

There are many ways of handling governance. One possible way is based on the federated approach described in FEA. The federal chief information officer (could be the firm’s CIO) has issued an executive policy to ensure that all federal agencies (could be the firm’s departments) apply minimal EA procedures. The policy states every federal agency (could be the firm’s departments) should include the following [SYS200501, FEA200101]:

Figure 2.16  Reference model, TEAF.
An Enforcement Policy defines the standards and process for determining the compliance of systems or projects to the FEAF (firm’s enterprise architecture) and for resolving the issues of noncompliance. The Enforcement Policy should answer the following questions:

- How and when will projects submit project plans to be reviewed for enterprise architecture compliance?
- Who will be responsible for compliance assessment or justification of waivers?
- How will compliance and noncompliance be documented and reported?
- How will outstanding issues of noncompliance be resolved or waivers be processed and approved?
- Who will be responsible for processing, authorizing, and reassessing waivers?
- What will be the content and format of waiver submissions?
- If a waiver is granted, how will projects achieve compliance in the future?
- What are the ramifications if a noncompliant project is not granted a waiver (e.g., funding or deployment restrictions)?

The processes and procedures should allow for exceptions. In many cases, existing systems in the operations and maintenance phase should be granted exceptions or waivers from the technical standards and constraints of the enterprise architecture. Alignment of some legacy systems with new standards could be unreasonably costly and introduce additional risk to the business users [SYS200501]. However, a chronic granting of exceptions may defeat the purpose of having an enterprise architecture. We believe that the granted exceptions need to be less than 10% of the total set of activities, in most instances; otherwise, the lack of standardization will continue to proliferate. Sometimes a developer could find that using a nonstandard solution could be more locally optimal (read, cheaper) than using the company standard. In those cases the funding mechanism must be addressed by the chief architect: a (financially accommodating) mechanism must exist to “assist” the developer make the correct decision. For example, assume that the company was UNIX based but a desire may exist to move to Linux. Assume that the project would cost $x$ under UNIX but $x + Δ$ for Linux, considering that the developers/programmers are less familiar with the technology (as it might be new in that firm in question); or, it could take then $y + Δ$ weeks to complete the project, compared with $y$ weeks under the old technology. Then a way must be found to subsidize these developers (so that they have the money or they are not penalized for “overspending”), for
the common good of the firm to move to the new, target-desired environment; the same would apply if the amount of time took longer.

The question may also arise if the Enforcement Policy should be managed at the firm’s level or at the departmental level. In companies where there are good centralized controls or where the company operates a headquarter-driven operation, then a firm-level Enforcement Policy can be institutionalized. In companies where there are not good centralized controls or where the company operates a distributed, franchised, loosely coupled operation, then a department-level Enforcement Policy may have to be used.

Appendix 2.1: Firm-Specific Architectural Principles

This appendix expands the discussion on architectural principles, and is based on TOGAF documentation, with permission [TOG200501]. It provides an example that illustrates both the typical content of a set of architecture principles, and the recommended format.

Business Principles

1. Principle: **Primacy of principles**
   Statement: These principles of information management apply to all organizations within the enterprise.
   Rationale: The only way one can provide a consistent and measurable level of quality information to decision makers is if all organizations abide by the principles.
   Implications:
   - Without this principle, exclusions, favoritism, and inconsistency would rapidly undermine the management of information.
   - Information management initiatives will not begin until they are examined for compliance with the principles.
   - A conflict with a principle will be resolved by changing the framework of the initiative.

2. Principle: **Maximize benefit to the enterprise**
   Statement: Information management decisions are made to provide maximum benefit to the enterprise as a whole.
   Rationale: This principle embodies “service above self.” Decisions made from an enterprise-wide perspective have greater long-term value than decisions made from any particular organizational perspective. Maximum return on investment requires information management decisions to adhere to enterprisewide drivers and priorities. No minority group will detract from the benefit of the whole. However, this principle will not prevent any minority group from getting its job done.
   Implications:
   - Achieving maximum enterprisewide benefit will require changes in the way one plans and manages information. Technology alone will not bring about this change.
   - Some organizations may have to concede their own preferences for the greater benefit of the entire enterprise.
   - Application development priorities must be established by the entire enterprise for the entire enterprise.
   - Application components should be shared across organizational boundaries.
Information management initiatives should be conducted in accordance with the enterprise plan. Individual organizations should pursue information management initiatives that conform to the blueprints and priorities established by the enterprise. The plan will be changed as needed.

As needs arise, priorities must be adjusted. A forum with comprehensive enterprise representation should make these decisions.

3. Principle: **Information management is everybody’s business**
   
   **Statement:** All organizations in the enterprise participate in information management decisions needed to accomplish business objectives.
   
   **Rationale:** Information users are the key stakeholders, or customers, in the application of technology to address a business need. To ensure that information management is aligned with the business, all organizations in the enterprise must be involved in all aspects of the information environment. The business experts from across the enterprise and the technical staff responsible for developing and sustaining the information environment need to come together as a team to jointly define the goals and objectives of information technology.
   
   **Implications:**
   
   To operate as a team, every stakeholder, or customer, will need to accept responsibility for developing the information environment.
   
   Commitment of resources will be required to implement this principle.

4. Principle: **Business continuity**
   
   **Statement:** Enterprise operations are maintained in spite of system interruptions.
   
   **Rationale:** As system operations become more pervasive, one becomes more dependent on them; therefore, one must consider the reliability of such systems throughout their design and use. Business premises throughout the enterprise must be provided the capability to continue their business functions regardless of external events. Hardware failure, natural disasters, and data corruption should not be allowed to disrupt or stop enterprise activities. The enterprise business functions must be capable of operating on alternative information delivery mechanisms.
   
   **Implications:**
   
   Dependency on shared system applications mandates that the risks of business interruption must be established in advance and managed. Management includes, but is not limited to, periodic reviews, testing for vulnerability and exposure, or designing mission-critical services to ensure business function continuity through redundant or alternative capabilities.
   
   Recoverability, redundancy, and maintainability should be addressed at the time of design.
   
   Applications must be assessed for criticality and impact on the enterprise mission, to determine what level of continuity is required and what corresponding recovery plan is necessary.

5. Principle: **Common-use applications**
   
   **Statement:** Development of applications used across the enterprise is preferred over the development of similar or duplicative applications that are only provided to a particular organization.
   
   **Rationale:** Duplicative capability is expensive and leads to proliferation of conflicting data.
Implications:
Organizations that depend on a capability which does not serve the entire enterprise must change over to the replacement enterprisewide capability. This will require establishment of and adherence to a policy requiring this.
Organizations will not be allowed to develop capabilities for their own use that are similar/duplicative of enterprisewide capabilities. In this way, expenditures of scarce resources to develop essentially the same capability in marginally different ways will be reduced.
Data and information used to support enterprise decision making will be standardized to a much greater extent than was previously done. This is because the smaller organizational capabilities that produced different data (which was not shared among other organizations) will be replaced by enterprisewide capabilities. The impetus for adding to the set of enterprisewide capabilities may well come from an organization making a convincing case for the value of the data/information previously produced by its organizational capability, but the resulting capability will become part of the enterprisewide system, and the data it produces will be shared across the enterprise.

6. Principle: **Compliance with law**
Statement: Enterprise information management processes comply with all relevant laws, policies, and regulations.
Rationale: Enterprise policy is to abide by laws, policies, and regulations. This will not preclude business process improvements that lead to changes in policies and regulations.
Implications:
The enterprise must be mindful to comply with laws, regulations, and external policies regarding the collection, retention, and management of data.
Education and access to the rules. Efficiency, need, and common sense are not the only drivers in architecture development. Changes in the law and changes in regulations may drive changes in the processes or applications.

7. Principle: **IT responsibility**
Statement: The IT organization is responsible for owning and implementing IT processes and infrastructure that enable solutions to meet user-defined requirements for functionality, service levels, cost, and delivery timing.
Rationale: Effectively align expectations with capabilities and costs so that all projects are cost-effective. Efficient and effective solutions have reasonable costs and clear benefits.
Implications:
A process must be created to prioritize projects
The IT function must define processes to manage business unit expectations
Data, application, and technology models must be created to enable integrated quality solutions and to maximize results.

8. Principle: **Protection of intellectual property**
Statement: The enterprise's intellectual property must be protected. This protection must be reflected in the IT architecture, implementation, and governance processes.
Rationale: A major part of an enterprise's intellectual property is hosted in the IT domain.
Implications:
Although protection of intellectual property assets is everybody's business, much of the actual protection is implemented in the IT domain. Even trust in non-IT processes can be managed by IT processes (e-mail, mandatory notes, etc.).
A Security Policy, governing human and IT actors, will be required that can substantially improve protection of intellectual property. This must be capable of both avoiding compromises and reducing liabilities.

Resources on such policies can be found at the SANS Institute (www.sans.org).

**Data Principles**

9. **Principle: Data is an asset**
   
   Statement: Data is an asset that has value to the enterprise and is managed accordingly.
   
   Rationale: Data is a valuable corporate resource; it has real, measurable value. In simple terms, the purpose of data is to aid decision making. Accurate, timely data is critical to accurate, timely decisions. Most corporate assets are carefully managed, and data is no exception. Data is the foundation of our decision making, so we must also carefully manage data to assure that one knows where it is, can rely upon its accuracy, and can obtain it when and where one needs it.
   
   Implications:
   
   This is one of three closely related principles regarding data: data is an asset; data is shared; and data is easily accessible. The implication is that there is an educational task to ensure that all organizations within the enterprise understand the relationship between value of data, sharing of data, and accessibility to data.
   
   Stewards must have the authority and means to manage the data for which they are accountable.
   
   One must make the cultural transition from “data-ownership” thinking to “data-stewardship” thinking.
   
   The role of data steward is critical because obsolete, incorrect, or inconsistent data could be passed to enterprise personnel and adversely affect decisions across the enterprise.
   
   Part of the role of the data steward, who manages the data, is to ensure data quality. Procedures must be developed and used to prevent and correct errors in the information and to improve those processes that produce flawed information. Data quality will need to be measured and steps taken to improve data quality—it is probable that policy and procedures will need to be developed for this as well.
   
   A forum with comprehensive enterprisewide representation should decide on process changes suggested by the steward.
   
   Because data is an asset of value to the entire enterprise, data stewards accountable for properly managing the data must be assigned at the enterprise level.

10. **Principle: Data is shared**
    
    Statement: Users have access to the data necessary to perform their duties; therefore, data is shared across enterprise functions and organizations.
    
    Rationale: Timely access to accurate data is essential to improving the quality and efficiency of enterprise decision making. It is less costly to maintain timely, accurate data in a single application, and then share it, than it is to maintain duplicative data in multiple applications. The enterprise holds a wealth of data, but it is stored in hundreds of incompatible stovepipe databases. The speed of data collection, creation, transfer, and assimilation is driven by the ability of the organization to efficiently share these islands of data across the organization.
Shared data will result in improved decisions because we will rely on fewer (ultimately one virtual) sources of more accurate and timely managed data for all of our decision making. Electronically shared data will result in increased efficiency when existing data entities can be used, without rekeying, to create new entities.

Implications:
This is one of three closely related principles regarding data: **data is an asset; data is shared; and data is easily accessible.** The implication is that there is an educational task to ensure that all organizations within the enterprise understand the relationship between value of data, sharing of data, and accessibility to data.

To enable data sharing one must develop and abide by a common set of policies, procedures, and standards governing data management and access for both the short and the long term.

For the short term, to preserve our significant investment in legacy systems, we must invest in software capable of migrating legacy system data into a shared-data environment.

One will also need to develop standard data models, data elements, and other metadata that define this shared environment, and develop a repository system for storing this metadata to make it accessible.

For the long term, as legacy systems are replaced, one must adopt and enforce common data access policies and guidelines for new application developers to ensure that data in new applications remains available to the shared environment and that data in the shared environment can continue to be used by the new applications.

For both the short term and the long term one must adopt common methods and tools for creating, maintaining, and accessing the data shared across the enterprise.

Data sharing will require a significant cultural change.

This principle of data sharing will continually “bump up against” the principle of data security. Under no circumstances will the data-sharing principle cause confidential data to be compromised.

Data made available for sharing will have to be relied upon by all users to execute their respective tasks. This will ensure that only the most accurate and timely data is relied upon for decision making. Shared data will become the enterprisewide “virtual single source” of data.

11. Principle: **Data is accessible**

Statement: Data is accessible for users to perform their functions.

Rationale: Wide access to data leads to efficiency and effectiveness in decision making, and enables timely response to information requests and service delivery. Information use must be considered from an enterprise perspective to allow access by a wide variety of users. Staff time is saved, and consistency of data is improved.

Implications:
This is one of three closely related principles regarding data: **data is an asset; data is shared; and data is easily accessible.** The implication is that there is an educational task to ensure that all organizations within the enterprise understand the relationship between value of data, sharing of data, and accessibility to data.

Accessibility involves the ease with which users obtain information.

The way information is accessed and displayed must be sufficiently adaptable to meet a wide range of enterprise users and their corresponding methods of access.
Access to data does not constitute understanding of the data. Personnel should be cautious not to misinterpret information.

Access to data does not necessarily grant the user access rights to modify or disclose the data. This will require an educational process and a change in the organizational culture that currently supports a belief in “ownership” of data by functional units.

12. Principle: **Data trustee**

Statement: Each data element has a trustee accountable for data quality.

Rationale: One of the benefits of an architected environment is the ability to share data (e.g., text, video, sound, etc.) across the enterprise. As the degree of data sharing grows and business units rely on common information, it becomes essential that only the data trustee make decisions about the content of data. Because data can lose its integrity when it is entered multiple times, the data trustee will have sole responsibility for data entry, which eliminates redundant human effort and data storage resources. (Note that a trustee is different from a steward—the trustee is responsible for accuracy and currency of the data, whereas responsibilities of a steward may be broader and include data standardization and definition tasks.)

Implications:
- Real trusteeship dissolves the data “ownership” issues and allows the data to be available to meet all users’ needs. This implies that a cultural change from data “ownership” to data “trusteeship” may be required.
- The data trustee will be responsible for meeting quality requirements levied upon the data for which the trustee is accountable.
- It is essential that the trustee have the ability to provide user confidence in the data based upon attributes such as “data source.”
- It is essential to identify the true source of the data so that the data authority can be assigned this trustee responsibility. This does not mean that classified sources will be revealed, nor does it mean the source will be the trustee.
- Information should be captured electronically once and immediately validated as close to the source as possible. Quality control measures must be implemented to ensure the integrity of the data.
- As a result of sharing data across the enterprise, the trustee is accountable and responsible for the accuracy and currency of their designated data elements, and subsequently, must then recognize the importance of this trusteeship responsibility.

13. Principle: **Common vocabulary and data definitions**

Statement: Data is defined consistently throughout the enterprise, and the definitions are understandable and available to all users.

Rationale: The data that will be used in the development of applications must have a common definition throughout the headquarters to enable sharing of data. A common vocabulary will facilitate communications and enable dialogue to be effective. In addition, it is required to interface systems and exchange data.

Implications:
- We are lulled into thinking that this issue is adequately addressed because there are people with “data administration” job titles and forums with charters implying responsibility. Significant additional energy and resources must be committed to this task. It is a key to the success of efforts to improve the information environment. This is separate from but related to the issue of data element definition, which is addressed by a broad community—this is more like a common vocabulary and definition.
The enterprise must establish the initial common vocabulary for the business. The definitions will be used uniformly throughout the enterprise.
Whenever a new data definition is required, the definition effort will be coordinated and reconciled with the corporate “glossary” of data descriptions. The Enterprise Data Administrator will provide this coordination.
Ambiguities resulting from multiple parochial definitions of data must give way to accepted enterprisewide definitions and understanding.
Multiple data standardization initiatives need to be coordinated.
Functional data administration responsibilities must be assigned.

14. Principle: **Data security**

Statement: Data is protected from unauthorized use and disclosure. In addition to the traditional aspects of national security classification, this includes, but is not limited to, protection of predecisional, sensitive, source selection sensitive, and proprietary information.

Rationale: Open sharing of information and the release of information via relevant legislation must be balanced against the need to restrict the availability of classified, proprietary, and sensitive information.

Existing laws and regulations require the safeguarding of national security and the privacy of data, while permitting free and open access. Predecisional (work in progress, not yet authorized for release) information must be protected to avoid unwarranted speculation, misinterpretation, and inappropriate use.

Implications:
Aggregation of data, both classified and not, will create a large target requiring review and declassification procedures to maintain appropriate control. Data owners or functional users must determine if the aggregation results in an increased classification level. We will need appropriate policy and procedures to handle this review and declassification. Access to information based on a need-to-know policy will force regular reviews of the body of information.

The current practice of having separate systems to contain different classifications needs to be rethought. Is there a software solution to separating classified and unclassified data? The current hardware solution is unwieldy, inefficient, and costly. It is more expensive to manage unclassified data on a classified system. Currently, the only way to combine the two is to place the unclassified data on the classified system, where it must remain.

To adequately provide access to open information while maintaining secure information, security needs must be identified and developed at the data level, not the application level.

Data security safeguards can be put in place to restrict access to “view only,” or “never see.” Sensitivity labeling for access to predecisional, decisional, classified, sensitive, or proprietary information must be determined.

Security must be designed into data elements from the beginning; it cannot be added later. Systems, data, and technologies must be protected from unauthorized access and manipulation. Headquarters’ information must be safeguarded against inadvertent or unauthorized alteration, sabotage, disaster, or disclosure.

Need new policies on managing duration of protection for predecisional information and other works in progress, in consideration of content freshness.
Application Principles

15. Principle: **Technology independence**
   Statement: Applications are independent of specific technology choices and therefore can operate on a variety of technology platforms.
   Rationale: Independence of applications from the underlying technology allows applications to be developed, upgraded, and operated in the most cost-effective and timely way. Otherwise, technology, which is subject to continual obsolescence and vendor dependence, becomes the driver rather than the user requirements themselves.
   Realizing that every decision made with respect to information technology makes one dependent on that technology, the intent of this principle is to ensure that application software is not dependent on specific hardware and operating system software.
   Implications:
   - This principle will require standards that support portability.
   - For COTS and GOTS applications, there may be limited current choices, as many of these applications are technology and platform dependent.
   - Application programming interfaces (APIs) will need to be developed to enable legacy applications to interoperate with applications and operating environments developed under the enterprise architecture.
   - Middleware should be used to decouple applications from specific software solutions.
   - As an example, this principle could lead to use of JAVA, and future JAVA-like protocols, which give a high degree of priority to platform independence.

16. Principle: **Ease of use**
   Statement: Applications are easy to use. The underlying technology is transparent to users, so they can concentrate on tasks at hand.
   Rationale: The more a user has to understand the underlying technology, the less productive that user is. Ease of use is a positive incentive for use of applications. It encourages users to work within the integrated information environment instead of developing isolated systems to accomplish the task outside of the enterprise’s integrated information environment. Most of the knowledge required to operate one system will be similar to others. Training is kept to a minimum, and the risk of using a system improperly is low.
   Using an application should be as intuitive as driving a different car.
   Implications:
   - Applications will be required to have a common “look and feel” and support ergonomic requirements. Hence, the common look and feel standard must be designed and usability test criteria must be developed.
   - Guidelines for user interfaces should not be constrained by narrow assumptions about user location, language, systems training, or physical capability. Factors such as linguistics, customer physical infirmities (visual acuity, ability to use keyboard/mouse), and proficiency in the use of technology have broad ramifications in determining the ease of use of an application.

Technical Principles

17. Principle: **Requirements-based change**
   Statement: Only in response to business needs are changes to applications and technology made.
Rationale: This principle will foster an atmosphere where the information environment changes in response to the needs of the business, rather than having the business change in response to information technology changes. This is to ensure that the purpose of the information support—the transaction of business—is the basis for any proposed change. Unintended effects on business due to information technology changes will be minimized. A change in technology may provide an opportunity to improve the business process, and hence, change business needs.

Implications:
- Changes in implementation will follow full examination of the proposed changes using the enterprise architecture.
- No funding for a technical improvement or system development is approved unless a documented business need exists.
- Change-management processes conforming to this principle will be developed and implemented.
- This principle may bump up against the responsive change principle. One must ensure the requirements documentation process does not hinder responsive change to meet legitimate business needs. The purpose of this principle is to keep the focus on business, not technology, needs—responsive change is also a business need.

18. Principle: **Responsive change management**
Statement: Changes to the enterprise information environment are implemented in a timely manner.
Rationale: If people are to be expected to work within the enterprise information environment, that information environment must be responsive to their needs.
Implications:
- Process development for managing and implementing change must not create delays.
- A user who feels a need for change will need to connect with a "business expert" to facilitate explanation and implementation of that need.
- If one is going to make changes, one must keep the architectures updated.
- Adopting this principle might require additional resources.
- This will conflict with other principles (e.g., maximum enterprisewide benefit, enterprisewide applications, etc.).

19. Principle: **Control technical diversity**
Statement: Technological diversity is controlled to minimize the nontrivial cost of maintaining expertise in and connectivity between multiple processing environments.
Rationale: There is a real, nontrivial cost of infrastructure required to support alternative technologies for processing environments. There are further infrastructural costs incurred to keep multiple processor constructs interconnected and maintained.
Limiting the number of supported components will simplify maintainability and reduce costs.
The business advantages of minimum technical diversity include standard packaging of components, predictable implementation impact, predictable valuations and returns, redefined testing, utility status, and increased flexibility to accommodate technological advancements. Common technology across the enterprise brings the benefits of economies of scale to the enterprise. Technical administration and support costs are better controlled when limited resources can focus on this shared set of technology.
Implications:
- Policies, standards, and procedures that govern acquisition of technology must be tied directly to this principle.
Technology choices will be constrained by the choices available within the technology blueprint. Procedures for augmenting the acceptable technology set to meet evolving requirements will have to be developed and emplaced.

There will be no freezing of the technology baseline. Technology advances will be welcomed and will change the technology blueprint when compatibility with the current infrastructure, improvement in operational efficiency, or a required capability has been demonstrated.

20. Principle: **Interoperability**

Statement: Software and hardware should conform to defined standards that promote interoperability for data, applications, and technology.

Rationale: Standards help ensure consistency, thus improving the ability to manage systems and improve user satisfaction, and protect existing IT investments, thus maximizing return on investment and reducing costs. Standards for interoperability additionally help ensure support from multiple vendors for their products, and facilitate supply-chain integration.

Implications:
- Interoperability standards and industry standards will be followed unless there is a compelling business reason to implement a nonstandard solution.
- A process for setting standards, reviewing and revising them periodically, and granting exceptions must be established.
- The existing IT platforms must be identified and documented.
This chapter deals with business process modeling (BPM) and requirements gathering/analysis in support of the business architecture. There are a number of modeling approaches; the chapter focuses on a handful of specific ones. Three topics are covered: (1) the Business Process Modeling Language (BPML) and the Business Process Modeling Notation (BPMN), both promulgated by the Business Process Management Initiative (BPMI.org)*; (2) the Unified Modeling Language™ (UML)†; and (3) the Model-Driven Architecture™ (MDA) (UML and MDA are both promulgated by the Object Management Group (OMG)). Figure 7.1 depicts the context of these tools, whereas Figure 7.2 provides a more inclusive snapshot of the environment.

Large enterprise applications must be more than just an aggregate of software modules: these applications must be structured (architected) in a way that the architecture enables scalability and reliable execution under normal or stressed conditions (of course, a well-designed architecture benefits any application, not just the large ones). The structure of these applications must be defined clearly and unambiguously so that (1) maintenance staff can quickly locate and fix any bugs that may show up long after the original programmers have moved on, and so that (2) developers can add new features that may be required over time by the business users. Another benefit of a architected structure is that it enables code reuse: design time is the best time to seek to structure an application as a collection of self-contained modules or components. Eventually, enterprises build up a library of models of components, each one representing an implementation stored in a

---

* In June of 2005, the Business Process Management Initiative (BPMI.org) and the Object Management Group™ (OMG™) announced the merger of their Business Process Management (BPM) activities to provide thought leadership and industry standards for this vital and growing industry. The combined group has named itself the Business Modeling & Integration (BMI) Domain Task Force (DTF). The BMI’s combined activities continue BPMI’s and OMG’s work and focus on all aspects of business process management. BPMI’s widely used standard for business modeling, Business Process Modeling Notation (BPMN), started the comment period required by OMG’s fast-track “Request for Comment” (RFC) adoption process at the OMG’s September 2005 Technical Meeting.

† The UML modeling format was originally introduced by Rational Software (now owned by IBM) and later adopted by the OMG as a standard.
library of code modules. When another application needs the same functionality, the designer can quickly import this module from the library; at coding time, the developer can import the code module into the executable [OMG200501].

In this context, modeling is the process of architecting and structurally designing a software application before starting the coding phase. Modeling is a critical effort for large software projects, and it is also useful for medium projects. Using a model, developers can assure themselves that business functionality is complete and correct, that end-user needs are met, and that program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation in code makes changes difficult and expensive to make. Models are useful because they allow one to work at a higher level of abstraction. A model may do this by hiding or masking details, bringing out the “big picture,” or by focusing on different aspects of the prototype. Typically, one wants to be able to zoom out from a detailed view of an application to the environment where it executes, visualizing connections to other applications or, even further, to other sites. Alternatively, one may want to focus on different aspects of the application, such as the business process that it automates, or a business rules view.
Returning to the topics covered in this chapter, BPML is a metalanguage for the modeling of business processes; it provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine. Its associated graphical notation, BPMN, is designed to be understandable by business users, by business analysts that create the initial drafts of the processes, by technical developers responsible for implementing the technology that will perform those processes, and by the business people who will manage and monitor those processes [BPM200501].

UML standardizes representation of object-oriented analysis and design. It lets architects and analysts visualize, specify, construct, and document applications in a standard way. The graphical language contains a dozen of diagram types including Use Case and Activity diagrams for requirements gathering, Class and Object diagrams for design, and Package and Subsystem diagrams for deployment. The key advantage of UML is that the models remain stable even as the technological landscape changes around them [OMG200501]. UML is a language with a broad scope that covers a diverse set of application domains; not all of its modeling capabilities are necessarily useful in all domains or applications; for this reason the language is structured modularly, with the ability to select only those parts of the language that are of direct interest.

MDA has the goal of unifying the modeling and middleware environments. MDA supports applications over their entire life cycle from analysis and design, through implementation and deployment, to maintenance and evolution. Based on UML models, MDA-based development seeks to integrate applications across the enterprise, and integrate the applications of one enterprise with applications of another [OMG200501]. Executable system code can be automatically generated by an MDA-based model (using an appropriate tool).

This chapter is only an introduction—it covers only some high-level aspects of these topics; interested readers should consult the cited documentation for detailed information. Note that to retain a manageable scope, this textbook does not discuss data modeling in support of the information architecture; the interested reader should look for this information in the appropriate literature. Also, the textbook does not cover the topic of application modeling to support the systems/solution architecture, except as achievable with UML/MDA, and except for service-oriented architecture (SOA) approaches, which are covered in Chapter 8.

### 7.1 Business Process Modeling

Business process modeling (BPM) seeks to standardize the management of business processes that span multiple applications, multiple data repositories, multiple corporate departments, or even multiple companies (or government agencies). BPM provides the foundation for interoperability, whether among departments or among affiliated organizations.

Business process modeling received a lot of attention in the early 1990s. Reengineering efforts arose in many quarters, including federal agencies that used the technique to rethink outdated ways of doing business. Staffers worked to document an organization's as-is environment and define the to-be environment, which aimed at supporting efficient processes and workflow arrangements. However, reengineering proved to be fairly time consuming and expensive; hence, the concept fell out of favor relatively quickly. At this juncture business process modeling is receiving a lot of renewed attention, driven by the Bush Administration's enterprise architecture mandate for government agencies. Business process modeling provides a way of visualizing the often-complex workflows within an organization. The idea is to create a graphical representation of business processes that describes activities and their interdependencies. The resulting diagram reveals
inefficiencies and areas for improvement. Governmentwide enterprise architecture initiatives have rekindled interest in business process modeling because it helps create such architectures, which are models of agency information technology systems and business operations [MOO200401]. The Clinger–Cohen Act of 1996 requires agencies to create enterprise architectures. OMB is now mandating Clinger–Cohen compliance; that calls for as-is and to-be business process models to be supported by an economic analysis demonstrating the benefits of a given investment.

BPM typically has the following objectives:

1. Obtaining knowledge about the business processes of the enterprise
2. Utilizing business process knowledge in business process reengineering projects to optimize the operation
3. Facilitating the decision-making efforts of the enterprise
4. Supporting interoperability of the business processes

Ultimately, the idea of “reuse,” with the goal of saving run-the-engine (RTE) costs, relies on (1) being able to apply a certain number of modeled processes from one department or application to another department or application, or (2) linking the various departmental models into an enterprise model. BPM is important within the context of an enterprise; however, the shift toward the “extended enterprise” paradigm makes objective #4, interoperability of business process models, even more critical. BPM standardization promotes inter- and intraenterprise business integration and collaboration, by developing shared models that support business modeling and the integration of systems, processes, and information across the enterprise, including business partners and customers [OMG200501]. International standards organizations and other advocacy groups deal with the subject at different levels of abstraction, providing architectures, frameworks, and explicit standards for different application arenas.

Modeling is a best-practices approach to ensure that enterprise IT systems deliver the functionality that a business requires, while at the same time enabling such systems to evolve in a controlled manner as business needs change over time. Systems that have been properly modeled are able to evolve more effectively over time, thus enabling firms to maximize the return on investment (ROI) for IT assets [SEI200201]. Clearly, one is interested in technology-independent representations of the business functionality and behavior. Models should enable the firm to represent exactly what a business application does in an industry-accepted manner.

For business process modeling, one therefore finds older (1990s vintage) and newer (2000s vintage) methods. Integrated Computer-Aided Manufacturing Definition (IDEF*) was the approach of choice in the 1990s and remains the only one compliant with Federal Information Processing Standards (FIPS). IDEF was developed 25 years ago and was prevalent in the 1990s; currently, IDEF plays a role in many enterprise architecture efforts. IDEF refers to a group of methods, each of which fulfills a specific purpose. IDEF0, for example, is used to model an organization’s functions, whereas IDEF1x is used for data modeling. IDEF0 and IDEF1x are the most heavily used IDEF methods in government; both were published as FIPS in 1993. There are several tools that support IDEF [MOO200401]. However, joining IDEF now are two other techniques identified earlier: UML and BPMN. Many view IDEF as being “nonintuitive”; BPMN’s creators recognized the need for a modeling notation that both business audiences and solution delivery specialists

* To be precise, IDEF is an abbreviation of another abbreviation. The workgroup Air Force Program for Integrated Computer-Aided Manufacturing (ICAM) developed ICAM definition language—ICAM Definition or IDEF. The IDEF is set of languages (IDEF 0,1,3,4,5). The mostly commonly used ones are IDEF 0 and IDEF 3. IDEF 0 is comparable to the UML Use Case diagram; IDEF 3 is comparable to the UML Activity diagram.
could interpret. The Business Process Management Initiative, which released BPMN 1.0 in 2003, describes the method as a common visual vocabulary for depicting business processes (companies such as Computer Sciences Corp., EDS, IBM Corp., Microsoft Corp. and Popkin Software reportedly support BPMN).

As noted, a number of approaches to BPM are available; these are modeling mechanisms that may be used to define a business architecture by a firm. Some of the newer BPM approaches are discussed in the sections that follow.

### 7.2 Business Process Modeling Standardization

#### 7.2.1 Business Process Modeling Language

The Business Process Modeling Language (BPML) is one example of an effort at BPM standardization. BPML is a metalanguage for the modeling of business processes. It provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine [BPM200501]. The language provides a model for expressing business processes and supporting entities. BPML defines a formal model for expressing abstract and executable processes that address all aspects of enterprise business processes, including activities of varying complexity, transactions and their compensation, data management, concurrency, exception handling, and operational semantics. BPML also provides a grammar in the form of an eXtensible Markup Language (XML) Schema for enabling the persistence and interchange of definitions across heterogeneous systems and modeling tools.

BPML itself does not define any application semantics such as particular processes or application of processes in a specific domain; rather, BPML defines an abstract model and grammar for expressing generic processes. This allows BPML to be used for a variety of purposes that include, but are not limited to, the definition of enterprise business processes, the definition of complex Web Services (WS), and, the definition of multiparty collaborations. A brief description of BPML follows, based on [BPM200201].

#### 7.2.1.1 Activities

In BPML an activity is a component that performs a specific function. Complex activities are composed of other activities and direct their execution. A process is such a composition, and may itself be an activity within a larger process. The semantics of an activity definition apply to a process definition with a few exceptions.

#### 7.2.1.2 Activity Types

An activity definition specifies the manner in which a given activity will execute. The behavior is defined by specifying the values of the activity’s attributes. An activity type definition specifies the attributes that are used in the definition of an activity of that type, and how the values of these attributes affect the execution of that activity. The BPML specification defines 17 activity types, and three process types. All activity types are derived from a common base type. The base type defines the following attributes:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The activity name (optional)</td>
</tr>
<tr>
<td>Documentation</td>
<td>Documentation (optional)</td>
</tr>
<tr>
<td>Other</td>
<td>Other attributes defined for the specific activity type</td>
</tr>
</tbody>
</table>

The *name* attribute provides a name that can be used to reference the activity definition or activity instance. Two activity definitions are distinct even if they have the same name. It is not an error if within a given context that name would reference both activity definitions. With the exception of process definitions, all activity definitions have an ordinal position within an activity list. If the *name* attribute is unspecified, the activity name is its ordinal position, for example, “1” for the first activity in the activity list, “2” for the second activity, and so forth. The *name* attribute is optional for all but process definitions. An activity type may define additional attributes that are specific to that type, for example, the *operation* attribute of the *action* activity, or the *condition* attribute of the *while* activity.

The syntax for the base type `bpml:activity` is as follows:

```xml
<{activity type}
name = NCName
{other attributes}>
Content: (documentation?, {other element}*)
</{activity type}>
```

Each activity type defines a syntax that specifies additional XML attributes and XML elements that represent values of the abstract model attributes. Other specifications may introduce additional activity types. The XML elements for these activity types are derived from the type `bpml:activity` and use the substitution group `bpml:otherActivity`. They must be defined in a namespace other than the BPML namespace.

The BPML specification defines simple activity types and complex activity types. A description follows.

### 7.2.1.2.1 Simple Activity Type

The verb list that follows identifies basic activity types.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Performs or invokes an operation involving the exchange of input and output messages.</td>
</tr>
<tr>
<td>assign</td>
<td>Assigns a new value to a property.</td>
</tr>
<tr>
<td>call</td>
<td>Instantiates a process and waits for it to complete.</td>
</tr>
<tr>
<td>compensate</td>
<td>Invokes compensation for the named processes.</td>
</tr>
<tr>
<td>delay</td>
<td>Expresses the passage of time.</td>
</tr>
<tr>
<td>empty</td>
<td>Does nothing.</td>
</tr>
<tr>
<td>fault</td>
<td>Throws a fault in the current context.</td>
</tr>
<tr>
<td>raise</td>
<td>Raises a signal.</td>
</tr>
<tr>
<td>spawn</td>
<td>Instantiates a process without waiting for it to complete.</td>
</tr>
<tr>
<td>synch</td>
<td>Synchronizes on a signal.</td>
</tr>
</tbody>
</table>
7.2.1.2.2 Complex Activity Type

The list that follows identifies complex activity types.

- **all** Executes activities in parallel.
- **choice** Executes activities from one of multiple sets, selected in response to an event.
- **foreach** Executes activities once for each item in an item list.
- **sequence** Executes activities in sequential order.
- **switch** Executes activities from one of multiple sets, selected based on the truth value of a condition.
- **until** Executes activities once or more based on the truth value of a condition.
- **while** Executes activities zero or more times based on the truth value of a condition.

7.2.1.3 The Activity Context

Activities that execute in the same context use the context to exchange information through properties defined in that context. For example, an activity that receives an input message sets the value of a property from the contents of the input message. A subsequent activity uses the value of that property to construct and send an output message. The context defines common behavior for all activities executing in that context, such as handling of exceptional conditions and faults, providing atomic semantics, defining a time constraint, and so forth. The context in which an activity executes is referred to as its current context. Activities and contexts are composed hierarchically. The current context of an activity may be the child context of some other context, and the parent of multiple child contexts.

The term downstream activity refers to an activity that executes following another activity. The downstream activity may depend on the value of properties set by the current activity, a signal raised by the current activity, or the instantiation of another activity from the current activity. Activities that execute in the same context are grouped together into an activity set. The activity set is a composition of one or more activity definitions and the definition of the context in which these activities execute—their current context.

The activity set contains an activity list, an ordered list of activity definitions. Generally, activities from the activity list are executed in sequential order—an activity must complete before executing the next activity in the list. The BPML specification defines one activity that causes activities from the activity list to execute in parallel. The activity set may define activities that can be executed multiple times in parallel with other activities defined in the activity set. These activities are modeled as process definitions and are contained in the activity set’s context definition. These are referred to as nested processes. The activity set may define activities that execute in response to exceptional condition and interrupt the execution of all other activities defined in the activity set. These activities are defined in a similar manner to nested processes and are referred to as exception processes.

The activity set construct is a composition of the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>A context definition (optional)</td>
</tr>
<tr>
<td>activities</td>
<td>One or more activity definitions (ordered)</td>
</tr>
</tbody>
</table>
The syntax for the activity set is given as follows:

\[
\text{Content: } (\text{context}?, \{\text{any activity}\}+)\]

The context element is absent if the context definition contains no local definitions (an empty context). The activity list must contain at least one activity definition. Any activity type may be used in the activity list including activity types defined in other specifications, with the exception of process definitions. Nested process definitions appear inside the context element. The occurrence of the bpm:activitySet model group in the content of an XML element indicates that it contains an activity set.

7.2.1.4 Simple and Complex Activities

A simple activity is an activity that cannot be further decomposed. For example, the action activity that performs a single operation, or the assign activity that assigns a new value to a property (see Figure 7.3). A complex activity is a composition of one or more activities. They may be simple activities or complex activities that are recursively composed of simple and complex activities. A complex activity definition contains one or more activity sets and directs the execution of activities from one of these activity sets. A complex activity that contains multiple activity sets must select which one activity set to use. The choice activity waits for an event to be triggered and selects the activity set associated with that event handler. The switch activity evaluates conditions and selects the activity set associated with a condition that evaluates to true. All other complex activities defined in the BPML specification contain a single activity set (see Figure 7.3).

A complex activity determines the number of times to execute activities from the activity set. The until activity repeats executing activities until a condition evaluates to true. The while activity

![Diagram of activity types defined in the BPML specification.](image-url)
repeats executing activities while the condition evaluates to true. The foreach activity repeats executing activities, once for each item in the item list. All other complex activities defined in the BPML specification execute activities from the activity set exactly once. A complex activity determines the order in which activities are executed. The sequence activity executes all activities from the activity set’s list in sequential order. The all activity executes all activities from the activity set’s list in parallel. All other complex activities defined in the BPML specification execute activities in sequential order. The complex activity completes after it has completed executing all activities from the activity set. This includes all activities that are defined in the activity list, and all processes instantiated from a definition made in the activity set’s context. Nested processes and exception processes are considered activities of the activity set.

Simple activities generate faults if they cannot complete because of an unexpected error. Complex activities throw faults if one of their activities generate a fault and they cannot recover from that fault. The complex activity aborts when a fault is thrown by one of its activities. To abort, the complex activity terminates all of its executing activities.

7.2.1.5 Processes

A process is a type of complex activity that defines its own context for execution. Similar to other complex activity types, it is a composition of activities, and it directs their execution. A process can also serve as an activity within a larger composition, either by defining it as part of a parent process or by invoking it from another process. Processes are often defined as reusable units of work.

A process that is defined independently of other processes is called a top-level process, because its definition is found at the package level. A process that is defined to execute within a specific context is called a nested process, because its definition is part of that context’s definition. An exception process is defined as part of a parent process to handle exceptional conditions that may interrupt activities executing in that process. A compensation process provides the compensation logic for its parent process. Exception processes and compensation processes are specific type of process definitions.

A process can be instantiated from the call, compensate, and spawn activities and from a schedule. Alternatively, it may define an instantiation event that responds to an input message, or instantiation event that responds to a raised signal. A BPML implementation that detects a process definition that violates one of the constraints defined in the BPML specification should flag the process definition as erroneous. It must not create instances from an erroneous process definition. A process definition is also erroneous if it references an erroneous process definition from the call and spawn activities.

Interested readers are referred to [BPM200201] for more information on BPML.

7.2.2 Business Process Modeling Notation

The Business Process Management Initiative (BPMI) has developed a standard modeling notation called the Business Process Modeling Notation (BPMN). This specification provides a graphical notation for expressing business processes in a Business Process Diagram (BPD). The BPMN specification also provides a binding between the notation’s graphical elements and the constructs of block-structured process execution languages, including BPML and Business Process Execution Language for Web Services (BPEL4WS). The first draft of BPMN became available in late
2002. This section is summarized from [BPM200301]; interested readers should consult the entire specification.

7.2.2.1 Introduction

The primary goal of BPMN is to provide a notation that is readily understandable by business users, from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor the processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation. Another goal is to ensure that XML languages designed for the execution of business processes, such as BPEL4WS, can be visualized with a common notation.

This specification defines the notation and semantics of a BPD and represents the amalgamation of best practices within the business modeling community. The intent of BPMN is to standardize a business process modeling notation in the face of many different modeling notations and viewpoints. In doing so, BPMN provides a simple means of communicating process information to other business users, process implementers, customers, and suppliers. The membership of the BPMI Notation Working Group has expertise and experience with the many existing notations and has sought to consolidate the best ideas from these divergent notations into a single standard notation. Examples of other notations or methodologies that were reviewed are UML Activity Diagram, UML Enterprise Distributed Object Computing (EDOC) Business Processes, Integrated Computer-Aided Manufacturing Definition (IDEF), Electronic Business using eXtensible Markup Language (ebXML) Business Process Specification Schema (BPSS) (ebXML BPSS*), Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM, and Event-Process Chains (EPCs).

7.2.2.2 BPMN Overview

There has been much activity in the past few years in developing Web-service-based XML execution languages for BPM systems. Languages such as BPEL4WS provide a formal mechanism for the definition of business processes. The key element of such languages is that they are optimized for the operation and interoperation of BPM Systems. The optimization of these languages for software operations renders them less suited for direct use by humans to design, manage, and monitor business processes. BPEL4WS has both graph and block structures and utilizes the principles of

---

* The Electronic Business using eXtensible Markup Language (ebXML) project was started in 1999, jointly by OASIS and UN/CEFACT, the United Nations Centre for Trade Facilitation and Electronic Business, to provide a modular suite of specifications that enable enterprises of any size and in any location to conduct business over the Internet. The original project envisioned and delivered five layers of substantive data specification, including XML standards for: business processes, core data components, collaboration protocol agreements, messaging, registries, and repositories. The ebXML Business Process Specification Schema provides a generic framework for business process collaborations, both between two parties/partners (binary) and multiparty (expressed as two or more binary collaborations). This framework includes the means and descriptions by which one or more activities are performed in performing a business collaboration. This business collaboration could be part of an enterprise-to-enterprise collaboration (B2B) or within an enterprise for a collaboration that is required to be enforceable. The original BPSS version 1.01 was approved in 2001. ebXML has seen some adoption since its release, e.g., by General Motors, the U.S. Centers for Disease Control (CDC), and TransCanada Pipelines.
formal mathematical models. This technical underpinning provides the foundation for business process execution to handle the complex nature of both internal and business-to-business (B2B) interactions and take advantage of the benefits of Web Services. Given the nature of BPEL4WS, a complex business process could be organized in a potentially complex, disjointed, and unintuitive format that is handled well by a software system (or a computer programmer), but would be hard to understand by the business analysts and managers tasked to develop, manage, and monitor the process. Thus, there is a human level of “interoperability” or “portability” that is not addressed by these WS-based XML execution languages.

Business people are often comfortable with visualizing business processes in a flowchart format. There are thousands of business analysts studying the way companies work and defining business processes with simple flowcharts. This creates a technical gap between the format of the initial design of business processes and the format of the languages, such as BPEL4WS, that will execute these business processes. This gap needs to be bridged with a formal mechanism that maps the appropriate visualization of the business processes (a notation) to the appropriate execution format (a BPM execution language) for these business processes.

Interoperation of business processes at the human level, rather than the software engine level, can be solved with standardization of BPMN. BPMN provides a BPD, which is a diagram designed for use by the people who design and manage business processes. BPMN also provides a formal mapping to an execution language of BPM Systems (BPEL4WS). Thus, BPMN provides a standard visualization mechanism for business processes defined in an execution-optimized business process language.

BPMN provides businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standardized manner. Currently, there are many process modeling tools and methodologies. Given that individuals may move from one company to another and that companies may merge and diverge, it is likely that business analysts are required to understand multiple representations of business processes—potentially different representations of the same process as it moves through its life cycle of development, implementation, execution, monitoring, and analysis. Therefore, a standard graphical notation facilitates the understanding of the performance collaborations and business transactions within and between the organizations. This ensures that businesses understand their own environments and the environment of participants in their business, and will enable organizations to adjust to new internal and B2B business circumstances quickly. To do this, BPMN follows the tradition of flowcharting notations for readability but at the same time provides mapping to the executable constructs. BPMI has used the experience of the business process notations that preceded BPMN to create the next-generation notation that combines readability, flexibility, and expandability.

BPMN also advances the capabilities of traditional business process notations by inherently handling B2B business process concepts, such as public and private processes and choreographies, as well as advanced modeling concepts, such as exception handling and transaction compensation.

7.2.2.2.1 BPMN Scope

BPMN is constrained to support only the concepts of modeling that are applicable to business processes. This means that other types of modeling done by organizations for business purposes will be outside BPMN’s scope; for example, the modeling of the following will not be a part of BPMN: organizational structures, functional breakdowns, and data models. In addition, although
BPMN will show the flow of data (messages) and the association of data artifacts to activities, it is not a data flow diagram.

7.2.2.2.2 Uses of BPMN

Business process modeling is used to communicate a wide variety of information to a wide variety of audiences. BPMN is designed to cover this wide range of usage and allows modeling of end-to-end business processes to allow the viewer of the diagram to be able to easily differentiate between sections of a BPMN diagram. There are three basic types of submodels within an end-to-end BPMN model:

- Private (internal) business processes
- Abstract (public) processes
- Collaboration (global) processes

7.2.2.2.2.1 Private (Internal) Business Processes

Private business processes are those that are internal to a specific organization and are the types of processes that have been generally called workflow or BPM processes. A single private business process will map to a single BPEL4WS document. If swimlanes are used, then a private business process will be contained within a single Pool. The Sequence Flow of the Process is therefore contained within the Pool and cannot cross its boundaries. Message Flow can cross the Pool boundary to show the interactions that exist among separate private business processes. Thus, a single BPMN diagram may show multiple private business processes, each mapping to a separate BPEL4WS process.

7.2.2.2.2.2 Abstract (Public) Processes

This represents the interactions between a private business process and another process or participant. Only those activities that are used to communicate outside the private business process are included in the abstract process. All other “internal” activities of the private business process are not shown in the abstract process. Thus, the abstract process shows to the outside world the sequence of messages that is required to interact with that business process. A single abstract process may be mapped to a single BPEL4WS abstract process (however, this mapping will not be done in this specification). Abstract processes are contained within a Pool and can be modeled separately or within a larger BPMN diagram to show the Message Flow between the abstract process activities and other entities. If the abstract process is in the same diagram as its corresponding private business process, then the activities that are common to both processes can be associated.

7.2.2.2.2.3 Collaboration (Global) Processes

A collaboration process depicts the interactions among two or more business entities. These interactions are defined as a sequence of activities that represents the message exchange patterns among the entities involved. A single collaboration process may be mapped to various collaboration languages, such as ebXML BPSS, RosettaNet, or the resultant specification from the W3C Choreography Working Group (however, these mappings are considered as future directions for BPMN). Collaboration processes may be contained within a Pool, and the different participant business interactions are shown as Lanes within the Pool. In this situation, each Lane would represent two participants and a direction of travel between them. They may also be shown as two
or more Abstract Processes interacting through Message Flow. These processes can be modeled separately or within a larger BPMN diagram to show the Associations between the collaboration process activities and other entities. If the collaboration process is in the same diagram as one of its corresponding private business processes, then the activities common to both processes can be associated.

7.2.2.2.2.4 Types of BPD Diagrams
Within and between these three BPMN submodels, many types of diagrams can be created. The following are the types of business processes that can be modeled with BPMN (those with asterisks may not map to an executable language):

- High-level private process activities (not functional breakdown)*
- Detailed private business process
  - As-is, or old, business process*
  - To-be, or new, business process
- Detailed private business process with interactions among one or more external entities (or “black box” processes)
- Two or more detailed private business processes interacting
- Detailed private business process relationship with Abstract Process
- Detailed private business process relationship with Collaboration Process
- Two or more Abstract Processes*
- Abstract Process relationship with Collaboration Process*
- Collaboration Process only (e.g., ebXML BPSS, or RosettaNet)*
- Two or more detailed private business processes interacting through their Abstract Processes
- Two or more detailed private business processes interacting through a Collaboration Process
  - Two or more detailed private business processes interacting through their Abstract Processes and a Collaboration Process

BPMN is designed to allow all the foregoing types of diagrams. However, it should be cautioned that if too many types of submodels are combined, such as three or more private processes with message flow between each of them, then the diagram may become too hard for someone to understand. Thus, we recommend that the modeler pick a focused purpose for the BPD, such as a private process, or a collaboration process.

7.2.2.2.2.5 BPMN Mappings
Because BPMN covers such a wide range of usage, it will map to more than one lower-level specification language:

- BPEL4WS are the primary languages that BPMN will map to, but they only cover a single executable private business process. If a BPMN diagram depicts more than one internal business process, then there will a separate mapping for each of the internal business processes.
- The abstract sections of a BPMN diagram will be mapped to Web service interfaces specifications, such as the abstract processes of BPEL4WS.
The Collaboration model sections of a BPMN will be mapped to Collaboration models such as ebXML BPSS, RosettaNet, and the W3C Choreography Working Group Specification (when it is completed).

The BPMN specification will only cover the mappings to BPEL4WS. Mappings to other specifications will have to be a separate effort, or perhaps a future direction of BPMN (beyond Version 1.0 of the BPMN specification). One cannot predict which mappings will be applied to BPMN at this point, as process language specifications is a volatile area of work, with many new offerings and mergings.

A BPD is not designed to graphically convey all the information required to execute a business process. Thus, the graphic elements of BPMN will be supported by attributes that will supply the additional information required to enable a mapping to BPEL4WS.

7.2.2.2.3 Diagram Point of View

As a BPMN diagram may depict the processes of different Participants, each Participant may view the diagram differently; that is, the Participants have different points of view regarding how the processes will behave. Some of the activities will be internal to the Participant (meaning performed by or under control of the Participant), and other activities will be external to the Participant. Each Participant will have a different perspective as to which processes are internal and external. At runtime, the difference between internal and external activities is important in how a Participant can view the status of the activities or troubleshoot any problems. However, the diagram itself remains the same. Figure 7.4 displays as an example a simple Business Process that has two points of view. One point of view is of a Patient, the other is of the Doctor’s office. The diagram shows the activities of both participants in the process, but when the process is actually being performed, each Participant will really have control over his or her own activities.

Although the diagram point of view is important for a viewer of the diagram to understand how the behavior of the process will relate to that viewer, BPMN does not currently specify any graphical mechanisms to highlight the point of view. It is open to the modeler or modeling tool vendor to provide any visual cues to emphasize this characteristic of a diagram.

![Figure 7.4: A business process diagram with two points of view.](image-url)
7.2.2.2.4 Extensibility of BPMN and Vertical Domains

BPMN is intended to be extensible by modelers and modeling tools. This extensibility allows modelers to add nonstandard elements or artifacts to satisfy a specific need, such as the unique requirements of a vertical domain. While being extensible, BPMN diagrams should still have the basic look and feel, so that a diagram by any modeler should be easily understood by any viewer of the diagram. Thus, the footprint of the basic flow elements (Events, Activities, and Gateways) should not be altered, nor should any new flow elements be added to a BPD, because there is no specification as to how Sequence and Message Flow will connect to any new flow object. In addition, mappings to execution languages may be affected if new flow elements are added. To satisfy additional modeling concepts that are not part of the basic set of flow elements, BPMN provides the concept of Artifacts that can be linked to the existing flow objects through Associations. Thus, Artifacts do not affect the basic Sequence or Message Flow, nor do they affect mappings to execution languages. The graphical elements of BPMN are designed to be open to allow specialized markers to convey specialized information. For example, the three types of Events all have open centers for the markers that BPMN standardizes, as well as for user-defined markers.

7.2.2.3 Business Process Diagrams

This section provides a summary of the BPMN graphical objects and their interrelationships. One of the goals of BPMN is that the notation be simple and adoptable by business analysts. Also, there is a potentially conflicting requirement that BPMN provide the power to depict complex business processes and map to BPM execution languages. To help understand how BPMN can manage both requirements, the list of BPMN graphic elements is presented in two groups.

First, there are the core elements that support the requirement of a simple notation. These are the elements that define the basic look and feel of BPMN. Most business processes can be modeled adequately with these elements. Second, all the elements, including the core elements, help support the requirement of a powerful notation to handle more advanced modeling situations. Further, the graphical elements of the notation are supported by nongraphical attributes that provide the remaining information necessary to map to an execution language or for other business modeling purposes.

7.2.2.3.1 BPD Core Element Set

It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for creating business process models. Of the core element set, there are three primary modeling elements (flow objects):

- Events
- Activities
- Gateways

There are three ways of connecting the primary modeling elements:

- Sequence Flow
- Message Flow
- Association
There are two ways of grouping the primary modeling elements through Swimlanes:

- Pools
- Lanes

Table 7.1 displays a list of the core modeling elements that are depicted by the notation. Table 7.2 displays a more extensive list of the business process concepts that could be depicted through a business process modeling notation.

### 7.2.2.3.2 Flow Object Connection Rules

An incoming Sequence Flow can connect to any location on a flow object (left, right, top, or bottom). Likewise, an outgoing Sequence Flow can connect from any location on a flow object (left, right, top, or bottom). Message Flows also have this capability. BPMN allows this flexibility; however, we also recommend that modelers use judgment or best practices in how flow objects should be connected so that readers of the diagrams will find the behavior clear and easy to follow. This is even more important when a diagram contains Sequence Flows and Message Flows. In these situations it is best to pick a direction of Sequence Flow, either left to right or top to bottom, and then direct the Message Flow at a 90° angle to the Sequence Flow. The resulting diagrams will be much easier to understand.

#### 7.2.2.3.2.1 Sequence Flow Rules

Table 7.3 displays the BPMN flow objects and shows how these objects can connect to one another through Sequence Flows. The symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies not specified here. Refer to the sections in the next chapter for each individual object for more detailed information on the appropriate connection rules. Note that if a subprocess has been expanded within a diagram, the objects within the subprocess cannot be connected to objects outside of the subprocess. Nor can Sequence Flows cross a Pool boundary.

#### 7.2.2.3.2.2 Message Flow Rules

Table 7.4 displays the BPMN modeling objects and shows how they can connect to one another through Message Flows. The symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies that are not specified here. Refer to the sections in the next chapter for each individual object for more detailed information on the appropriate connection rules. Note that Message Flows cannot connect to objects that are within the same Participant Lane boundary.

### 7.2.2.4 Examples

The description given earlier is only a truncated synopsis of the modeling language. The interested reader should consult the full documentation. A handful of examples follow for illustrative purposes (Figures 7.5 through 7.10.) The figures that follow are just some examples.

Interested readers are referred to [BPM200301] for more information on BPMN.
# Business Process Modeling

## Table 7.1 BPD Core Element Set

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>An event is something that “happens” during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.</td>
<td><img src="image" alt="Event Notation" /></td>
</tr>
<tr>
<td>Activity</td>
<td>An activity is a generic term for work that the company performs. An activity can be atomic or nonatomic (compound). The types of activities that are a part of a Process Model are Process, Subprocess, and Task. Tasks and Subprocesses are rounded rectangles. Processes are either unbounded or a contained within a Pool.</td>
<td><img src="image" alt="Activity Notation" /></td>
</tr>
<tr>
<td>Gateway</td>
<td>A Gateway is used to control the divergence and convergence of Sequence Flow. Thus, it will determine branching, forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.</td>
<td><img src="image" alt="Gateway Notation" /></td>
</tr>
<tr>
<td>Sequence Flow</td>
<td>A Sequence Flow is used to show the order that activities will be performed in a Process.</td>
<td><img src="image" alt="Sequence Flow Notation" /></td>
</tr>
<tr>
<td>Message Flow</td>
<td>A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the diagram will represent the two entities (participants).</td>
<td><img src="image" alt="Message Flow Notation" /></td>
</tr>
<tr>
<td>Association</td>
<td>An Association is used to associate information with flow objects. Text and graphical nonflow objects can be associated with the flow objects.</td>
<td><img src="image" alt="Association Notation" /></td>
</tr>
<tr>
<td>Pool</td>
<td>A Pool is a “swimlane” and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.</td>
<td><img src="image" alt="Pool Notation" /></td>
</tr>
<tr>
<td>Lane</td>
<td>A Lane is a subpartition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities.</td>
<td><img src="image" alt="Lane Notation" /></td>
</tr>
</tbody>
</table>
## Table 7.2  BPD Complete Element Set

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>An event is something that “happens” during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.</td>
<td><img src="image" alt="Event" /></td>
</tr>
<tr>
<td><strong>Flow Dimension</strong> (e.g., Start, Intermediate, End)</td>
<td></td>
<td><img src="image" alt="Flow" /></td>
</tr>
<tr>
<td>Start (None, Message, Timer, Rule, Link, Multiple)</td>
<td>As the name implies, the Start Event indicates where a particular process will start. Intermediate Events occur between a Start Event and an End Event. It will affect the flow of the process, but will not start or (directly) terminate the process.</td>
<td><img src="image" alt="Start" /></td>
</tr>
<tr>
<td>Intermediate (None, Message, Timer, Exception, Cancel, Compensation, Rule, Link, Multiple, Branching)</td>
<td>As the name implies, the End Event indicates where a process will end.</td>
<td><img src="image" alt="Intermediate" /></td>
</tr>
<tr>
<td>End (None, Message, Exception, Cancel, Compensation, Link, Terminate, Multiple)</td>
<td></td>
<td><img src="image" alt="End" /></td>
</tr>
<tr>
<td><strong>Type Dimension</strong> (e.g., Message, Timer, Exception, Cancel, Compensation, Rule, Link, Multiple, Terminate)</td>
<td>Start and Intermediate Events have “Triggers” that define the cause of the event. There are multiple ways that these events can be triggered. End Events may define a “Result” that is a consequence of a Sequence Flow ending.</td>
<td><img src="image" alt="Type" /></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Notation</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Task (Atomic)</td>
<td>A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail.</td>
<td><img src="image" alt="Task" /></td>
</tr>
<tr>
<td>Process/Subprocess</td>
<td>A Subprocess is a compound activity that is included within a Process. It is compound in that it can be broken down into a finer level of detail (a Process) through a set of Subactivities.</td>
<td><img src="image" alt="Subprocess" />See Next Two Figures</td>
</tr>
<tr>
<td>Collapsed Subprocess</td>
<td>The details of the Subprocess are not visible in the diagram. A “plus” sign in the lower-center of the shape indicates that the activity is a Subprocess and has a lower level of detail.</td>
<td><img src="image" alt="Collapsed Subprocess" /></td>
</tr>
<tr>
<td>Expanded Subprocess</td>
<td>The boundary of the Subprocess is expanded and the details (a Process) are visible within its boundary. Note that Sequence Flow cannot cross the boundary of a Subprocess.</td>
<td><img src="image" alt="Expanded Subprocess" /></td>
</tr>
<tr>
<td>Gateway</td>
<td>A Gateway is used to control the divergence and convergence of multiple Sequence Flows. Thus, it determines branching, forking, merging, and joining of paths.</td>
<td><img src="image" alt="Gateway" /></td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
</table>
| Gateway Control Types    | Icons within the diamond shape indicate the type of flow control behavior. The types of control include the following:  
  - XOR – exclusive decision and merging. Both Data-Based and Event-Based. Data-Based can be shown with or without the “X” marker.  
  - OR—inclusive decision  
  - Complex—complex conditions and situations (e.g., 3 out of 5)  
  - AND—forking and joining  
  Each type of control affects both the incoming and outgoing Flow. | ![Gateway Control Types](image) |
| Sequence flow            | A Sequence Flow is used to show the order that activities will be performed in a Process. | ![Sequence Flow](image) |
| Normal flow              | Normal Sequence Flow refers to the flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event. | ![Normal Flow](image) |
| Uncontrolled flow        | Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a Gateway. The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple Sequence Flows that converge on or diverge from an activity. For each uncontrolled Sequence Flow a “Token” will flow from the source object to the target object. | ![Uncontrolled Flow](image) |
Element | Description | Notation
--- | --- | ---
Conditional flow | Sequence Flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be used. If the conditional flow is outgoing from an activity, then the Sequence Flow will have a minidiamond at the beginning of the line (see figure to the right). If the conditional flow is outgoing from a Gateway, then the line will not have a minidiamond (see figure in the preceding row). | ![Minidiamond](image1.png)
Default flow | For Data-Based Exclusive Decisions, one type of flow is the Default condition flow. This flow will be used only if all the other outgoing conditional flows are not true at runtime. These Sequence Flows will have a diagonal slash added to the beginning of the line (see the figure to the right). Note that it is an open issue whether Default Conditions will be used for Inclusive Decision situations. | ![Diagonal Slash](image2.png)
Exception flow | Exception flow occurs outside the normal flow of the Process and is based upon an Intermediate Event that occurs during the performance of the Process. | ![Exception](image3.png)
Message flow | A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the diagram represent the two entities. | ![Message](image4.png)

(continued)
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compensation Association</td>
<td>Compensation Association occurs outside the normal flow of the Process and is based upon an event (a Cancel Intermediate Event) that is triggered through the failure of a Transaction or a Compensate Event. The target of the Association must be marked as a Compensation Activity.</td>
<td></td>
</tr>
<tr>
<td>Data Object</td>
<td>Data Objects are considered artifacts because they do not have any direct effect on the Sequence Flow or Message Flow of the Process, but they do provide information about what the Process does.</td>
<td></td>
</tr>
<tr>
<td>For (AND-Split)</td>
<td>BPMN uses the term “fork” to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split). It is a place in the Process where activities can be performed concurrently, rather than serially. There are two options: Multiple Outgoing Sequence Flow can be used (see figure at top right). This represents “uncontrolled” flow and is the preferred method for most situations. A Parallel (AND) Gateway can be used (see figure at bottom right). This will be used rarely, usually in combination with other Gateways.</td>
<td></td>
</tr>
<tr>
<td>Join (AND-Join)</td>
<td>BPMN uses the term “join” to refer to the combining of two or more parallel paths into one path (also known as an AND-Join or synchronization). A Parallel (AND) Gateway is used to show the joining of multiple flows.</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Notation</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>Decision, Branching Point; (OR-Split)</td>
<td>Decisions are Gateways within a business process where the flow of control can take one or more alternative paths.</td>
<td>See next five rows.</td>
</tr>
<tr>
<td>Exclusive</td>
<td>An Exclusive Gateway (XOR) restricts the flow such that only one of a set of alternatives may be chosen during runtime. There are two types of Exclusive Gateways: Data-Based and Event-Based.</td>
<td><img src="data-based-exclusive-gateway.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Data-Based</td>
<td>This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow. Only one of the Alternatives will be chosen.</td>
<td><img src="data-based-exclusive-gateway.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Event-Based</td>
<td>This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process. The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the Alternatives will be chosen. There are two options for receiving Messages. Tasks of Type Receive can be used (see figure at top right). Intermediate Events of Type Message can be used (see figure bottom-right).</td>
<td><img src="event-based-exclusive-gateway.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

(continued)
### Table 7.2 BPD Complete Element Set (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inclusive</td>
<td>This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow. In a sense it is a grouping of related independent Binary (Yes/No) Decisions. Because each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. There are two versions of this type of Decision. The first uses a collection of conditional Sequence Flow, marked with minidiamonds (see top-right figure). The second uses an OR Gateway, usually in combination with other Gateways (see bottom-right picture).</td>
<td></td>
</tr>
<tr>
<td>Merging (OR-Join)</td>
<td>BPMN uses the term merge to refer to the exclusive combining of two or more paths into one path (also known as an a OR-Join). A Merging (XOR) Gateway is used to show the merging of multiple flows. If all the incoming flow is alternative, then a Gateway is not needed; that is, uncontrolled flow provides the same behavior.</td>
<td></td>
</tr>
<tr>
<td>Looping</td>
<td>BPMN provides two mechanisms for looping within a Process.</td>
<td>See Next Two Figures</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Notation</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Activity Looping</td>
<td>The properties of Tasks and Subprocesses will determine if they are repeated or performed once. There are two types of loops: Standard and Multi-Instance. A small looping indicator will be displayed at the bottom center of the activity.</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>Sequence Flow Looping</td>
<td>Loops can be created by connecting a Sequence Flow to an “upstream” object. An object is considered to be upstream if it has an outgoing Sequence Flow that leads to a series of other Sequence Flows, the last of which is an incoming Sequence Flow for the original object.</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>Multiple Instances</td>
<td>The attributes of Tasks and Subprocesses will determine if they are repeated or performed once. A small parallel indicator will be displayed at the bottom center of the activity.</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>Process Break (something out of the control of the process makes it pause)</td>
<td>A Process Break is a location in the Process that shows where an expected delay will occur within it. An Intermediate Event is used to show the actual behavior (see top-right figure). In addition, a Process Break artifact, as designed by a modeler or modeling tool, can be associated with the Event to highlight the location of the delay within the flow.</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction</td>
<td>A transaction is an activity, either a Task or a Subprocess, that is supported by a special protocol that ensures that all parties involved have complete agreement that the activity should be completed or cancelled. The attributes of the activity will determine if the activity is a transaction. A double-lined boundary indicates that the activity is a Transaction.</td>
<td></td>
</tr>
<tr>
<td>Nested Subprocess (Inline Block)</td>
<td>A nested Subprocess is an activity that shares the same set of data as its parent process. This is opposed to a Subprocess that is independent, reusable, and referenced from the parent process. Data needs to be passed to the referenced Subprocess, but not to the nested Subprocess.</td>
<td>There is no special indicator for nested Subprocesses</td>
</tr>
<tr>
<td>Group (a box around a group of objects for documentation purposes)</td>
<td>A grouping of activities that does not affect the Sequence Flow. The grouping can be used for documentation or analysis purposes. Groups can also be used to identify the activities of a distributed transaction that is shown across Pools.</td>
<td></td>
</tr>
<tr>
<td>Off-Page Connector</td>
<td>Generally used for printing, this object will show where the Sequence Flow leaves one page and then restarts on the next. A Link Intermediate Event can be used as an Off-Page Connector.</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Notation</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Association</td>
<td>An Association is used to associate information with flow objects. Text and graphical nonflow objects can be associated with the flow objects.</td>
<td></td>
</tr>
<tr>
<td>Text Annotation (attached with an Association)</td>
<td>Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN Diagram.</td>
<td></td>
</tr>
<tr>
<td>Pool</td>
<td>A Pool is a “swimlane” and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.</td>
<td></td>
</tr>
<tr>
<td>Lanes</td>
<td>A Lane is a subpartition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities within a Pool.</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3  Sequence Flow Connection Rules
Table 7.4 Message Flow Connection Rules

<table>
<thead>
<tr>
<th>From/To</th>
<th>Name (Pool)</th>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.5 An example of a transaction expanded Subprocess.
Figure 7.6 Message Flow connecting to flow objects within two Pools.

Figure 7.7 A Group around activities in different Pools.
Figure 7.8  Message Flow connecting to boundary of Subprocess and Internal objects.

Figure 7.9  A Process with Expanded Subprocess without a Start Event and End Event.
7.2.3 Business Process Query Language

The Business Process Query Language (BPQL) defines a standard interface to Business Process Management Systems (BPMSs). It allows system administrators to manage the BPMS and business analysts to query the instances of business processes it executes. BPQL is a management interface to a business process management infrastructure that includes a process execution facility (Process Server) and a process deployment facility (Process Repository). The BPQL interface to a Process Server enables business analysts to query the state and control the execution of process instances managed by the Process Server. This interface is based on the Simple Object Access Protocol (SOAP).

The BPQL interface to a Process Repository enables business analysts to manage the deployment of process models managed by the repository. This interface is based on the Distributed Authoring and Versioning Protocol (WebDAV). Process models managed by the Process Repository through the BPQL interface can be exposed as Universal Description, Discovery, and Integration (UDDI) services for process registration, advertising, and discovery purposes.

7.3 Unified Modeling Language™

The Unified Modeling Language™ (in Version 2.0 at press time) is OMG’s most-used modeling specification. UML allows users to model the business process, application structure, application behavior, data structure, and architecture. Hence, UML can be utilized for the top three layers of the architecture, as seen in Figure 7.2. However, we discussed it here mostly in the context of the business process. UML, along with the Meta Object Facility (MOF™), also provides a foundation for OMG’s Model-Driven Architecture, which unifies steps of development and integration from business modeling, through architectural and application modeling, to development, deployment, maintenance, and evolution. OMG is a not-for-profit computer industry specification consortium;
members define and maintain the various specifications discussed in this section (available for free). Software providers have built tools that conform to these specifications. Some of the key milestones of OMG accomplishments in the past few years include the following:

- 1989: OMG Established
- Standardization of Distributed Object Middleware
  - 1995: CORBA2
  - 2001: CORBA2.5
- Modeling Standardization
  - 1997: UML (Unified Modeling Language)
  - 1997: MOF
  - 1999: XMI
  - 2000: CWM
  - 2001: Application-specific UML Profiles (EDOC, EAI)
  - 2005: UML V2
- Architecture (Reference Model)
  - 1990: OMA (Object Management Architecture)
  - 2001: MDA (Model Driven Architecture)

UML is a visual language for specifying, constructing, and documenting the artifacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, .NET). The OMG adopted the UML 1.1 specification in 1997; since then UML Revision Task Forces have produced several minor revisions. Under the stewardship of the OMG, UML has emerged as the software industry’s dominant modeling language; it has been successfully applied to a wide range of domains, ranging from health and finance to aerospace to E-commerce. As should be expected, its extensive use has raised application and implementation issues by modelers and vendors. Consequently, in the early 2000s the OMG has sought to develop a new version, namely, UML 2.0. Version 2.0 started to be available in the 2004/5 timeframe and is now being rolled into other OMG constructs. With UML 2.0, both developers and tool vendors can use common semantics through integrated tools. UML 2.0 also defines compliance levels.

The observations that follow in this entire subsection are based on OMG information on UML and related capabilities [OMG200501].

7.3.1 Overview

UML helps firms specify, visualize, and document models of software systems, including their structure and design. (Firms can use UML for business modeling and modeling of other nonsoftware systems too.) Utilizing any one of the large number of UML-based tools on the market, firms can analyze their application's requirements and design a solution that meets these requirements while representing the results using UML's 12 standard diagram types.

Within UML one can model most types of applications, running on any type and combination of hardware, operating system, programming language, and network. UML’s flexibility allows firms to model distributed applications that use any middleware on the market. Built upon the MOF metamodel, which defines class and operation as fundamental concepts, UML is a good
fit for object-oriented (OO) languages and environments such as C++, Java, and C#; one can use it to model non-OO applications as well, for example, Fortran, VB, or COBOL. UML Profiles (that is, subsets of UML tailored for specific purposes) help the developers model transactional, real-time, and fault-tolerant systems in a natural way. UML profiles are subsets of UML tailored to specific environments. For example, OMG has defined a profile for EDOC that is especially good at modeling collaborations, and a profile for Enterprise Application Integration (EAI), specialized for applications based on asynchronous communication [SEI200201].

The process of gathering and analyzing an application’s requirements and incorporating them into a program design is a complex one. The industry currently supports many methodologies that define formal procedures specifying how to go about it. One characteristic of UML is that it is methodology independent. Regardless of the methodology that one uses to perform the analysis and design, architects can use UML to express the results; using XML Metadata Interchange (XMI), another OMG standard, one can transfer a UML model from one tool into a repository, or into another tool for refinement or the next step in the firm’s chosen development process.

As noted, in the early 2000s OMG members started work on a major upgrade of UML (the most current version prior to the current UML v2.0 was UML v1.4). Four separate documents were defined as part of the upgrade: (1) UML Infrastructure, (2) UML Superstructure, (3) Object Constraint Language (OCL), and (4) UML Diagram Interchange. Adoption of the UML 2.0 Superstructure is complete (the Superstructure specification has been stable since it took its adopted form in 2004); adoption of the other three parts of UML 2.0 (also called UML2) was nearly complete at press time.

- The first specification, **UML 2.0: Infrastructure**, serves as the architectural foundation. It defines the foundational language constructs required for UML 2.0; specifically, it defines base classes that form the foundation not only for the UML 2.0 superstructure, but also for MOF 2.0.
- **UML 2.0: Superstructure** is the second of two complementary specifications. The superstructure defines the user-level constructs required for UML 2.0. It defines structure diagrams, behavior diagrams, interaction diagrams, and the elements that comprise them. The two complementary specifications constitute a complete specification for the UML 2.0 modeling language.
  (Note that as the two volumes cross-reference each other and the specifications are fully integrated, these two volumes could easily be combined into a single volume at a later time.)
- The **UML 2.0 OCL** allows setting of pre- and post-conditions, invariants, and other conditions.
- The **UML 2.0 Diagram Interchange** is a specification that extends the UML metamodel with a supplementary package for graph-oriented information, allowing models to be exchanged or stored/retrieved and then displayed as they were originally.

UML defines twelve types of diagrams, divided into three categories: four diagram types represent static application structure; five represent different aspects of dynamic behavior; and three represent ways one can organize and manage application modules. These are as follows:

- **Structural Diagrams** include the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.
Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram.

Model Management Diagrams include Packages, Subsystems, and Models.

There are many UML tools on the market that allow architects to make practical use of the UML family of specifications. Some UML tools analyze existing source code and reverse-engineer it into a set of UML diagrams. UML has a focus on design rather than execution; however, some tools on the market execute UML models, typically in one of two ways:

1. Some tools execute a model interpretively in a way that lets the user confirm that it really does what the user wants, but without the scalability and speed that will be needed in the deployed application.

2. Other tools (typically designed to work only within a restricted application domain such as telecommunications or finance) generate program language code from UML, producing a bug-free, deployable application that runs quickly if the code generator incorporates best-practice scalable patterns for, e.g., transactional database operations or other common program tasks.

A number of tools on the market generate test and verification suites from UML models.

Middleware is important in software development, particularly in the context of enterprise application integration. According to observers, a few years ago the major problem a developer faced when starting a distributed programming project was finding a middleware with the needed functionality, that ran on the hardware and operating systems deployed at the firm. Today, faced with an array of middleware platforms, the developer has a number of different challenges: (1) selecting a specific middleware; (2) getting the middleware to work with the other platforms already deployed not only in his or her own shop, but also those of his customers and suppliers; and (3) interfacing or migrating to new environments or applications.

Keeping in mind the importance of middleware, it is worth noting that by design, UML is middleware-independent. A UML model can be either platform-independent or platform-specific, as one chooses. The Model-Driven Architecture development process, which UML supports, uses both of these forms. Every MDA standard or application is based, normatively, on a Platform-Independent Model (PIM); the PIM represents its business functionality and behavior very precisely but does not include technical aspects. From the PIM, MDA-enabled development tools follow OMG-standardized mappings to produce one or more Platform-Specific Models (PSMs), also in UML, one for each target platform that the developer chooses. (This conversion step is automated to a large degree: before the tool produces a PSM, the developer must annotate the base PIM to produce a more specific but still platform-independent PIM that includes details of desired semantics, and guides choices that the tool will have to make. Because of the similarities among middleware platforms of a given genre—component based or messaging based, for example—this guidance can be included in a PIM without rendering it platform specific. Still, developers have to fine-tune the produced PSMs to some extent, more in the early days of MDA but less and less as tools and algorithms advance.) The PSM contains the same information as an implementation, but in the form of a UML model instead of executable code. In the next step, the tool generates the running code from the PSM, along with other necessary files (including interface definition files if necessary, configuration files, makefiles, and other file types). After giving the developer an opportunity to hand-tune the generated code, the tool executes the makefiles to produce a deployable final application. MDA applications are composable: if one imports PIMs for modules,
services, or other MDA applications into one’s development tool, one can direct it to generate calls using whatever interfaces and protocols are required, even if these run cross-platform. MDA is revisited in Section 7.4.

At a macro level, developers may proceed as follows:

1. **Select a methodology**: A methodology formally defines the process that one uses to gather requirements, analyze them, and design an application that meets them in every way. There are many methodologies, each differing in some way or ways from the others. There are many reasons why one methodology may be better than another for one’s particular project: for example, some are better suited for large enterprise applications, whereas others are built to design small embedded or safety-critical systems. Looking at other considerations, some methods better support large numbers of architects and designers working on the same project, whereas others work better when used by one person or a small group.

2. **Select a UML development tool**: Because most (although not all) UML-based tools implement a particular methodology, in some cases it might not be practical to select a tool and then try to use it with a methodology that it was not built for. (For other tool/methodology combinations, this might not be an issue, or might be easy to work around.) However, some methodologies have been implemented on multiple tools, so this is not strictly a one-choice environment.

### 7.3.2 Scratching the Surface of UML

This section provides a brief foray into UML. It is based on the *UML 2.0: Superstructure* [UML200501] and on the *UML 2.0: Infrastructure* [UMI200401]. The reader should consult these references for a complete coverage of the topic.

#### 7.3.2.1 Conformance

As noted earlier, UML is a language with a broad scope that covers a diverse set of application domains. Not all of its modeling capabilities are necessarily useful in all domains or applications. This suggests that the language should be, and in fact is, structured modularly, with the ability to select only those parts of the language that are of direct interest. On the other hand, an excess of this type of flexibility increases the likelihood that two different UML tools will be supporting different subsets of the language, leading to interchange problems between them. Consequently, the definition of compliance for UML requires a balance to be drawn between modularity and ease of interchange. Experience has indicated that the ability to exchange models between tools is of paramount interest to a large community of users. For that reason, the UML specification defines a number of *compliance levels*, thereby increasing the likelihood that two or more compliant tools will support the same or compatible language subsets. However, in recognition of the need for flexibility in learning and using the language, UML also provides the concept of *language units*.

#### 7.3.2.1.1 Language Units

The modeling concepts of UML are grouped into *language units*. A language unit consists of a collection of tightly coupled modeling concepts that provide users with the ability to represent aspects of the system under study according to a particular paradigm or formalism. For example,
the State Machines’ language unit enables modelers to specify discrete event-driven behavior using a variant of the well-known statecharts’ formalism, whereas the Activities’ language unit provides for modeling behavior based on a workflow-like paradigm. From the user’s perspective, this partitioning of UML means that they need only be concerned with those parts of the language that they consider necessary for their models. If those needs change over time, further language units can be added to the user’s repertoire as required. Hence, a UML user does not have to know the full language to use it effectively. In addition, most language units are partitioned into multiple increments, each adding more modeling capabilities to the previous ones. This fine-grained decomposition of UML serves to make the language easier to learn and use, but the individual segments within this structure do not represent separate compliance points. The latter strategy would lead to an excess of compliance points and result in the interoperability problems described earlier. Nevertheless, the groupings provided by language units and their increments do serve to simplify the definition of UML compliance.

7.3.2.1.2 Compliance Levels

The stratification of language units is used as the foundation for defining compliance in UML. The set of modeling concepts of UML is partitioned into horizontal layers of increasing capability called compliance levels. Compliance levels cut across the various language units, although some language units are only present in the upper levels. As their name suggests, each compliance level is a distinct compliance point. For ease of model interchange, there are just four compliance levels defined for the whole of UML:

- **Level 0 (L0):** This compliance level is formally defined in the UML Infrastructure. It contains a single language unit that provides for modeling the kinds of class-based structures encountered in most popular object-oriented programming languages. As such, it provides an entry-level modeling capability. More importantly, it represents a low-cost common denominator that can serve as a basis for interoperability between different categories of modeling tools.

- **Level 1 (L1):** This level adds new language units and extends the capabilities provided by Level 0. Specifically, it adds language units for use cases, interactions, structures, actions, and activities.

- **Level 2 (L2):** This level extends the language units already provided in Level 1 and adds language units for deployment, state machine modeling, and profiles.

- **Level 3 (L3):** This level represents the complete UML. It extends the language units provided by Level 2 and adds new language units for modeling information flows, templates, and model packaging.

The contents of language units are defined by corresponding top-tier packages of the UML metamodel, whereas the contents of their various increments are defined by second-tier packages within language unit packages. Therefore, the contents of a compliance level are defined by the set of metamodel packages that belong to that level.

As noted previously, compliance levels build on supporting compliance levels. The principal mechanism used in the UML specification for achieving this is package merge. Package merge allows modeling concepts defined at one level to be extended with new features. Most importantly, this is achieved in the context of the same namespace, which enables interchange of models at different levels of compliance. For this reason, all compliance levels are defined as extensions to a
single core UML package that defines the common namespace shared by all the compliance levels (see Table 7.5 for an example).

Level 0 is defined by the top-level metamodel shown in Figure 7.11. In this model, UML is originally an empty package that simply merges in the contents of the Basic package from the UML Infrastructure. This package contains elementary concepts such as Class, Package, DataType, Operation, etc. (see the UML 2.0 Infrastructure specification for the complete list of contents).

At the next level (Level 1), the contents of the UML package, now including the packages merged into Level 0 and their contents, are extended with additional packages as shown in Figure 7.12. Note that each of the four packages shown in the figure merges in additional packages that are not shown in the diagram. They are defined in the corresponding package diagrams in this specification. Consequently, the set of language units that results from this model is more numerous than indicated by the top-level model in the diagram. The specific packages included at this level are listed in Table 7.6.

### Table 7.5 Example Compliance Statement

<table>
<thead>
<tr>
<th>Compliance level</th>
<th>Abstract Syntax</th>
<th>Concrete Syntax</th>
<th>Diagram</th>
<th>Interchange Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Level 1</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Level 2</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

![Figure 7.11] Level 0 package diagram.

![Figure 7.12] Level 1 top-level package merges.
Level 2 adds further language units and extensions to those provided by Level 1. Once again, the package UML now incorporates the complete Level 1 shown in Figure 7.13. The actual language units and packages included at this level of compliance are listed in Table 7.7.

Finally, Level 3, incorporating the full UML definition, is shown in Figure 7.14. Its contents are described in Table 7.8.

### 7.3.2.1.3 Meaning and Types of Compliance

Compliance to a given level entails full realization of all language units that are defined for that compliance level. This also implies full realization of all language units in all the levels below that level. “Full realization” for a language unit at a given level means supporting the complete set of

#### Table 7.6 Metamodel Packages Added in Level 1

<table>
<thead>
<tr>
<th>Language Unit</th>
<th>Metamodel Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions</td>
<td>Actions::BasicActions</td>
</tr>
<tr>
<td>Activities</td>
<td>Activities::FundamentalActivities Activities::BasicActivities</td>
</tr>
<tr>
<td>Classes</td>
<td>Classes::Kernel Classes::Dependencies Classes::Interfaces</td>
</tr>
<tr>
<td>General Behavior</td>
<td>CommonBehaviors::BasicBehaviors</td>
</tr>
<tr>
<td>Structures</td>
<td>CompositeStructure::InternalStructures</td>
</tr>
<tr>
<td>Interactions</td>
<td>Interactions::BasicInteractions</td>
</tr>
<tr>
<td>UseCases</td>
<td>UseCases</td>
</tr>
</tbody>
</table>

![Figure 7.13 Level 2 top-level package merges.](image-url)
modeling concepts defined for that language unit at that level. Thus, it is not meaningful to claim compliance with, say, Level 2, without also being compliant with the Level 0 and Level 1. A tool that is compliant at a given level must be able to import models from tools that are compliant with lower levels without loss of information.

### Table 7.7 Metamodel Packages Added in Level 2

<table>
<thead>
<tr>
<th>Language Unit</th>
<th>Metamodel Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions</td>
<td>Actions::StructuredActions&lt;br&gt;Actions::IntermediateActions</td>
</tr>
<tr>
<td>Activities</td>
<td>Activities::IntermediateActivities&lt;br&gt;Activities::StructuredActivities</td>
</tr>
<tr>
<td>Components</td>
<td>Components::BasicComponents</td>
</tr>
<tr>
<td>Deployments</td>
<td>Deployments::Artifacts&lt;br&gt;Deployments::Nodes</td>
</tr>
<tr>
<td>General Behavior</td>
<td>CommonBehaviors::Communications&lt;br&gt;CommonBehaviors::SimpleTime</td>
</tr>
<tr>
<td>Interactions</td>
<td>Interactions::Fragments</td>
</tr>
<tr>
<td>Profiles</td>
<td>AuxiliaryConstructs::Profiles</td>
</tr>
<tr>
<td>Structures</td>
<td>CompositeStructures::InvocationActions&lt;br&gt;CompositeStructures::Ports&lt;br&gt;CompositeStructures::StructuredClasses</td>
</tr>
<tr>
<td>State Machines</td>
<td>StateMachines::BehaviorStateMachines</td>
</tr>
</tbody>
</table>

![Figure 7.14 Level 3 top-level package merges.](image)
There are two types of compliance:

- **Abstract syntax compliance**: For a given compliance level, this entails
  - Compliance with the metaclasses, their structural relationships, and any constraints defined as part of the merged UML metamodel for that compliance level
  - The ability to output models and to read in models based on the XMI schema corresponding to that compliance level

- **Concrete syntax compliance**: For a given compliance level, this entails
  - Compliance with the notation defined in the “Notation” sections in this specification for those metamodel elements that are defined as part of the merged metamodel for that compliance level and, by implication, the diagram types in which those elements may appear.
  - Optionally, the ability to output diagrams and to read in diagrams based on the XMI schema defined by the Diagram Interchange specification for notation at that level. This option requires abstract syntax and concrete syntax compliance.

Concrete syntax compliance does not require compliance with any presentation options that are defined as part of the notation. Compliance for a given level can be expressed as follows:

- Abstract syntax compliance
- Concrete syntax compliance
- Abstract syntax with concrete syntax compliance
- Abstract syntax with concrete syntax and diagram interchange compliance

In the case of tools that generate program code from models or those that are capable of executing models, it is also useful to understand the level of support for the runtime semantics described in the various “Semantics” subsections of the specification. However, the presence of numerous variation points in these semantics (and the fact that they are defined informally using natural
language) make it impractical to define this as a formal compliance type, because the number of possible combinations is very large. A similar situation exists with presentation options, because different implementers may make different choices on which ones to support. Finally, it is recognized that some implementers and profile designers may want to support only a subset of features from levels that are above their formal compliance level. (Note, however, that they can only claim compliance with the level that they fully support, even if they implement significant parts of the capabilities of higher levels.) Given this potential variability, it is useful to be able to specify clearly and efficiently which capabilities are supported by a given implementation. To this end, in addition to a formal statement of compliance, implementers and profile designers may also provide informal feature support statements. These statements identify support for additional features in terms of language units or individual metamodel packages, as well as for less precisely defined dimensions such as presentation options and semantic variation points.

7.3.2.2 Runtime Semantics of UML

It is useful to have a high-level view of the runtime semantics of UML. The term runtime is used to refer to the execution environment. Runtime semantics, therefore, are specified as a mapping of modeling concepts into corresponding program execution phenomena (there are other semantics relevant to UML specifications, such as the repository semantics, that is, how a UML model behaves in a model repository; however, those semantics are really part of the definition of the MOF).

7.3.2.2.1 The Basic Premises

There are two fundamental premises regarding the nature of UML semantics. The first is the assumption that all behavior in a modeled system is ultimately caused by actions executed by so-called active objects. This includes behaviors, which are objects in UML 2.0, that can be active and coordinate other behaviors. The second is that UML behavioral semantics only deal with event-driven, or discrete, behaviors. However, UML does not specify the amount of time between events, which can be as small as needed by the application, for example, when simulating continuous behaviors.

7.3.2.2.2 The Semantics Architecture

Figure 7.14 identifies the key semantic areas covered by the current standard and how they relate to one another. The items in the upper layers depend on the items in the lower layers, but not the other way around. (Note that the structure of metamodel package dependencies is somewhat similar to the dependency structure indicated here; however, they are not the same and should be distinguished—this is because package dependencies specify repository dependencies, not necessarily runtime dependencies.)

At the highest level of abstraction, it is possible to distinguish three distinct composite layers of semantic definitions. The foundational layer is structural. This reflects the premise that there is no disembodied behavior in UML—all behavior is the consequence of the actions of structural entities. The next layer is behavioral and provides the foundation for the semantic description of all the higher-level behavioral formalisms (the term behavioral formalism refers to a formalized framework for describing behavior, such as state machines, Petri nets, data flow graphs, etc.). This layer, represented by the shaded box in Figure 7.15, is the behavioral semantic base and consists of
three separate subareas arranged into two sublayers. The bottom sublayer consists of the *interobject behavior base*, which deals with how structural entities communicate with one another, and the *intraobject behavior base*, which addresses the behavior occurring within structural entities. The *actions* sublayer is placed on top of these two. It defines the semantics of individual actions. Actions are the fundamental units of behavior in UML and are used to define fine-grained behaviors. Their resolution and expressive power are comparable to the executable instructions in traditional programming languages. Actions in this sublayer are available to any of the higher-level formalisms to be used for describing detailed behaviors. The topmost layer in the semantics hierarchy defines the semantics of the higher-level behavioral formalisms of UML: *activities*, *state machines*, and *interactions*. Other behavioral formalisms may be added to this layer in the future.

7.3.2.2.3 The Basic Causality Model

The “causality model” is a specification of how things happen at runtime. It is briefly summarized here using the example depicted in the communication diagram in Figure 7.16. The example shows two independent, and possibly concurrent, threads of causally chained interactions. The first, identified by the thread prefix “A,” consists of a sequence of events that commences with activeObject-1 sending signal s1 to activeObject-2. In turn, activeObject-2 responds by invoking operation op1( ) on passiveObject-1 after which it sends signal s2 to activeObject-3. The second thread, distinguished by the thread prefix “B,” starts with activeObject-4 invoking operation op2( ) on passiveObject-1. The latter responds by executing the method that realizes this operation, in which it sends signal s3 to activeObject-2. The causality model is quite straightforward: Objects respond to messages that are generated by objects executing communication actions. When these messages arrive, the receiving objects eventually respond by executing the behavior that is matched to that message. The dispatching method by which a particular behavior is associated with a given message depends on the higher-level formalism used and is not defined in the UML specification (i.e., it is a semantic variation point).

The causality model also subsumes behaviors invoking one another and passing information to one another through arguments to parameters of the invoked behavior, as enabled by CallBehaviorAction. This purely “procedural,” or “process,” model can be used by itself or in conjunction with the object-oriented model of the previous example.
7.3.2.3 The UML Metamodel

7.3.2.3.1 Models and What They Model

A model contains three major categories of elements: classifiers, events, and behaviors. Each major category models individuals in an incarnation of the system being modeled. A classifier describes a set of objects; an object is an individual thing with a state and relationships to other objects. An event describes a set of possible occurrences; an occurrence is something that happens that has some consequence within the system. A behavior describes a set of possible executions; an execution is the performance of an algorithm according to a set of rules. Models do not contain objects, occurrences, and executions, because they are the subject of models, not their content. Classes, events, and behaviors model sets of objects, occurrences, and executions with similar properties. Value specifications, occurrence specifications, and execution specifications model individual objects, occurrences, and executions, because they are the subject of models, not their content. Classes, events, and behaviors model sets of objects, occurrences, and executions with similar properties. Value specifications, occurrence specifications, and execution specifications model individual objects, occurrences, and executions within a particular context. The distinction between objects and models of objects, for example, may appear subtle, but it is important. Objects (and occurrences and executions) are the domain of a model and, as such, are always complete, precise, and concrete. Models of objects (such as value specifications) can be incomplete, imprecise, and abstract according to their purpose in the model.

7.3.2.3.2 Semantic Levels and Naming

A large number of UML metaclasses can be arranged into four levels with metasemantic relationships among the metaclasses in the different levels that transcend different semantic categories (e.g., classifiers, events, and behaviors). The specification attempts to provide a consistent naming pattern across the various categories to place elements into levels and emphasize metarelationships among related elements in different levels. The following four levels are important:

Type level—Represents generic types of entities in models, such as classes, states, activities, events, etc. These are the most common constituents of models because models are primarily about making generic specifications.

Instance level—These are the things that models represent at runtime. They do not appear in models directly (except very occasionally as detailed examples), but they are necessary to explain the semantics of what models mean. These classes do not appear at all in the UML2 metamodel or in UML models, but they underlie the meaning of models.
Value specifications—A realization of UML2, compared to UML, is that values can be specified at various levels of precision. The specification of a value is not necessarily an instance; it might be a large set of possible instances consistent with certain conditions. What appears in models is usually not instances (individual values) but specifications of values that may or may not be limited to a single value. In any case, models contain specifications of values, not values themselves, which are runtime entities.

Individual appearances of a type within a context—These are roles within a generic, reusable context. When their context is instantiated, they are also bound to contained instances, but as model elements they are reusable structural parts of their context; they are not instances themselves. A realization of UML2 was that the things called instances in UML1 were mostly roles: they map to instances in an instance of their container, but they are model elements, not instances, because they are generic and can be used many times to generate many different instances.

The specification establishes the following naming patterns:

- **Types:** Classifier, Class, Instance, Object
- **Instances:** InstanceSpecification, Part, Role, Attribute
- **Values:** XXXUse (e.g., CollaborationUse)
- **Uses:** Various (e.g., Trigger)
- **Behaviors:** Execution: ExecutionSpecification: various (e.g., ActivityNode, State)
- **Uses:** XXXUse (e.g., InteractionUse)

The appearances category has too wide a variety of elements to reduce to a single pattern, although the form XXXUse is suggested for simple cases in which an appearance of an element is contained in a definition of the same kind of element. In particular, the word *event* has been used inconsistently in the past to mean both type and instance. The word *event* now means the type, and the word *occurrence* means the instance. When necessary, the phrases *event type* (for event) and *event occurrence* (for occurrence) may be used. Note that this is consistent with the frequent English usage *an event occurs* = the occurrence of an event of a given type; so, to describe a runtime situation, one could say “event X occurs” or “an occurrence of event X,” depending on which form is more convenient in a sentence. It is redundant and incorrect to say “an event occurrence occurs.”

7.3.2.4 UML Infrastructure Specification

The observations that follow in this entire subsection are based on OMG information [UMI200401]

7.3.2.4.1 Language Architecture

The UML specification is defined using a metamodeling approach (i.e., a metamodel is used to specify the model that comprises UML) that adapts formal specification techniques. Although this approach lacks some of the rigor of a formal specification method, it offers the advantages of being more intuitive and pragmatic for most implementers and practitioners.
7.3.2.4.1.1 Design Principles
The UML metamodel has been architected with the following design principles in mind:

- **Modularity**—This principle of strong cohesion and loose coupling is applied to group constructs into packages and organize features into metaclasses.
- **Layering**—Layering is applied in two ways to the UML metamodel. First, the package structure is layered to separate the metalanguage core constructs from the higher-level constructs that use them. Second, a four-layer metamodel architectural pattern is consistently applied to separate concerns (especially regarding instantiation) across layers of abstraction. These layers are referred to as M3, M2, M1, and M0.
- **Partitioning**—Partitioning is used to organize conceptual areas within the same layer. In the case of the InfrastructureLibrary, fine-grained partitioning is used to provide the flexibility required by current and future metamodeling standards. In the case of the UML metamodel, the partitioning is coarser-grained to increase the cohesion within packages and relaxing the coupling across packages.
- **Extensibility**—The UML can be extended in two ways:
  1. A new dialect of UML can be defined by using Profiles to customize the language for particular platforms (e.g., J2EE/EJB, .NET/COM+) and domains (e.g., finance, telecommunications, aerospace).
  2. A new language related to UML can be specified by reusing part of the InfrastructureLibrary package and augmenting with appropriate metaclasses and metarelationships. The former case defines a new dialect of UML, whereas the latter case defines a new member of the UML family of languages.
- **Reuse**—A fine-grained, flexible metamodel library is provided that is reused to define the UML metamodel, as well as other architecturally related metamodels, such as the MOF and the Common Warehouse Model (CWM). (CWM is a metamodel that enables data mining across database boundaries; it is discussed in Section 7.4.2.4.)

7.3.2.4.1.2 Infrastructure Architecture
The Infrastructure of the UML is defined by the package *InfrastructureLibrary*, which satisfies the following design requirements:

- Define a metalanguage core that can be reused to define a variety of metamodels, including UML, MOF, and CWM.
- Architecturally align UML, MOF, and XMI so that model interchange is fully supported.
- Allow customization of UML through Profiles and creation of new languages (family of languages) based on the same metalanguage core as UML.

As shown in Figure 7.17, Infrastructure is represented by the package *InfrastructureLibrary*, which consists of the packages *Core* and *Profiles*, the latter defining the mechanisms that are used to customize metamodels and the former containing core concepts used when metamodeling.

7.3.2.4.1.2.1 The Core Package
In its first capacity, the *Core* package is a complete metamodel specifically designed for high reusability, where other metamodels at the same metalevel either import or specialize its specified metaclasses. This is illustrated in Figure 7.18, where it is shown how UML, CWM, and MOF each depends on
a common core. Because these metamodels are at the core of MDA, the common core may also be considered the architectural kernel of MDA. The intent is for UML and other MDA metamodels to reuse all or parts of the Core package, which allows other metamodels to benefit from the abstract syntax and semantics that have already been defined.

To facilitate reuse, the Core package is subdivided into a number of packages: PrimitiveTypes, Abstractions, Basic, and Constructs, as shown in Figure 7.19. Some of these packages are then further divided into even more fine-grained packages to make it possible to pick and choose the relevant parts when defining a new metamodel. Note, however, that choosing a specific package also implies choosing the dependent packages. The package PrimitiveTypes simply contains a few predefined types that are commonly used when metamodeling, and is designed specifically with the needs of UML and MOF in mind. Other metamodels may need other or overlapping sets of primitive types. There are minor differences in the design rationale for the other two packages. The package Abstractions mostly contains abstract metaclasses that are intended to be further specialized or that are expected to be commonly reused by many metamodels. Very few assumptions are made about the metamodels that may want to reuse this package; for this reason, the package Abstractions is also subdivided into several smaller packages. The package Constructs, on the other hand, mostly contains concrete metaclasses that lend themselves primarily to object-oriented modeling; this package, in particular, is reused by both MOF and UML, and represents a significant part of the work that has gone into aligning the two metamodels. The package Basic represents a few constructs that are used as the basis for the produced XMI for UML, MOF, and other metamodels based on the InfrastructureLibrary.

In its second capacity, the Core package is used to define the modeling constructs used to create metamodels. This is done through instantiation of metaclasses in the InfrastructureLi-
Whereas instantiation of metaclasses is carried out through MOF, the InfrastructureLibrary defines the actual metaclasses that are used to instantiate the elements of UML, MOF, CWM and, indeed, the elements of the InfrastructureLibrary itself. In this respect, the InfrastructureLibrary is said to be self-describing, or reflective.

7.3.2.4.1.2.2 The Profiles Package
As was depicted in Figure 7.17, the Profiles package depends on the Core package, and defines the mechanisms used to tailor existing metamodels toward specific platforms such as C++, Common Object Request Broker Architecture (CORBA), or Enterprise JavaBeans (EJBs), or domains, such as real-time, business objects, or software process modeling. The primary target for profiles is UML, but it is possible to use profiles together with any metamodel that is based on (i.e., instantiated from) the common core. A profile must be based on a metamodel such as the UML that it extends, and is not very useful stand-alone. Profiles have been aligned with the extension mechanism offered by MOF, but provide a more lightweight approach with restrictions that are enforced to ensure that the implementation and usage of profiles are straightforward and more easily supported by tool vendors.

7.3.2.4.1.2.3 Architectural Alignment between UML and MOF
One of the goals of the Infrastructure Specification has been to architecturally align UML and MOF. The first approach to accomplish this has been to define the common core, which is realized as the package Core, in such a way that the model elements are shared between UML and MOF. The second approach has been to make sure that UML is defined as a model that is based on MOF used as a metamodel, as illustrated in Figure 7.20. Note that MOF is used as the metamodel for not only UML, but also for other languages such as CWM.

As was depicted in Figure 7.18, the model elements of UML are shared between UML and MOF, respectively, as was shown in Figure 7.18. In the case of MOF, the metaclasses of the InfrastructureLibrary are used as is, whereas in the case of UML these model elements are given additional properties. The reason for these differences is that the requirements when metamodeling differ slightly from the requirements when modeling applications of a very diverse nature.

MOF defines, for example, how UML models are interchanged between tools using XMI. MOF also defines reflective interfaces (MOF::Reflection) for introspection that work for MOF itself, but also for CWM, UML, and any other metamodel that is an instance of MOF. It further defines an
extension mechanism that can be used to extend metamodels as an alternative to, or in conjunction with, profiles. In fact, profiles are defined to be a subset of the MOF extension mechanism.

7.3.2.4.1.2.4 Superstructure Architecture
The UML Superstructure metamodel is specified by the UML package, which is divided into a number of packages that deal with structural and behavioral modeling, as shown in Figure 7.21. Each of these areas is described in a separate chapter of the UML 2.0: Superstructure specification. Note that some packages are dependent on one another in circular dependencies. This is because the dependencies between the top-level packages show a summary of all relationships between their subpackages; there are no circular dependencies between subpackages of those packages.

7.3.2.4.1.2.5 Reusing Infrastructure
One of the primary uses of the UML 2.0 Infrastructure specification is that it should be reused when creating other metamodels. The UML metamodel reuses the InfrastructureLibrary in two different ways:
All of the UML metamodel is instantiated from meta-metaclasses that are defined in the InfrastructureLibrary.

The UML metamodel imports and specializes all metaclasses in the InfrastructureLibrary.

As was discussed earlier, it is possible for a model to be used as a metamodel, and here we make use of this fact. The InfrastructureLibrary is in one capacity used as a meta-metamodel and in the other capacity as a metamodel, and is thus reused in two dimensions.

7.3.2.4.1.2.6 The Kernel Package
The InfrastructureLibrary is primarily reused in the Kernel package of Classes in UML 2.0: Superstructure; this is done by bringing together the different packages of the Infrastructure using package merge. The Kernel package is at the very heart of UML, and the metaclasses of every other package are directly or indirectly dependent on it. The Kernel package is very similar to the Constructs package of the InfrastructureLibrary, but adds more capabilities to the modeling constructs that were not necessary to include for purposes of reuse or alignment with MOF.

Because the Infrastructure has been designed for reuse, there are metaclasses—particularly in Abstractions—that are partially defined in several different packages. These different aspects are for the most part brought together into a single metaclass already in Constructs, but in some cases this is done only in Kernel. In general, if metaclasses with the same name occurs in multiple packages, they are meant to represent the same metaclass, and each package where it is defined (specialized) represents a specific factorization. This same pattern of partial definitions also occurs in Superstructure, where some aspects of, for example, the metaclass Class are factored out into separate packages to form compliance points (see the following text).

7.3.2.4.1.2.7 Metamodel Layering
The architecture that is centered around the Core package is a complementary view of the four-layer metamodel hierarchy on which the UML metamodel has traditionally been based. When dealing with metalayers to define languages, there are generally three layers that always have to be taken into account:

- The language specification, or the metamodel
- The user specification, or the model
- Objects of the model

This structure can be applied recursively many times so that we get a possibly infinite number of metalayers; what is a metamodel in one case can be a model in another case, and this is what happens with UML and MOF. UML is a language specification (metamodel) from which users can define their own models. Similarly, MOF is also a language specification (metamodel) from which users can define their own models. From the perspective of MOF, however, UML is viewed as a user (i.e., the members of the OMG that have developed the language) specification that is based on MOF as a language specification. In the four-layer metamodel hierarchy, MOF is commonly referred to as a meta-metamodel, even though strictly speaking it is a metamodel.

7.3.2.4.1.2.8 The Four-Layer Metamodel Hierarchy
The meta-metamodeling layer forms the foundation of the metamodeling hierarchy. The primary responsibility of this layer is to define the language for specifying a metamodel. The layer is often referred to as M3, and MOF is an example of a meta-metamodel. A meta-metamodel is typically
more compact than the metamodel that it describes, and often defines several metamodels. It is generally desirable that related metamodels and meta-metamodels share common design philosophies and constructs. However, each layer can be viewed independently of other layers, and needs to maintain its own design integrity.

A metamodel is an instance of a meta-metamodel, meaning that every element of the metamodel is an instance of an element in the meta-metamodel. The primary responsibility of the metamodel layer is to define a language for specifying models. The layer is often referred to as M2; UML and the OMG CWM are examples of metamodels. Metamodels are typically more elaborate than the meta-metamodels that describe them, especially when they define dynamic semantics. The UML metamodel is an instance of the MOF (in effect, each UML metaclass is an instance of an element in *InfrastructureLibrary*).

A model is an instance of a metamodel. The primary responsibility of the model layer is to define languages that describe semantic domains, i.e., to allow users to model a wide variety of different problem domains, such as software, business processes, and requirements. The things that are being modeled reside outside the metamodel hierarchy. This layer is often referred to as M1. A user model is an instance of the UML metamodel. Note that the user model contains both model elements and snapshots (illustrations) of instances of these model elements. The metamodel hierarchy bottoms out at M0, which contains the runtime instances of model elements defined in a model. The snapshots that are modeled at M1 are constrained versions of the M0 runtime instances.

When dealing with more than three metalayers, it is usually the case that the ones above M2 gradually get smaller and more compact the higher up they are in the hierarchy. In the case of MOF, which is at M3, it consequently only shares some of the metaclasses that are defined in UML. A specific characteristic about metamodeling is the ability to define languages as being reflective, i.e., languages that can be used to define themselves. The *InfrastructureLibrary* is an example of this, because it contains all the metaclasses required to define itself. When a language is reflective, there is no need to define another language to specify its semantics. MOF is reflective because it is based on the *InfrastructureLibrary*, and there is thus no need to have additional metalayers above MOF.

### 7.3.2.4.1.2.9 Metamodeling

When metamodeling, one primarily distinguishes between metamodels and models. As already stated, a model that is instantiated from a metamodel can in turn be used as a metamodel of another model in a recursive manner. A model typically contains model elements. These are created by instantiating model elements from a metamodel, i.e., metamodel elements.

The typical role of a metamodel is to define the semantics for how model elements in a model gets instantiated. As an example, consider Figure 7.22, where the metaclasses Association and Class are

![Diagram](image-url)

Figure 7.22 An example of metamodeling; note that not all InstanceOf relationships are shown.
both defined as part of the UML metamodel. These are instantiated in a user model in such a way that the classes Person and Car are both instances of the metaclass Class, and the association Person\-car between the classes is an instance of the metaclass Association. The semantics of UML defines what happens when the user-defined model elements are instantiated at M0, and we get an instance of Person, an instance of Car, and a link (i.e., an instance of the association) between them.

The instances—sometimes referred to as _runtime instances_—that are created at M0 from, for example, Person should not be confused with instances of the metaclass InstanceSpecification that are also defined as part of the UML metamodel. An instance of an InstanceSpecification is defined in a model at the same level as the model elements that it illustrates, as is depicted in Figure 7.23, where the instance specification Mike is an illustration (or a snapshot) of an instance of class Person.

### 7.3.2.4.1.2.10 An Example of the Four-Level Metamodel Hierarchy

An illustration of how these metalayers relate to one another is shown in Figure 7.24. It should be noted that we are by no means restricted to only these four metalayers, and it would be possible to

---

**Figure 7.23**  Giving an illustration of a class using an instance specification.

**Figure 7.24**  An example of the four-layer metamodel hierarchy.
define additional ones. As is shown, the metalayers are usually numbered from M0 and upward, depending on how many metalayers are used. In this particular case, the numbering goes up to M3, which corresponds to MOF.

7.3.2.4.2 Language Formalism

The UML specification is defined by using a metamodeling approach that adapts formal specification techniques. The formal specification techniques are used to increase the precision and correctness of the specification. This subsection explains the specification techniques used to define UML.

The following are the goals of the specification techniques used to define UML:

- Correctness—The specification techniques should improve the correctness of the metamodel by helping to validate it. For example, the well-formedness rules should help validate the abstract syntax and help identify errors.
- Precision—The specification techniques should increase the precision of both the syntax and semantics. The precision should be sufficient so that there is no syntactic nor semantic ambiguity for either implementers or users.
- Conciseness—The specification techniques should be parsimonious, so that the precise syntax and semantics are defined without superfluous detail.
- Consistency—The specification techniques should complement the metamodeling approach by adding essential detail in a consistent manner.
- Understandability—While increasing the precision and conciseness, the specification techniques should also improve the readability of the specification. For this reason a less than strict formalism is applied, because a strict formalism requires formal techniques.

The specification technique used describes the metamodel in three views using both text and graphic presentations. It is important to note that the current description is not a completely formal specification of the language, because to do so would have added significant complexity without clear benefit. The structure of the language is nevertheless given a precise specification, which is required for tool interoperability. The detailed semantics are described using natural language, although in a precise way so they can easily be understood. Currently, the semantics are not considered essential for the development of tools; however, this will probably change in the future.

7.3.2.4.2.1 Levels of Formalism

A common technique for specification of languages is to first define the syntax of the language and then to describe its static and dynamic semantics. The syntax defines what constructs exist in the language and how the constructs are built up in terms of other constructs. Sometimes, especially if the language has a graphic syntax, it is important to define the syntax in a notation-independent way (i.e., to define the abstract syntax of the language). The concrete syntax is then defined by mapping the notation onto the abstract syntax.

The static semantics of a language define how an instance of a construct should be connected to other instances to be meaningful, and the dynamic semantics define the meaning of a well-formed construct. The meaning of a description written in the language is defined only if the description is well formed (i.e., if it fulfills the rules defined in the static semantics).
The specification uses a combination of languages—a subset of UML, an object constraint language, and precise natural language to describe the abstract syntax and semantics of the full UML. The description is self-contained; no other sources of information are needed to read the document. Although this is a metacircular description, understanding this document is practical because only a small subset of UML constructs are needed to describe its semantics.

In constructing the UML metamodel, different techniques have been used to specify language constructs, using some of the capabilities of UML. The main language constructs are reified into metaclasses in the metamodel. Other constructs, in essence being variants of other ones, are defined as stereotypes of metaclasses in the metamodel. This mechanism allows the semantics of the variant construct to be significantly different from the base metaclass. Another more “lightweight” way of defining variants is to use meta-attributes. As an example, the aggregation construct is specified by an attribute of the metaclass AssociationEnd, which is used to indicate if an association is an ordinary aggregate, a composite aggregate, or a common association.

Interested readers are referred to [UML200501] and [UMI200401] for additional information on UML.

7.4 Model-Driven Architecture

7.4.1 MDA Background

The OMG Model-Driven Architecture™ (MDA) addresses the life cycle that spans design, deployment, integration, and management of applications. The OMG’s MDA initiative is an evolving conceptual architecture for a set of industrywide technology specifications that support a model-driven approach to software development. Although MDA is not itself a technology specification, it represents an approach and a plan to achieve a cohesive set of model-driven technology specifications. MDA aims at unifying business modeling and supporting technology (from legacy systems to the newly introduced middleware platforms) into an industry-standard architecture [SEI200201]. Covering both the modeling and development arena, MDA is a comprehensive IT architecture that unifies business modeling and implementation into a synergistic environment (such environment is seen by proponents as being able to maximize IT ROI and give businesses that employ it a competitive advantage.) Figure 7.25 depicts the MDA environment graphically. The summary that follows is based in large part on OMG’s information; additional details can be obtained by direct reference to these materials [MDA200301].

Before MDA became available, models and programming code were developed separately by different groups of people. Programmers regarded the models as guidelines or rough plans rather than firm requirements. Compared with development without modeling, this approach typically can be more expensive, and it can take longer to produce a final application. Here, a firm may find limited benefit from its modeling effort. MDA addresses this issue by codifying and standardizing the steps that take a model through development into implementation. MDA-based tools produce applications that meet the business requirements along with related nonfunctional requirements (e.g., scalability, reliability, security) that have been built into models by domain experts and IT architects. Models become the prime development artifact in this environment, not only defining and recording business requirements but also serving as the basis for development, maintenance, and evolution [SEI200201]. Typically, if a developer is building a Web Services application, for example, the developer is forced to assemble a number of legacy functions on legacy middleware to
a new front end; MDA is ideally suited to designing and building this kind of application. *UML 2.0: Infrastructure* supports MDA. MDA is now supported by many prominent software vendors.

OMG members voted to establish the MDA as the base architecture in late 2001. Software development in the MDA starts with a PIM of an application’s business functionality and behavior, constructed using a modeling language based on OMG’s MOF. The MDA approach is a way of developing applications and writing specifications, based on a PIM of the application or specification’s business functionality and behavior. A complete MDA specification consists of a definitive platform-independent base model, plus one or more PSMs and sets of interface definitions, each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support. MDA was defined and is nurtured by the OMG membership, which includes a diverse cross section of computer vendors, software suppliers, and many end users. MDA is defined by the *MDA Guide*, Version 1.0.1. OMG members were expecting to replace this interim version with an update, based on the Foundation Model also just mentioned, at the time of this writing.

MDA (as its name declares) is an architecture that works above the level of a middleware platform, including, for example, .NET and Web Services. A middleware platform is incorporated into the MDA as a platform-specific profile; OMG members may (in due course) define platform-specific profiles for .NET and Web Services.

MDA is both an OMG standard and a generic way to develop software. The OMG standard incorporates, among others, UML, MOF, the XMI, and the CWM. The two most important elements are UML and MOF, which helps to translate the models into specific code (XMI deals with sharing models and other development artifacts over the Web.) Some see MDA as helping UML to become more “real”: prior to MDA one could use UML to develop graphs to help communicate concepts with stakeholders; however, the hard work of coding was a stand-alone effort. MDA makes modeling concrete, allowing UML to become, once again, a programming language. Among its other features, UML 2.0 improves the support for component-based software [AMB200301].
The MDA-based approach affords a number of benefits. Some benefits are business-oriented (for example, requirements built into the model always appear in the final implementation); others are technical (MDA-based applications are interoperable, portable, and middleware-platform-independent) [SEI200201]. MDA benefits include the following:

- In an MDA development project, attention focuses first on the application’s business functionality and behavior, allowing stakeholders’ investment to concentrate on the aspects that critically affect core business processes. Technical aspects, also critical but secondary to business functions, are well handled by automated or semiautomated development tools.
- An architecture based on the MDA is able to evolve over time; also, such an architecture makes it relatively easy to integrate applications and facilities across middleware boundaries.
- Domain facilities defined in the MDA provide interoperability: the application is available on a domain’s preferred platform, and on multiple platforms whenever there is a need.

Additional benefits of MDAs include the following [SEI200201]:

- MDA-enabled tools follow OMG-standardized pathways to automate the transformation from the designers’ business model into a firm-specific implementation, producing new applications faster, better, and cheaper.
- The MDA process ensures not only that the business requirements built into the design end up in the final implementation, but also that nonbusiness functional requirements carry through as well.
- Code generated by MDA-enabled development tools is derived from libraries based on patterns designed by the industry’s best developers.
- MDA applications interoperate. The MDA was designed from the start for implementation in multiple middleware platforms, and codes cross-platform invocations when needed, not only within a given application, but also from one to another regardless of the target platform assigned to each.
- MDA applications are portable. As they are based on technology-independent business models, they can be generated on any middleware platform.
- MDA applications may well be future-proof. When new platforms are introduced (as they must over the next decades as networking continues to mature and computers become smaller, more specialized, and more ubiquitous), OMG is expected to add mappings for them, and tool vendors will implement them in their offerings. Using these new mappings, existing MDA-based applications can be made either to interoperate with others, or can be entirely reimplemented on the new platform.
- The MDA supports enterprise application integration. The entire enterprise’s suite of applications can be thought as being a library of UML models. The developer selects the ones that need to work together, incorporates them into the MDA tool, and draws lines denoting the interoperability pathways. An MDA application does not have to make all of its invocations using the middleware of its PSM: the code database of an MDA tool includes invocation formats for every supported middleware platform. It follows that developers can pull models of existing applications and services from libraries into the project’s environment as they construct new PIMs, and set up cross-platform invocations by simply drawing the connections in their new model. It is likely that some of these existing applications will not be on the same platform as the new PSM. Taking their cue from the actual middleware platform
of these existing applications, MDA tools will generate cross-platform invocations where needed. Legacy application based on a UML model and a supported middleware platform can be included in a company’s circle of MDA interoperability by simply importing its model into MDA tools as PIMs for new applications are built.

To benefit an industry, a standard must be used by a critical mass of companies. Technology-specific standards will encounter challenges getting established where platform incompatibility prevents the achievement of this critical mass. Sometimes, the problem is more deep-rooted than this: in some industries, architecturally excellent standards were adopted in the formal sense but failed to win adherence because they were written for a platform that few companies were willing to support. MDA removes these roadblocks. Under MDA, the functional description of every standard is technology-independent, and the architecture is capable of producing interoperating implementations on multiple platforms. This allows an industry to define the business functionality and behavior of its standards as a PIM, and then produce PSMs and implementations on whatever platforms the participants require. In addition, technology-based standards become obsolete as their base platform ages. This is not a problem for MDA-based specifications: because they are based on platform-independent PIMs and can be made to interoperate with new platforms, or even reimplemented on them, through the MDA development process, MDA-based applications and specifications are truly “future-proof.”

MDA development focuses first on the functionality and behavior of a distributed application or system, undistorted by idiosyncrasies of the technology platform or platforms on which it will be implemented. In this way, MDA separates implementation details from business functions. Thus, it is not necessary to repeat the process of defining an application or system’s functionality and behavior each time a new technology (Web Services, for example) comes along. Other architectures are generally tied to a particular technology. With MDA, functionality and behavior are modeled just once. Mapping from a PIM through a PSM to the supported MDA platforms is being implemented by tools, easing the task of supporting new or different technologies. MDA models must be detailed because the application code will be generated from it and the code will include only functionality represented explicitly.

One can think of MDA as a stack with business at the top and technology at the bottom. Business domain experts work at the top, with the goal of modeling the needed business function. Here, UML-based tools provide support and the UML language’s structure and narrower profiles (i.e., tailored subsets) provide guidance. The product of the first development step, termed the PIM, represents the business functionality and behavior that this MDA application will execute. As one moves down toward the bottom, the business domain recedes and technology takes over. In a perfectly efficient world, the MDA might jump directly from the business model at the top to the implementation at the bottom, but today the discontinuities are too great, so the MDA inserts an intermediate step. (The artifact produced in this step is termed the PSM.) Produced primarily by MDA-enabled tools following OMG-standardized mappings, the PSM provides a middle stage where skilled architects can mark up the model with their preferences or hints about how they want particular steps to be designed or executed. The completed PSM contains the same information set as a coded application, but in the form of a UML model instead of program language and makefiles. Taking advantage of the tight mapping, MDA-enabled development tools automate the conversion from PSM to code very well (this step is more mature than the PIM–PSM conversion in the previous step) [SEI200201].

Although it has always been true that UML models can be implemented on any platform, the continuing proliferation of middleware suggested that a platform-independent MOF-based model
is the key to software stability; such a model remains fixed while the infrastructure landscape around it shifts over time. The MDA unites OMG’s well-established modeling standards with middleware technology to integrate what one has built, with what the firm is building, with what the firm might build in the future. MDA designs portability and interoperability into the application at the model level.

Even though UML is usually thought of as the basis for MDA, it is actually MOF compliance that is formally required for a tool or tool chain to be labeled “MDA Compliant.” The MOF is OMG’s foundation specification for modeling languages; MOF compliance allows UML structural and behavioral models, and CWM data models, to be transmitted via XMI, stored in MOF-compliant repositories, and transformed and manipulated by MOF-compliant tools and code generators. Models in the context of the MDA Foundation Model are instances of MOF metamodels and therefore consist of model elements and links between them. This required MOF compliance enables the automated transformations on which MDA is built. UML compliance, although common, is not a requirement for MDA models. Additional OMG specifications populate the architecture. Development tools, provided by vendors, implement the supporting standards. Working synergistically, these tools constitute the working MDA modeling and development environment in which architects and developers create MDA applications.

Although not formally required, UML is still a key enabling technology for the MDA and the basis for nearly all MDA development projects. (Work in some specialized fields requires specially tailored modeling languages, although the additional capabilities added to UML by the 2.0 revision satisfies this need in many cases.) Hence, application development using the MDA is typically based on a normative, platform-independent UML model. By leveraging OMG’s MOF and UML standards, MDA allows creation of applications that are portable across, and interoperable naturally across, a broad spectrum of systems from embedded, to desktop, to server, to mainframe, and across the Internet. Any modeling language used in MDA must be described in terms of the MOF language to enable the metadata to be understood in a standard manner, which is a precondition for the ability to perform automated transformations.

Patterns play a key role in most MDA-based development projects. Successful transformation from PIM to PSM, and from PSM to code, requires that the PIM contain enough detail to completely guide the software tools through the process. By incorporating this detail through the use of patterns instead of inserting it by hand, one can gain multiple benefits: architects do less work, the resulting PIM reflects the collective wisdom of many contributors, and the tools can work the pattern (parameterized as necessary in the UML models) through the transformations, ultimately pulling implementation code from a library written by experts and inserting it into the application.

In the MDA, middleware-specific models and implementations are secondary artifacts: a specification’s PIM—the primary artifact—defines one or more PSMs, each specifying how the base model is implemented on a different middleware platform. Because the PIM, PSMs, and interface definitions are all part of an MDA specification, OMG now adopts specifications in multiple middleware platforms under the MDA. Suitable targets for MDA development projects and OMG specifications include WS, XML, .NET, EJB, and OMG’s CORBA.

MDA enables cross-platform interoperability. In the MDA, the base specification of every service, facility, and application is a platform-independent model. Architects specify links from an application to needed services and facilities, and to other applications, as part of its model. As the PIM for the new MDA specification or application is transformed into a PSM and then a coded application by the MDA tool chain, interoperability with other applications and services is built into it according to these links, which are implemented properly regardless of the target’s native middleware.
An extensive set of services is necessary to support distributed computing, both within an enterprise and among many over the Internet. In the MDA, these are known as Pervasive Services; a single implementation of each, regardless of the platform on which it runs, can service every application or client that needs its capabilities via MDA-generated cross-platform bridges. There are four Pervasive Services:

- Directory Services
- Transaction Services
- Security Services
- Distributed Event and Notification Services

Additional services are expected to be added as needed to keep the environment complete.

In terms of products, MDA is being implemented by tools—or tool chains, which may come from a single vendor or a number of vendors—that integrate modeling and development into a single environment that carries an application from the PIM, through the PSM, and then via code generation to a set of language and configuration files implementing interfaces, bridges to services and facilities, and possibly even business functionality. Several vendors already provide tools that support integration at about this level, including substantial code generation. Today’s tools typically automate 50% to 70% of the PIM-to-PSM transformation; because the industry got started on the second step much earlier, automation of the PSM-to-code transformation is typically 100% or nearly so.

MDA development tools, available now from many vendors, convert the PIM first to a PSM and then to a working implementation on virtually any middleware platform: Web Services, XML/SOAP, EJB, C#/.Net, OMG’s CORBA, or others. Portability and interoperability are built into the architecture. OMG Task Forces organized around industries including finance, manufacturing, biotechnology, space technology, and others use the MDA to standardize facilities in their domains.

7.4.2 MDA Support

This section identifies OMG elements that support/comprise MDA. These have been mentioned along the way, but are covered here as a group.

7.4.2.1 The Meta-Object Facility (MOF™)

In the MDA, models are first-class artifacts, integrated into the development process through the chain of transformations from PIM through PSM to coded application. To enable this, the MDA requires models to be expressed in a MOF-based language. This guarantees that the models can be stored in a MOF-compliant repository, parsed, and transformed by MOF-compliant tools, once rendered into XMI for transport over a network. This does not constrain the types of models you can use—MOF-based languages today model application structure, behavior (in many different ways), and data; OMG’s UML and CWM are examples of MOF-based modeling languages but are not the only ones. The foundation for UML 2.0 is MOF 2.0.

As noted, each MDA-based specification has, as its normative base, two levels of models: a PIM, and one or more PSMs. For many specifications, these will be defined in UML, making OMG’s standard modeling language a foundation of the MDA. (Use of UML, although common, is not a
requirement; MOF is the mandatory modeling foundation for MDA.) Tailored to MDA requirements, UML 2.0 improves Business, Architectural, Structural, and Behavioral modeling and is being done in four parts. Several additional specifications help tailor the UML to support MDA:

- A Human-Usable Textual Notation (HUTN) enables a new class of model-editing programs and enhances the way models (if they are built using a MOF-based modeling language, of course) can be manipulated. Notation elements map one-to-one to the more verbose XMI, but the syntax of the HUTN is much more human-friendly.
- A standard Software Process Engineering Metamodel defines a framework for describing methodologies in a standard way. It does not standardize any particular methodology, but enhances interoperability from one methodology to another.

7.4.2.2 UML Profiles

UML Profiles tailor the language to particular areas of computing (such as Enterprise Distributed Object Computing) or particular platforms (such as EJB or CORBA). In the MDA, both PIMs and PSMs are defined using UML profiles; OMG is well along the way, defining a suite of profiles that span the entire scope of potential MDA applications. The current suite of profiles includes the following:

- The UML Profile for CORBA, which defines the mapping from a PIM to a CORBA-specific PSM.
- The UML Profile for CCM (the CORBA Component Model), OMG’s contribution to component-based programming. EJBs are the Java mapping of CCM; an initial take on a profile for EJB appears as an appendix of the UML 2.0 Superstructure specification.
- The UML Profile for EDOC is used to build PIMs of enterprise applications. It defines representations for entities, events, process, relationships, patterns, and an Enterprise Collaboration Architecture. As a PIM profile, it needs mappings to platform-specific profiles. A mapping to Web Services is under way; additional mappings will follow.
- The UML Profile for EAI defines a profile for loosely coupled systems—that is, those that communicate using either asynchronous or messaging-based methods. These modes are typically used in Enterprise Application Integration, but are used elsewhere as well.
- The UML Profile for Quality of Service (QoS) and Fault Tolerance defines frameworks for real-time and high-assurance environments.
- The UML Profile for Schedulability, Performance, and Time supports precise modeling of predictable—that is, real-time—systems, precisely enough to enable quantitative analysis of their schedulability, performance, and timeliness characteristics.
- The UML Testing Profile provides important support for automated testing in MDA-based development environments.

7.4.2.3 XML Metadata Interchange (XMI™)

XMI defines an XML-based interchange format for UML and other MOF-based metamodels and models (because a metamodel is just a special case of a model), by standardizing XML document formats, DTDs, and schemas. In so doing, it also defines a mapping from UML to XML. Because one of OMG’s XMI updates reflects the incorporation of XML Schemas, while MOF
point updates were made periodically through OMG’s established maintenance process, numbering of XMI and MOF versions diverged. The following table provides a mapping.

<table>
<thead>
<tr>
<th>MOF</th>
<th>XMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3</td>
<td>1.1</td>
</tr>
<tr>
<td>1.4 (current)</td>
<td>1.2</td>
</tr>
<tr>
<td>1.4 (current)</td>
<td>1.3 (adds Schema support)</td>
</tr>
<tr>
<td>1.4 (current)</td>
<td>2.0 (current; new format)</td>
</tr>
<tr>
<td>2.0 (in process)</td>
<td>2.1 (in process)</td>
</tr>
</tbody>
</table>

7.4.2.4 Common Warehouse MetaModel (CWM™)

The CWM standardizes a complete, comprehensive metamodel that enables data mining across database boundaries at an enterprise. Similar to a UML profile but in data space instead of application space, it forms the MDA mapping to database schemas. CWM does for data modeling what UML does for application modeling. The CWM is a formal OMG specification. A supplementary specification, CWM Metadata Interchange Patterns, defines patterns that smooth the way to data modeling tool interoperability.

7.4.2.5 CORBA®

Although MDA can target every middleware platform and will map to all that have significant market presence, CORBA plays a key role as a target platform because of its programming language, operating system, and vendor independence. CORBA is a vendor-independent middleware standard.

7.4.2.6 Writing Standards in the MDA

Applications and frameworks (that is, parts of applications that perform a particular function) can all be defined in the MDA as a base PIM that maps to one or more PSMs and implementations. Standards written this way enjoy two advantages:

- The base PIM is truly a business specification, defining *business functionality and behavior* in a technology-independent way. Technological considerations do not intrude at this stage, making it easy for business experts to model exactly the business rules they want into the PIM.
- Once business experts have completed the PIM, it can be implemented on virtually any platform, or on multiple platforms with interoperability among them, to meet the needs of the industry and companies that use it.

OMG Domain Task Forces, after years of writing specifications in only CORBA, are now writing their base specifications in the MDA to take advantage of these considerations. OMG recognizes (based on analogy with the CORBA-based Object Management Architecture) three levels of MDA-based specifications:

- The Pervasive Services, including enterprise necessities such as Directory Services, Transactions, Security, and Event handling (Notification).
The Domain Facilities, in industries such as healthcare, manufacturing, telecommunications, biotechnology, and others.

Applications themselves, perhaps created and maintained by a software vendor or end-user company or enterprise using MDA tools to run an MDA-based methodology, but not standardized by OMG.

7.4.2.7 The Pervasive Services

This category includes at least the following:

- Directory and Naming Services
- Event Handling/Notification Services

Additional pervasive services may be defined, either from the list of CORBA services already standardized by OMG or from other suggestions from OMG members. Transactions and Security, the other two most popular CORBA services, may or may not be part of this group—in the component world, transactionality and security are attributes of a running system rather than services that a program calls, because of the way the Component Container or Application Server is set up to run transactionally and securely as part of an application’s environment. OMG members are already taking the group’s well-established CORBA service specifications and mapping them back to PIMs, where they can serve all platforms through the MDA development pathway.

7.4.2.8 Domain-Specific Specifications

Various industry segments have typically defined computing standards based on a particular technology. This provides interoperability, but requires every company to use the same middleware. By defining standards in the MDA, industries avoid the underlying disadvantages that invariably arise: defined fundamentally as a PIM, their standard can be implemented equivalently and interoperably on multiple middleware platforms. Over time, if one or some of these platforms become obsolete, the industry can define new implementations on new platforms from the original PIM. Many industries are working on MDA standards at OMG, including telecommunications, biotechnology, manufacturing, healthcare, and finance [SEI200201].

Interested readers are referred to [MDA200301] for additional information on MDA.
Chapter 3
Positioning Enterprise Architecture

In the previous chapter, we have discussed the needs for enterprise architecture. This chapter is concerned with enterprise architecture as a means to meet these needs. We will start this chapter with a historical perspective on the concept of architecture as a means of obtaining insight into, as well as harnessing, complexity. To gain a better insight into the role of enterprise architecture in governing transformations, Sect. 3.2 will then discuss the governance paradigm and relate this to the role of enterprise architecture. Based on this discussion, Sect. 3.3 then continues by identifying seven possible applications of enterprise architecture from a governance perspective. Using this as a context, Sect. 3.4 provides a discussion of several definitions of enterprise architecture, while also providing the definition of enterprise architecture as used in this book. To make this definition more specific and tangible, Sect. 3.5 will discuss the key concepts underlying enterprise architecture, while Sect. 3.6 will highlight the benefits of enterprise architecture in relation to the needs identified in the previous chapter. Finally, Sect. 3.7 takes a first brief look at the competencies needed from the architect.

3.1 A Historical Perspective on Enterprise Architecture

The recorded history of classical architecting began more than 4,000 years ago in Egypt with the erection of the pyramids, the complexity of which had been overwhelming designers and builders alike [82, 113]. This complexity had at its roots in the phenomenon that as systems became increasingly more ambitious, the number of interrelationships among the elements increased far faster than the number of elements themselves. Pyramids were no longer simple burial sites; they had to be demonstrations of political and religious power, secure repositories of god-like rulers and their wealth, and impressive engineering accomplishments. Each of demands, in itself, already required major resources. The complex interrelationships among combined elements were well beyond what the traditional tools of the engineers and builders could handle. This led to the introduction of architecture as a means to obtain and maintain insight into these complex relationships. We have remained doing so. Following the evolution of our societies, we have used architecture as a means of obtaining insight and harnessing complexity of a wide variety of constructs as illustrated in Fig. 3.1.

After years of architecture in the physical world, the term has also taken a foothold in the field of IT. Architecture is well known from the world of construction. Therefore, some 20 years ago, the IT industry became confronted with complex structures and decision, a comparison with the construction industry seemed an obvious one. Probably the first person to use the term architecture in this context was
Blaauw [6]: “The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.” Blaauw was the codeveloper of the IBM 360 computer family in the 1960s. In his publications, he refers to the architecture (i.e., design) of computers, while discussing such topics as modularity, reliability, and consistency. At about the same time, Dijkstra started his work on structured programming. Although he did not use the word architecture, he repeatedly underlined the importance of the structure of software, thus laying certain foundations for architecture. This comparison leads to terms such as software engineering and structured programming. At that time, this comparison brought a degree of order into many aspects of the creation of these programs as advocated by Parnas [100].

When software applications became larger and larger, people such as Shaw and Garlan coined the term software architecture [123]. This notion of architecture deals with the key design principles underlying software artefacts. In the 1980s and 1990s, people became aware that the development of information technology (IT) should be done in conjunction with the development of the context in which it was used. This led to the identification of the so-called business/IT alignment problem [55, 99, 135]. Solving the business/IT alignment problem requires enterprises to align human, organizational, informational, and technological aspects of systems. Quite early on, the term architecture was also introduced as a means to further alignment, and thus analyzes and solves business/IT alignment problems [21, 135, 155]. Recently, the awareness emerged that alignment between business and IT is not enough; there are
many more aspects in the enterprise in need of alignment. This has led to the use of the term architecture at the enterprise level: enterprise architecture [18, 20, 37, 78].

### 3.2 Governance Paradigm

According to [1], governance is “the activity of controlling a company or an organization” or in other words, the supervision of the compliance of rules. In our view, enterprise architecting is an integral part of the governance of an enterprise and its transformation.

Ideally, an enterprise architecture plays a pivotal role in the continuous improvement process of an enterprise. In order to better understand the governing role of enterprise architecture, this section offers a discussion of the governance paradigm [79], and consequently applies it to an enterprise transformation context. Figure 3.2, which is based on [79], depicts the basic governance paradigm. The governance paradigm involves three important assumptions:

1. there is some system,¹ the target system, which interacts with its environment;
2. this target system needs to be governed;
3. there is another system, the governing system which does the actual governing.

The essence of the governance paradigm is that during the realization of a process there is some kind of interaction with the environment (input and output), and that this process is controlled by some (internal) authority which monitors, and if necessary adjusts, the process to make sure the intended objectives are reached. This

![Fig. 3.2 The basic governance paradigm](image_url)

¹Note: system here is to be understood in its original sense of the term [11], and not as a synonym to application system as is the case in software development. In the context of enterprise architecture, we are specifically interested in active systems [28].
authority is called governing system (GS).\textsuperscript{2} The system governed by the GS is referred to as the target system (TS). Since an organization is part of a larger system, the GS also interacts with the environment to determine which services of products to deliver, to determine new opportunities and to determine changes in the environment.

In the case of enterprise architecting, the target system that needs governing is the transformation process of the enterprise, where not-transforming, i.e., maintaining a status quo is considered as a special transformation process. The latter case may actually take more effort than one would expect. Maintaining a status quo requires activities preventing erosion of an existing structure. Taking the enterprise as the target system, leads to the situation as depicted in Fig. 3.3. In an operational enterprise,

\textsuperscript{2}Note that the original governance paradigm used Dutch terminology. In [80], an English translation can be found using the term target system and controlling organ. Since in the field of enterprise architecture, the term governance is used rather than controlling, we prefer to use term governing rather than controlling. To also stress the fact that the governing organ really is a system, we shall actually use the term governing system.
3.2 Governance Paradigm

Fig. 3.4 The role of enterprise architecture

a distinction is made between a target system comprising the operational processes and a governing system, which governs these operational processes. The operational enterprise is transformed (to better meet the challenges and opportunities posed by its environment) by an enterprise transformation system. This transformation system is comprised of a transformation governing system and the actual transformation process(es). These latter processes constitute the target of the transformation governing system.

As mentioned before, enterprise architecting should be regarded as being a part of the governance of the enterprise transformation. Figure 3.4, therefore, shows a refined view on the governance of an enterprise’s transformation processes involving three subdomains: strategy, architecture, and program management [116].

Based on the requirements on enterprise architecture as a means, as discussed in the previous chapter, enterprise architecting can be likened to the use of a “dashboard” which allows the architect and stakeholders to steer the enterprise’s transformation processes. When using the dashboard as a metaphor, the “dashboard” displays the enterprise architecture in terms of relevant aspects of the current state of the enterprise, its future direction, and the desired states of the enterprise. Just as the selected/displayed speed, altitude, and direction of an airplane is not the dashboard, but rather displayed on the dashboard, the dashboard is not the enterprise architecture. Analogously, it is the enterprise architecture, or rather a part thereof, what will be displayed on the dashboard. In addition, the dashboard may contain a report on the gaps between the current state and desired states, as well as its operational performance in terms of its current state.

In an airplane, a “dashboard” comprises of indicators (meters, lights, etc.) and controls (levers, handles, pedals, and knobs). In the case of enterprise architecture as a means to govern transformations, the dashboard needs at least:

- **indicators** giving insight into:
  - the enterprise’s current state,
  - the enterprise’s future state,
  - the enterprise’s current performance,
  - the enterprise’s future (expected) performance,
  - the direction and progress of its transformation processes,
Fig. 3.5 Enterprise architecture on a dashboard

- **controls** allowing the transformation processes to be influenced.

   The indicators may take the form of models, views, performance measurements, etc. The controls may take the form of (enforced) reference models, design principles, standards, etc. This is illustrated in Fig. 3.5. The process of measuring, providing insight, decision-making, and directing the enterprise’s transformation process is a continuous (and far from linear) process. Based on the insights provided by the dashboard, the stakeholders in conjunction with the architect may decide to adjust the directions as set out on the dashboard.

   The situation depicted in Fig. 3.5 is still somewhat naive in the sense that it takes a rather reactive perspective. If the architect and stakeholders would have some kind of a predictive model, which predicts future properties of the enterprise, the transformation processes, and their environment (ecosystem), then this model can be used to more proactively steer the transformation process of the enterprise. This leads to the situation as shown in Fig. 3.6. Using a model of possible target systems, the enterprise system and the ecosystem in which they operate, what-if analysis can be conducted based upon which the actual transformation processes can be directed more proactively. This essentially leads to an experimentation environment with a shadow dashboard and shadow (eco)system. This experimentation environment will provide the stakeholder with insight in the impact of change, based on different scenarios.

### 3.3 Key Applications for Enterprise Architecture

Based on the needs and challenges of enterprises as discussed in the previous chapter (in particular, Sect. 2.6), we identify seven key applications for enterprise architecture as a means. In combination, these applications provide an instrument to make informed decisions as well as to ensure compliance of the transformation to these decisions, at several levels of specificity:

- **Situation description**—Use enterprise architecture as a means for goal/cause analysis to investigate problems/shortcomings in an existing situation. This also
involves the creation of a shared (among stakeholders) understanding of the existing situation.

– **Strategic direction**—Use enterprise architecture to express (and motivate) the future direction of an enterprise, as well as investigate (and evaluate) different alternatives. This also involves the creation of a shared (among stakeholders) conceptualization of the (possible) future directions, and shared agreement for the selected alternative.

– **Gap analysis**—Use enterprise architecture to identify key problems, challenges, issues, impediments, chances, threats, etc., as well as make well-motivated design decisions that enable a move from the existing situation into the desired strategic direction.

– **Tactical planning**—Use enterprise architecture to provide boundaries and identify plateaus (intermediary steps) for the transformation of the enterprise toward the articulated strategic direction. In this context, enterprise architecture is used as a planning tool, making the realization of a strategy more tangible.

– **Operational planning**—Use enterprise architecture to give a clear context and direction for a portfolio of projects working toward the realization of the first plateau as defined at the tactical planning level.

– **Selection of partial solutions**—Use enterprise architecture as a means to select one or more standard solutions and/or packages that are to become part of the solution and/or decide to outsource an entire business process/service to another enterprise.
3.4 Defining Enterprise Architecture

The previous sections will undoubtedly already have shed some light on what we regard as enterprise architecture. In this section, we will make this more specific by providing our own definition of this concept.

3.4.1 Definitions of Enterprise Architecture

Before providing our definition of enterprise architecture, we start with a discussion of some of the existing definitions of IT/information-enterprise architecture:

- **Solution architecture**—Use enterprise architecture to create the high level design of an actual step in the enterprise transformation as it will be realized (and implemented) in the context of a specific project.

  In Fig. 3.7, we have illustrated these seven application areas. Each of these seven application areas will yield different enterprise architectures, which are clearly interdependent. By ensuring compliance among these architectures, governance, and informed decision-making, from the strategic level to the operational level is enabled.

- The Institute of Electrical and Electronics Engineers (IEEE) defines architecture as: “An architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [60].”
• The Open Group’s Architectural Framework (TOGAF) defines architecture as: “Architecture has two meanings depending upon its contextual usage: (1) A formal description of a system, or a detailed plan of the system at component level to guide its implementation; (2) The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time [139].”

• The Clinger–Cohen Act’s definition of IT architecture is: “The term “information technology architecture,” with respect to an executive agency, means an integrated framework for evolving or maintaining existing information technology and acquiring new information technology to achieve the agency’s strategic goals and information resources management goals [142].”

• The Netherlands Architecture Forum (NAF), defines architecture conceptually as “a normative restriction of design freedom” and operationally as “a set of design principles [154].” As a background to this definition, NAF writes: “In general, the design freedom of designers is undesirable large. The idea of architecture is to take advantage of this. Therefore, architecture is defined as normative restriction of design freedom. This idea of consciously applying normative restriction of design freedom is the really new thing. It makes architecture a prescriptive notion; any descriptive interpretation is cogently rejected.”

• The ArchiMate Foundation defines enterprise architecture to be “A coherent whole of principles, methods, and models that are used in the design and realization of an enterprise’s organizational structure, business processes, information systems, and infrastructure [78].”

• The current architecture definition of Capgemini is: “An architecture is a set of principles, rules, standards, and guidelines, expressing and visualizing a vision and implementing concepts, containing a mixture of style, engineering, and construction principles.”

• A recent definition from the Gartner Group is: “Enterprise architecture (EA) is the process of translating business vision and strategy into effective enterprise change by creating, communicating, and improving the key principles and models that describe the enterprise’s future state and enable its evolution.”

The variety in these definitions does seem to indicate that the field of enterprise architecture is still in its infancy. At the same time, however, the wide spread attention of enterprise architecture does indicate that enterprises do feel a profound need to steer their development (including their business and IT portfolio), and that they are looking toward enterprise architecture as a means to fill this need.

3.4.2 Perspectives on the Role of Enterprise Architecture

While the above definitions may seem to differ considerably, what all these definitions seem to have in common is a reference to structure and relationships combined with a reference to a set of governing principles that provide guidance and support
for directions and decisions. Enterprise architecture focuses on shaping and governing the design of the future enterprise using principles to stipulate future direction and models to underpin and visualize future states. In our opinion, there are three important perspectives on the role of an enterprise architecture:

- **A regulation-oriented perspective**—which manifests itself as a prescriptive notion governing the design of an enterprise. When taking this perspective, one will focus on principles, leading to rules, guidelines, and standards, focusing the enterprise’s design freedom in the direction of its success.

- **A design-oriented perspective**—which emphasizes the comprehensive and cohesive specification of an enterprise in all its facets, as a high level design. This perspective focuses on essential design decisions, as well as its core structures. When taking this perspective, one typically produces models that describe the design of actual systemic artefacts and their interrelations.

- **A patterns-oriented perspective**—which focuses on the use of design patterns. This perspective forms a bridge between the regulative and the design perspectives. To meet the regulations set out in the regulative perspective, during design activities, suitable patterns can be applied.

The regulation and design-oriented perspectives correspond to the earlier mentioned indicator and control aspects of the dashboard paradigm as depicted in Fig. 3.5, and are complementary to each other in that the regulation-oriented perspective accommodates for the need to steer and direct developments, while the second perspective supports the need to gain insight into an enterprise’s design while also providing guidance to designers of enterprise systems.

Even though not many definitions of architecture explicitly refer to the patterns-oriented perspective, the role of patterns to capture and reuse design knowledge (such as the quality attributes that will result from using specific patterns) in the creation of architecture (be it for buildings, software, or enterprises) is evident [5, 15, 44, 123].

### 3.4.3 Definition of Enterprise Architecture

Using these perspectives, we can now define what we regard as enterprise architecture:

A coherent set of descriptions, covering a regulations-oriented, design-oriented and patterns-oriented perspective on an enterprise, which provides indicators and controls that enable the informed governance of the enterprise’s evolution and success.

### 3.4.4 Views in Enterprise Architectures

In practice, an enterprise architecture covers several foci that blend together to form the enterprise architecture. Without attempting to provide an exhaustive list, some typical (example) views are:
3.5 Key Concept of Enterprise Architecture

- In a business view, one would define the integrated structure of the overall business itself (in terms of organization, people and processes, and resources). Business architecture supports business change with a more holistic perspective. This approach is becoming more important with the move toward service-oriented architecture at the business level.
- In an IT view, one would define and describe the structure and relationships of IT systems including the way IT supports the enterprise to achieve its business goals.
- A governance view would address the full range of governance, from business governance (how to manage overall business processes, both formal and informal) to organizational and systems governance and also IT systems management capabilities.
- A security view addresses the full range of security, from business and information security to IT security. It also addresses the required security for organizational and business-related services. It is often linked to governance aspects to address security management.

In Chap. 4, we will discuss several dimensions along which to identify additional views. In the next section, the concept of view will be defined as being one of the key concepts of enterprise architecture.

3.5 Key Concept of Enterprise Architecture

Enterprise architecture can help organizations and their transformation processes in successfully executing their strategy. As such, it acts as an active planning and steering instrument, which can be used in translating strategy to programs and projects, and revolves around four main components: principles, models, views, and frameworks. Organizational transformation processes, embodied in programs and projects, can use the principles, models and views as a means of content based steering in the coherence of the solution. In this section, we will explore the concepts of concerns, principles, models, views, and frameworks.

3.5.1 Stakeholders and Their Concerns

An enterprise has many stakeholders. Future development of an enterprise is likely to impact on the interests of these stakeholders. In this section, we briefly survey some classes of stakeholders and their specific concerns. In this book, we use the definition of stakeholder and concern as provided in [60]. A stakeholder is an individual, team, or organization (or classes thereof) with interest in, or concerns relative to, a system (such as an enterprise). Concerns are those interests, which pertain to the system’s development, its operation or any other aspect that is critical or otherwise important to one or more stakeholders.
In making decisions about an enterprise’s future directions, stakeholders want to obtain insight into the impact these directions will have on their concerns, and understand the risks involved in current and future initiatives. Even more, since present day enterprises are complex social systems of interrelated processes, people and technology, stakeholders are keen on finding a way to harness this complexity when judging the impact on their concerns.

As discussed before, each type of stakeholder has its specific need for insight, control, and overview. At the same time, they all want insight into the potential impact on the enterprise resulting from changes in its own strategy or its environment, and consequences of decisions about the enterprise’s future directions. They also have the desire to communicate about these changes and impact. Communication will take place at enterprise level, business unit level, department level, and project level depending on the responsibilities of the stakeholder involved in the communication. Below, we briefly zoom in on the interests and concerns of three typical classes of stakeholders, and their needs on enterprise architecture.

### 3.5.2 Principles

An univocal understanding about what is of fundamental importance for the organization is essential. This is represented by the term “principle.” Even though no broadly accepted definition of principle exists yet, principles are generally regarded as constraints on the design space for enterprise engineers [98]. According to TOGAF [139], principles are general rules and guidelines, intended to be enduring and seldom amended, that inform and support the way in which an organization sets about fulfilling its mission. The extensible Architecture Framework (xAF) defines a principle as “a generic (functional or constructional) requirement for a class of systems [154],” where a class of systems is, e.g., all enterprise information systems, so not only for an individual system. According to Capgemini’s integrated architecture framework (IAF), a principle is a statement of belief, approach, or intent which directs the formulation of the architecture, and may refer to the current state or a desired future state [30, 45]. In this book, we will primarily follow the xAF definition as it provides an operational way of steering business and/or IT.

According to TOGAF, “a good set of principles will be founded in the beliefs and values of the organization and expressed in language that the business understands and uses. Principles should be few in number, future oriented, and endorsed, and championed by senior management. They provide a firm foundation for making architecture and planning decisions, framing policies, procedures, and standards, and supporting resolution of contradictory situations [139].” As discussed in [22], when considering the many different definitions of principles, three typical perspectives on principles can be discerned:

- **Principles as inherent laws**—referring to properties of (classes of) a system that can be observed and validated. Examples are the law of gravity, relativity theory, law of requisite variety, etc.
3.5 Key Concept of Enterprise Architecture

- **Principles as imposed laws**—referring to properties of (classes of) a system that can be validated. Examples are: traffic laws, societal laws, policies and regulations within organizations, such as *we opt for customer intimacy*, *we comply with privacy laws*, and *business flexibility has precedence over efficiency*. Principles as imposed laws typically address the concerns of stakeholders. Some of these concerns may actually be triggered by an *inherent law* which might have a negative impact on the system/enterprise being engineered.

- **Guidelines**—are properties of (classes of) a system that are specific enough to provide guidance to operational behavior to make it fit within the borders set out by imposed laws, possibly referring to the use of mechanisms. For example: “use your car’s cruise control” is an advisable *guideline* to abide by, in an effort to obeying *imposed laws* concerning maximum speeds on roads, using the in-built mechanism of the car’s cruise control.

In line with the definition of enterprise architecture used in this book, we will primarily use the last two perspectives on principles.

### 3.5.3 Models

In general, models are a purposeful abstraction of reality. More specifically, a model is defined as “any subject using a system $A$ that is neither directly nor indirectly interacting with a system $B$, to obtain information about the system $B$, is using $A$ as a model for $B$ [8].” In colloquial use in the context of enterprise engineering, the term model is equated to some graphical diagram. This colloquialism can be explained as most models used in software development, business process (re)engineering, etc., are graphical models. Models, however, do not necessarily have to be graphical.
As depicted in Fig. 3.8, in general, three categories of systems can be distinguished: concrete systems, symbolic systems, and conceptual systems [35], also leading to three main classes of models. A concrete model of a concrete system is called an imitation (e.g., a scale model of a car). A conceptual model of a concrete system is called a conceptualization (e.g., a process model as the conceptualization of processes). A concrete model of a conceptual system is called an implementation (e.g., a process as the implementation of a process model). A conceptual model of a conceptual system is called a conversion (e.g., the algebraic concept of a circle \((x^2 + y^2 = r^2)\) is a conversion of the geometry of its concept). A symbolic model of a conceptual system is called a formulation, and is expressed in some formal language. A conceptual model of a symbolic system is called an interpretation and is the reverse of a formulation. A symbolic model of a symbolic system is called a transformation (e.g., the transformation from Morse code to Roman notation of letters).

In enterprise architecting, a multitude of graphical and nongraphical models are needed. The set of required models spans over multiple dimensions of focus, goals, and purpose. Some examples are:

- differing levels of realization: from conceptual via logical to physical;
- differing aspects of transformation: from contextual (why) via design (where to) to the actual transformations (how);
- different aspects of a enterprises: from goals via services, products and processes to IT;
- differing levels of aggregation: from enterprise level to the level of specific (partial) processes or applications.

Even more, models referring to one specific version/alternative of an enterprise, need to be coherent, also requiring coherence between models over the above dimensions. A core driver of the ArchiMate project [78] was also to increase the coherence between different aspects and models used in an enterprise architecture. In [78], several examples are shown which illustrate the need for coherence between different models used in an enterprise architecture.

### 3.5.4 Views

The complexity of the execution of an enterprise’s strategy is likely to be immense because many processes, departments, and information systems are involved. When using enterprise architecture as a planning and steering instrument, then this instrument should reflect this complexity (the law of requisite variety [15]). As a result, it is almost undoable to make one single univocal and comprehensive set of models that can be used for all people concerned, therefore, several views are needed which focus on specific stakeholders and their concerns [78]. In Sect. 4.3, we will discuss the most common types of stakeholders involved in an architecture project. Stakeholders are important and their cooperation is necessary for a successful project,
because they are the providers of resources, most of them are influencers, some even decision-makers, and they have information about objectives and constraints. Therefore, the architectural descriptions should answer their concerns.

Different views based upon the stakeholders concerns are an important communication means to obtain the cooperation of the stakeholders. A view is a representation of a whole system from the perspective of a related set of concerns [60]. This puts the notion of a view close to the notion of a model. We actually treat a model as being a special kind of view:

1. a model is a purposeful abstraction of reality that cannot be formally derived from another model without changing the way in which the model represents the domain;
2. a view is a purposeful abstraction of reality that is derived formally from one or more models without changing the way in which the model represents the domain.

Therefore, each model is a view, but not each view is a model. As a background to these definitions, we refer to [129]; Stachowiak distinguishes between three different “model features”:

1. The mapping feature, concerned with the fact that a model is based on an original (the modeled domain).
2. The reduction feature, which deals with the fact that a model reflects a relevant selection of an original’s properties.
3. The pragmatic feature, which is concerned with the usability of the model as a placeholder for the original with respect to some purpose.

Creating a model means creating/adjusting the mapping feature of a specific model. In creating views, one makes changes to the reduction and pragmatic features, without changing the mapping feature. Changing the latter would lead to another model.

### 3.5.5 Frameworks

The (example) dimensions for models as discussed above, apply to views as well. Even more, in the case of views one typically feels the urge to introduce views that are tuned to the interests and cognitive abilities of stakeholders as well as the communication goal at hand [107, 108].

To provide architects with some structure to select views, architecture frameworks have been introduced. These frameworks intend to aid architects by providing an ontology, which uses different abstraction levels to map all kinds of information needed. Architecture frameworks position architecture results and enable diverse communication (stakeholders, detail). Often tools and best practices are included in the framework to support the work needed.
3.6 Benefits of Enterprise Architecture

In Sect. 3.3, we already discussed seven key applications for enterprise architecture: situation description, strategic direction, gap analysis, tactical planning, operational planning, selection of partial solutions, and solution architecture, enabling informed governance. We will now revisit the issue of the benefits of enterprise architecture as an instrument for informed governance, where we aim to make the benefits of enterprise architecture more explicit.

Even though a thorough scientific evaluation of the benefits of enterprise architecture is still lacking, the case for enterprise architecture has indeed been made by several market watchers, practitioners, and business visionaries. Drawing on their study of numerous companies worldwide, [118] show how constructing the right enterprise architecture enhances profitability and time to market, while it improves strategy execution. A similar line of reasoning is expressed as “To keep the business from disintegrating, the concept of information systems architecture is becoming less of an option and more of a necessity for establishing some order and control in the investment of information system resources” in [155]. Nevertheless, an initial attempt for such evaluations has been reported in [121], though we still find objective figures lacking.

3.6.1 Uses of Architectural Descriptions

It goes without saying that enterprise architecture is a means to an end. This justifiably raises the question of the benefit of enterprise architecture. We position it to be a tool or means to support strategy formulation, planning and strategy execution. In essence an enterprise architecture is a tool to manage complexity and risks. It enables informed decision-making, planning and governing of transformations. As a means it can be used:

- within strategic business/IT planning;
- to align strategic objectives and IT;
- to define and guide large scale business and/or IT transformation;
- to structure organization reengineering;
- to enable design of organizational networks (shared service centers, BPO, etc.);
- to define and monitor IT programs.

The IEEE working group on (software) architecture [60] mentions the following potential uses for architecture-models and associated descriptions:

- analysis of alternative architectures;
- business planning for transition from a legacy architecture to a new architecture;
- communications among organizations involved in the development, production, fielding, operation, and maintenance of a system;
- communications between acquirers and developers as a part of contract negotiations;
- criteria for certifying conformance of implementations to the architecture;
3.6 Benefits of Enterprise Architecture

- as development and maintenance documentation, including material for reuse repositories and training materials;
- input to subsequent system design and development activities;
- input to system generation and analysis tools;
- operational and infrastructure support; configuration management and repair; redesign and maintenance of systems, subsystems, and components;
- planning and budget support;
- preparation of acquisition documents (e.g., requests for proposal and statements of work);
- review, analysis, and evaluation of the system across the life cycle;
- specification for a group of systems sharing a common set of features, (e.g., product lines).

Even though the IEEE working group was primarily working on software architecture, the above list of uses equally well applies to descriptions produced in the case of enterprise architecture (when replacing system by enterprise in the above texts).

In [13], the Software Engineering Institute has identified the following potential uses for architectural descriptions, which can also be generalized to enterprise architecture:

- it is a vehicle for communication among stakeholders;
- it captures early design decisions, both functional aspects as well as quality aspects;
- the global structure decided upon in the architecture, also structures further development;
- it is a transferable abstraction of a system.

3.6.2 Value of Enterprise Architecture

In terms of the uses as sketched above, and taking the dashboard perspective into account, enterprise architecture can deliver value to the business in many different ways. In an attempt to make this more concrete, the following are some examples of the values that can be realized through the use of architecture [30]. To demonstrate their impact effectively, they have been categorized as specific to business, IT, or both.

3.6.2.1 Value for the Business Stakeholder

- providing a full and coherent overview and understanding of an enterprise, i.e., people, roles, processes, organization, goals, policies, rules, events, locations, etc.;
- providing an atlas and compass for management;
- business process improvement by structuring the business according to key services needed by the enterprise, based on a clear understanding of the goals/drivers of the business;
- eliminating (or resolving) enterprise duplication, enabling a move toward a “shared service” model, including identification of those services that may be better sourced externally (temporarily or permanently) [9, 10];
- underpinning decision-making on organization splitting and organization contracting [93, 94, 96];
- assess the impact of introducing a new product by determining whether the enterprise is able to deliver this product, which parts can be produced in-house (by reusing current business services) and which parts should be outsourced (or produced by using external business services) [78];
- identifying opportunities for in-sourcing, including its consequences;
- a means to ensuring business compliance and governance;
- translating strategy in executable projects.

3.6.2.2 Value for IT

- reducing solution delivery time and development costs by maximizing reuse of models and existing systems, services, and solutions;
- by conscious choices in abstraction, solutions can be designed that are either more agile for the same costs or consciously limited in their agility at a lower cost;
- reducing the risk of IT noncompliance with key regulations, especially as business becomes more regulated, e.g., Sarbanes–Oxley, etc.;
- ensure effective IT planning and management of IT roadmaps (and portfolio management), also enabling improved planning for resource skills and training and including application portfolio rationalization [94];
- implementing and managing security by design instead of reacting to breaches as they are discovered;
- delivering solutions against IT service level definitions that are linked back to real business objectives and reduce instances of costly, ill-engineered solutions.

3.6.2.3 Value for Business and IT

- improving business and IT alignment, allowing, for example, the identification of misalignment of individual projects with strategic outcome in early stages;
- ensuring alignment of data and information management with business objectives (e.g., partnerships);
- creating and maintaining a common vision of the future that is shared by both the Business and IT communities;
- ensure effective integrated change planning, reckoning with business and IT coherence.
3.6 Benefits of Enterprise Architecture

3.6.3 Added Value over Classical Approaches

It is a fair question to ask what the added value is of an architectural approach in comparison to existing approaches in which an enterprise’s strategy is translated to a program of activities, which is consequently executed. Enterprise architecture is positioned (see Fig. 3.4) between strategy and transformation program. This immediately raises questions such as: What is new about this? Can’t one do without it? In Sect. 2.5, we already surveyed traditional approaches in meeting an enterprise’s challenge. Now that we have defined what enterprise architecture is, we can identify the added value over classical approaches:

- By designing a coherent conceptualization of a solution first, one assures that programs to realize the solution are complementary to each other instead of overlapping or even incompatible;
- It enables the management to more fundamentally and explicitly underpin their decisions about the sequence of projects;
- It offers guidance and boundaries for the realization. When not applying architecture, each project will use the solution alternatives that are optimal for the project and possibly not the best for the coherent solution.

As also illustrated in Fig. 3.4, enterprise architecture does not aim to replace the classic (mostly program management) approach, but rather aims to complement it. Program management and enterprise architecture need each other. Program management cares for effectiveness and the control of time and budget. Enterprise architecture focuses on steering toward coherent solutions, aligning projects with this coherent solution, as well as setting boundaries for and providing guidance to the implementation of complex systems.

Whenever an enterprise is faced with a complex or messy problem [117] about its future organizational structure, IT support, etc., it is sensible to use architecture to gain better insight into the issues involved. By handling problems one by one, solution development becomes phased and manageable. The drawback is that solutions not necessarily match and fit together. In recent years, we have seen many examples of these mismatches in change portfolios. This yields surprises in systems management costs and users not being satisfied with their business and IT support.

By applying architecture, we treat problems in coherence. Instead of jumping to solutions right away, we develop a solution concept that takes away a fair amount of the degrees of freedom one traditionally used to have, but caters for detailed decisions made during systems development, while maintaining consistency. This sparks of interoperability, economies of scale (through common use), and possibilities for standards. It enforces a more consistent overall experience of the systems.

3.6.4 Use it Wisely

After discussing the potential added value in the previous subsection, one might wonder “is architecture the cure for every issue an enterprise has to deal with?” The simple answer is no. However, we will now add some nuance to this.
We do regard enterprise architecture as being a powerful means for management to obtain a holistic view of the enterprise and for facilitating decision-making and to set boundaries and provide guidance for implementation of complex systems. However, it is only a powerful means when it is applied properly and for the right reasons. So be aware when applying enterprise architecture that the chosen means does indeed fit the intended objectives. For example:

- Enterprise architecture typically provides management with an outlook on the coming 3 to 5 years. This outlook is rendered out of the many and various inputs the management provided themselves. Based on this outlook, management is able to plan programs for the realization of the chosen future direction. If an enterprise only aims to build a detached system with the intention to dispose of it, since it has no role of importance to play in the longer term strategy of the enterprise, one should not use enterprise architecture as a means to guide the realization of this temporary system.

- At the same time, however, one needs to beware that these disposable, short-term solutions, do not actually become permanent or even worse, become critical for the execution of the business. When such a risk does exist, either an enterprise architecture should be used after all, or measures (governance!) should be taken to ensure that the disposable system is indeed disposed of.

- Enterprise architecture is used to provide insight and to reduce risks. If a system being designed is relatively simple and risk-free, applying enterprise architecture will not provide additional insights and is, therefore, overkill.

Some additional (anonymized) examples of scenarios leading to potential failures in using enterprise architecture are:

- To use an enterprise architecture for a different means than it was developed for. As a typical example: suppose a specific enterprise architecture was developed to support the development of a business case. By nature, the business case will focus on feasibility of the proposed initiative and its cost/benefit. In this case, the enterprise architecture was designed as a high level solution and will help to find the major investment areas; as such it will not contain any guidelines for implementation. If the objective of this enterprise architecture is not clearly stated, an organization may be tempted to use this enterprise architecture to guide realization projects.

- An enterprise architecture is used after its period of validity. In some cases, an enterprise architecture has been developed, but not put into use or not been maintained. If that same enterprise architecture is again used after some time, without checking whether it is still valid, the wrong decisions will be made.

- An enterprise architecture has been designed for a regulatory use, but no measures have been implemented to monitor and control the adherence of projects to these regulations.

- A view on the enterprise architecture that was developed to communicate about the enterprise architecture to senior management, is taken to be the actual enterprise architecture (and not the underlying models).
• Any initiative—including quick wins or those to solve an immediate issue—can only be realized if they adhere to the enterprise architecture. In this case, the enterprise architecture, intended for strategic and long term initiatives, is wrongly used to restrict and complicate short term projects.

• Key decisions were made individually instead of in shared agreement with all relevant stakeholders, resulting in suboptimal and possibly overlapping solutions. Examples of this can easily be found in situations of decentralized governance, where for the same problem several solutions exist in several organizational units or (geographically defined) regions.

• Only a part of the architectural engagement was carried out. If, for instance, an aspect such as security is forgotten, it might lead to a coherent, but unsafe solution.

• One looks only from a limited perspective at the system being designed (not holistic). This might work well if it is by chance the most relevant perspective. Examples of this can be found where the IT department decides to implement new technology that does not effectively facilitate the business processes, most likely resulting in additional work in the business process as well as day-to-day irritations of the users.

• Enterprise architecture facilitates decision-making processes by providing a holistic view of the enterprise, leading to better-informed decision-making. At the same time, though, this is likely to make the decision-making process harder. This provides a challenge for the use of effective viewpoints providing decision makers with effective insight into all (and precisely all) relevant aspects affected by the decision to be made. When not using these mechanisms wisely, however, the result might be more confusion rather than more insight.

• Architecture is a means, and should not become a goal itself. One should only design the architecture at such level needed for the necessary insight and then stop. Nevertheless, numerous projects show that this risk is quite real!

### 3.7 Competencies of an Enterprise Architect

Even though Chap. 6 we provides a more detailed discussion of the skills required by architects and the challenges facing them, the discussions in this chapter already do allow us to briefly reflect on these competencies.

As mentioned before, for a proper execution of a strategy, an additional means in addition to vision, strategy and program management is needed: enterprise architecture. The means should be an unambiguous and understandable instrument supporting stakeholders in their joint decision making, setting the direction and guiding the execution. Enterprise architecture should indeed reflect the shared conceptualization of all stakeholders at a sufficiently specific level.

The enterprise architect will face the challenge of creating and applying such an instrument in a qualitative manner. For instance, his architectural description should answer the concerns of the stakeholders. To what extent, is it complete with respect to stakeholders, concerns, and answers and to what extent is completeness
feasible? In short: when is the architecture good enough from the perspective of product and process? Such an instrument should have a continuous value in steering the enterprise. Therefore, enterprise architecture needs to be embedded in the overall change and governance processes of the enterprise, as is the case in portfolio and program management. And it should be adapted to changes in the technology and business environment and concerns of stakeholders.

There is not one way of creating an enterprise architecture. Each specific situation has its own stakeholders, complexity, subject matter, and scale. The current state of the craft is that many methods, tools, and frameworks exist, both for products and processes. The quality of the enterprise architect determines the proper selection, adaptation and use of those in the specific situation. The current body of knowledge mainly exists of unrelated best practices in methods and frameworks. Current scientific developments work toward streamlining and finding common ground under successful best practices. We see enterprise architecture emerging as a new and exciting trans-discipline.

3.8 Summary

In this chapter, we started out with historical account of the term “architecture” and how it found its way from construction via computer hardware to software, and finally to the design of enterprises. To more theoretically underpin the role of enterprise architecture as an instrument for governance, we continued with a discussion of the governance paradigm. We proposed to regard enterprise architecting as a process involving a dashboard giving stakeholders indicators and controls allowing the gain insight into the current state of enterprise, alternatives for the future, as well as the performance of the transformation process(es), and to steer/direct these transformations. As a next step, we discussed seven key applications for enterprise architecture: situation description, strategic direction, gap analysis, tactical planning, operational planning, selection of partial solutions, and solution architecture, enabling informed governance.

We then went on to discuss several definitions of architectures, finally leading to an understanding of how enterprise architecture is regarded in this book. In our definition of enterprise architecture, we regard it as being combination of a regulation-oriented, design-oriented and a patterns-oriented perspective, where the design-oriented perspective is mainly pivoted toward the indicators on the dashboard and the design-oriented perspective toward the controls.

We then turned our focus to the key concepts of enterprise architecture: stakeholders, concerns, principles, models, views, and frameworks. Using the discussion of the key concepts as a background, we then revisited the potential benefits of enterprise architecture, and the potential value of architectural descriptions. We also stressed the fact that enterprise architecture is not the right answer to all situations. When applied in the wrong situation, serious negative consequences may result. We finished this chapter with a first exploration of the challenges that should be met by enterprise architects.
3.9 Discussion Statements

1. Enterprise architecture is answering the needs that business administration and organizational design should address! Even more, we should add this knowledge to the body of knowledge for business administration and organizational design.

2. Enterprise architecture should restrain itself to the regulation-oriented perspective; all other work is simply high-level design. There is no need for yet another concept.

3. Enterprise architecture is an enjoyable intellectual exercise. However, in the end, senior management will take their decisions mainly based on their business feeling and intuition.

4. Full business/IT alignment can not be reached without the use of enterprise architecture.

5. The instruments of the enterprise architecture (principles, models, views, frameworks) are far less important than the process involved-in/concerning enterprise architecture.

6. The creation and use of an enterprise architecture takes too long and, therefore, falls in exactly the same pitfall as “endless strategy formulation.”

7. A project which aims to realize a part of an enterprise architecture does not need an enterprise architect. The enterprise architecture itself provides all the guidance needed.

8. Architecture principles are useless since they are not specific enough. A real design is much more useful as it makes the final result much more tangible.
Enterprise Architecture at Work

Modelling, Communication, and Analysis
7 Viewpoints and Visualisation .................................................................147
  7.1 Architecture Viewpoints ...............................................................147
    7.1.1 Origin of Viewpoints ...........................................................148
    7.1.2 Architecture Viewpoints .....................................................149
    7.1.3 Viewpoint Frameworks .......................................................150
  7.2 Models, Views, and Visualisations .............................................152
    7.2.1 Example: Process Illustrations ............................................154
    7.2.2 Example: Landscape Maps ..................................................155
  7.3 Visualisation and Interaction .....................................................157
    7.3.1 Actions in Views ...............................................................158
  7.4 Creating, Selecting, and Using Viewpoints ................................161
    7.4.1 Classification of Viewpoints ..............................................161
    7.4.2 Guidelines for Using Viewpoints ........................................165
    7.4.3 Scoping .................................................................165
    7.4.4 Creation of Views ...........................................................166
    7.4.5 Validation .................................................................167
    7.4.6 Obtaining Commitment ....................................................168
    7.4.7 Informing Stakeholders ....................................................169
  7.5 Basic Design Viewpoints ..........................................................170
    7.5.1 Introductory Viewpoint ......................................................173
    7.5.2 Organisation Viewpoint .....................................................175
    7.5.3 Actor Cooperation Viewpoint .............................................175
    7.5.4 Business Function Viewpoint .............................................177
    7.5.5 Product Viewpoint ..........................................................178
    7.5.6 Service Realisation Viewpoint ..........................................179
    7.5.7 Business Process Cooperation Viewpoint ..............................180
    7.5.8 Business Process Viewpoint ..............................................181
    7.5.9 Information Structure Viewpoint ........................................182
    7.5.10 Application Cooperation Viewpoint ...................................182
    7.5.11 Application Usage Viewpoint ..........................................184
    7.5.12 Application Behaviour Viewpoint ......................................185
    7.5.13 Application Structure Viewpoint ......................................186
    7.5.14 Infrastructure Viewpoint .................................................186
    7.5.15 Infrastructure Usage Viewpoint .......................................187
    7.5.16 Implementation & Deployment Viewpoint ............................188
  7.6 Summary .................................................................189
Establishing and maintaining a coherent enterprise architecture is clearly a complex task, because it involves many different people with differing backgrounds using various notations. In order to get to grips with this complexity, researchers have initially focused on the definition of architectural frameworks for classifying and positioning the various architecture descriptions with respect to each other. A problem with looking at enterprise architecture through the lens of an architectural framework is that it categorises and divides architecture descriptions rather than providing insight into their coherence.

To integrate the diverse architecture descriptions, we advocate an approach in which architects and other stakeholders can define their own views of the enterprise architecture. In this approach views are specified by viewpoints. Viewpoints define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns. Viewpoints can be used both to view certain aspects in isolation, and for relating two or more aspects.

This chapter focuses on the use of views of enterprise architectures, both to create and manipulate architectural models and to give others insight into the architectures being describe. We describe the use of viewpoints in communication, and the distinction between an architecture model, a view of that model, and its visualisation and manipulation. We give guidelines for the selection and use of viewpoints, and we outline a number of viewpoints on the ArchiMate language that can be used by architects involved in the creation or change of enterprise architecture models.

7.1 Architecture Viewpoints

In this section we discuss the notion of views and viewpoints as basic tools in communicating about architectures. In the context of enterprise architectures, a viewpoint is typically used for activities like design, analysis, obtaining commitment, formal decision making, etc. As we argued in Chap. 4, we regard all of these activities to be communicative in nature.
As defined in Sect. 3.2.4, a viewpoint essentially prescribes the concepts, models, analysis techniques, and visualisations that are to be used in the construction of different views of an architecture description. A view is typically geared towards a set of stakeholders and their concerns. Simply put, a view is what you see, and a viewpoint describes from where you are looking.

In discussing the notion of viewpoint, we will first provide a brief overview of the origin of viewpoints. This is followed by a more precise definition of viewpoints, and the concept of viewpoint frameworks.

### 7.1.1 Origin of Viewpoints

The concept of viewpoint is not new. For example, in the mid 1980s, Multiview (Wood-Harper et al. 1985) already introduced the notion of views. In fact, Multiview identified five viewpoints for the development of (computerised) information systems: Human Activity System, Information Modelling, Socio-Technical System, Human–Computer Interface, and the Technical System. During the same period in which Multiview was developed, the so-called CRIS Task Group of IFIP Working Group 8.1 developed similar notions, where stakeholder views were reconciled via appropriate ‘representations’. Special attention was paid to disagreement about which aspect (or perspective) was to dominate the system design (namely, ‘process’, ‘data’, or ‘behaviour’). As a precursor to the notion of concern, the CRIS Task Group identified several human roles involved in information system development, such as executive responsible, development coordinator, business analyst, business designer (Olle et al. 1988).

The use of viewpoints is not limited to the information systems community, it was also introduced by the software engineering community. In the 1990s, a substantial number of software engineering researchers worked on what was phrased as ‘the multiple perspectives problem’ (Finkelstein et al. 1992, Kotonya and Sommerville 1992, Nuseibeh 1994, Reeves et al. 1995). By this term, the authors referred to the problem of how to organise and guide (software) development in a setting with many actors, using diverse representation schemes, having diverse domain knowledge, and using different development strategies. A general framework has been developed in order to address the diverse issues related to this problem (Finkelstein et al. 1992, Kotonya and Sommerville 1992, Nuseibeh 1994). In this framework, a viewpoint combines the notion of actor, role, or agent in the development process with the idea of a perspective or view which an actor maintains. A viewpoint is more than a partial specification; in addition, it contains partial knowledge of how further to develop that partial
specification. These early ideas on viewpoint-oriented software engineering have found their way into the IEEE 1471 standard for architecture description (IEEE Computer Society 2000) on which we have based our definitions below.

7.1.2 Architecture Viewpoints

In the context of architecture, viewpoints provide a means to focus on particular aspects of an architecture description. These aspects are determined by the concerns of the stakeholders with whom communication takes place. What should and should not be visible from a specific viewpoint is therefore entirely dependent on argumentation with respect to a stakeholder’s concerns. Viewpoints are designed for the purpose of serving as a means of communication in a conversation about certain aspects of an architecture. Though viewpoints can be used in strictly uni-directional, informative conversations, they can in general also be used in bi-directional classes of conversations: the architect informs stakeholders, and stakeholders give their feedback (critique or consent) on the presented aspects. What is and what is not shown in a view depends on the scope of the viewpoint and on what is relevant to the concerns of the stakeholders. Ideally, these are the same, i.e., the viewpoint is designed with the specific concerns of a stakeholder in mind. Relevance to a stakeholder’s concern, therefore, is the selection criterion that is used to determine which objects and relations are to appear in a view.

Below we list some examples of stakeholders and their concerns, which could typically serve as the basis for the definition/selection of viewpoints:

- Upper-level management: How can we ensure our policies are followed in the development and operation of processes and systems? What is the impact of decisions (on personnel, finance, ICT, etc.)? Which improvements can a new system bring to a pre-existing situation in relation to the costs of acquiring that system?
- Middle-level management: What is the current situation with regards to the computerised support of a business process?
- End user: What is the potential impact of a new system on the activities of a prospective user?
- Architect: What are the consequences for the maintainability of a system with respect to corrective, preventive, and adaptive maintenance?
- Operational manager: What new technologies do we need to prepare for? Is there a need to adapt maintenance processes? What is the impact of changes to existing applications? How secure are the systems?
- Project manager (of system development project): What are the relevant domains and their relations? What is the dependence of business processes on the applications to be built? What is their expected performance?
- System developer: What are the modifications with respect to the current situation that need to be performed?
- System administrators: What is the potential impact of a new system on the work of the system administrators that are to maintain the new system?

In line with the IEEE 1471 standard, and based on the detailed definition given in Proper (2004) we define a viewpoint as follows:

| Viewpoint: a specification of the conventions for constructing and using views. |

This should also involve the various ‘ways of …’ that we outlined in Sect. 3.2.5, but in this chapter we will focus on the selection of the content of views, the visual representation of this content, and the typical use of these viewpoints, i.e., on the ways of modelling, communicating, and using. The ‘way of supporting’, i.e., tool support for views, will be addressed in Chap. 10, and the ‘way of working’ has already been addressed in Chap. 6.

### 7.1.3 Viewpoint Frameworks

In the context of architecture descriptions, a score of viewpoint frameworks exists, leaving designers and architects with the burden of selecting the viewpoints to be used in a specific situation. Some of these frameworks of viewpoints are: the Zachman framework (Zachman 1987), Kruchten’s 4+1 view model (Kruchten 1995), RM-ODP (ITU 1996), and TOGAF (The Open Group 2002). These frameworks have usually been constructed by their authors in an attempt to cover all relevant aspects/concerns of the architecture of some class of systems. In practice, numerous large organisations have defined their own frameworks of viewpoints by which they describe their architectures. We shall discuss two of these framework in more detail below.

#### The ‘4+1’ View Model

Kruchten (1995) introduced a framework of viewpoints (a view model) comprising five viewpoints. The use of multiple viewpoints is motivated by the observation that it ‘allows to address separately the concerns of the
various stakeholders of the architecture: end-user, developers, systems engineers, project managers, etc., and to handle separately the functional and non-functional requirements’.

The goals, stakeholders, concerns, and meta-model of the 4+1 framework can be presented, in brief, as in Table 7.1. Note that in Kruchten (2000), the viewpoints have been renamed; physical viewpoint → deployment viewpoint, development viewpoint → implementation viewpoint, and scenario viewpoint → use-case viewpoint, better to match the terminology of UML.

The framework proposes modelling concepts (the meta-model) for each of the specific viewpoints. It does so, however, without explicitly discussing how these modelling concepts contribute to the goals of the specific viewpoints. One might, for example, wonder whether object classes, associations, etc., are the right concepts for communication with end users about the services they require from the system. The 4+1 framework is based on experiences in practical settings by its author.

Table 7.1. Kruchten’s ‘4+1’ view model.

<table>
<thead>
<tr>
<th>Viewpoint</th>
<th>Logical</th>
<th>Process</th>
<th>Development</th>
<th>Physical</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Capture the services which the system should provide</td>
<td>Capture concurrency and synchronisation aspects of the design</td>
<td>Describe static organisation of the software and its development</td>
<td>Describe mapping of software onto hardware, and its distribution</td>
<td>Provide a driver to discover key elements in design Validation and illustration</td>
</tr>
<tr>
<td><strong>Stakeholders</strong></td>
<td>Architect</td>
<td>End users</td>
<td>Architect System designer</td>
<td>Architect Developer Manager</td>
<td>Architect System designer</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td>Functional- ity</td>
<td>Performance Availability Fault tolerance</td>
<td>Organisation Reuse Portability</td>
<td>Scalability Performance Availability</td>
<td>Understand- ability</td>
</tr>
<tr>
<td><strong>Meta-model</strong></td>
<td>Object classes Associations Inheritance</td>
<td>Event Message Broadcast</td>
<td>Module Subsystem Layer</td>
<td>Processor Device Bandwidth</td>
<td>Objects Events Steps</td>
</tr>
</tbody>
</table>

**RM-ODP**

The Reference Model for Open Distributed Processing (RM-ODP) (ITU 1996) was produced in a joint effort by the international standard bodies
ISO and ITU in order to develop a coordinating framework for the standardisation of open distributed processing. The resulting framework defines five viewpoints: enterprise, information, computation, engineering and technology. The modelling concepts used in each of these views are based on the object-oriented paradigm.

The goals, concerns, and associated meta-models of the viewpoints identified by the RM-ODP can be presented, in brief, as in Table 7.2.

<table>
<thead>
<tr>
<th>Viewpoint</th>
<th>Enterprise</th>
<th>Information</th>
<th>Computational</th>
<th>Engineering</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Capture purpose, scope, and policies of the system</td>
<td>Capture semantics of information and processing performed by the system</td>
<td>Express distribution of the system in interacting objects</td>
<td>Describe design of distribution-oriented aspects of the system</td>
<td>Describe choice of technology used in the system</td>
</tr>
<tr>
<td>Concerns</td>
<td>Organisational requirements and structure</td>
<td>Information and processing required</td>
<td>Distribution of system</td>
<td>Distribution of the system, and mechanisms and functions needed</td>
<td>Hardware and software choices</td>
</tr>
<tr>
<td>Meta-model</td>
<td>Objects, Communities, Permissions, Obligations, Contract</td>
<td>Objects, Classifiers, Associations, Process</td>
<td>Objects, Interfaces, Interaction, Activities</td>
<td>Objects, Channels, Node, Capsule, Cluster</td>
<td>Not stated explicitly</td>
</tr>
</tbody>
</table>

RM-ODP provides a modelling language for each of the viewpoints identified. It furthermore states: ‘Each language [for creating views/models conforming to a viewpoint] has sufficient expressive power to specify an ODP function, application or policy from the corresponding viewpoint.’ RM-ODP does not explicitly associate viewpoints to a specific class of stakeholders. This is left implicit in the concerns which the viewpoints aim to address.

7.2 Models, Views, and Visualisations

An important principle in our approach is the separation of the content and the presentation or visualisation of a view. This separation is not explicitly made in the IEEE standard, but it has important advantages. It facilitates the use of different visualisation techniques on the same modelling con-
cepts, and vice versa. Operations on the visualisation of a view, e.g., changing its layout, need not change its content.

The view content, referred to as the ‘view’ in the remainder of this chapter, is a selection or derivation from a (symbolic) model of the architecture, and is expressed in terms of the same modelling concepts. The presentation or notation of this view, referred to as ‘visualisation’ in the remainder, can take many forms, from standard diagrams to tables, cartoons, or even dynamic visualisations like movies. Editing operations on this visualisation can lead to updates of the view and of the underlying model. The creation and update of both the view and the visualisation are governed by a viewpoint. This viewpoint is jointly defined and/or selected in an iterative process by architect and stakeholder together. This is illustrated in Fig. 7.1.

![Diagram of Model, View, Visualisation, and Viewpoint](image)

**Fig. 7.1.** Separation of concerns: model, view, visualisation, and viewpoint.

The separation between view and visualisation is based on the notion of ‘meaning’. In Chap. 3 we introduced the concept of the signature of an architecture as its alphabet: that is, the set of symbols used to describe the concepts of the architecture and the relations among these concepts. This idea can also be used to clarify the distinction between view and its visualisation. A further discussion of these formal foundations can be found in Chap. 8.

A view stripped from its visual properties can be formalised just like any other model, e.g., by defining its signature, as outlined in Chap. 3. By formalising its relation with an underlying model, a view’s quality and consistency can be greatly enhanced and new opportunities for its use may arise, e.g., in changing the underlying models by interacting with such a view.
7.2.1 Example: Process Illustrations

To illustrate the difference between a view and its visualisation, we introduce the *process illustration* viewpoint. This viewpoint illustrates a process model in an informal way for employees and managers. A process illustration is derived from a model of the architecture using a set of translation and abstraction rules. As process illustrations are meant for communicating the coherence between business processes, they typically abstract from details regarding the applications and technology involved. Moreover, process illustrations do not apply abstract concepts and notations, but rather use recognisable terms and intuitive notations.

A process illustration of the Car Tax Collection process is depicted in Fig. 7.2. The figure shows the various subprocesses involved and the information flows between them. The figure is derived from an ArchiMate model via a series of translation and abstraction rules, for instance to replace abstract shapes with meaningful symbols, abstract from complex relations, and visually group all objects and relations that belong to or happen within a certain actor.

---

**Fig. 7.2.** Process illustration of the Car Tax Collection process.
In Fig. 7.3 you can see a number of presentation rules that can be applied in the ‘model-to-illustration’ derivation. The basic idea behind these rules is to find suitable and intuitive graphic symbols that will replace ArchiMate shapes. These rules apply to ArchiMate concepts for which there is an immediate correspondent in the process illustration notation (i.e., actor, role, device, service, business object, etc.).

Of course, many other rules can be added here. For instance, rules referring to a specific layout of the final drawing or to the more extensive usage of 3D graphic symbols can increase the readability and usability of the final drawing.

7.2.2 Example: Landscape Maps

A more complex example to illustrate the differences between a model, a view, and its visualisation, is the landscape map viewpoint. Landscape maps, as defined in van der Sanden and Sturm (1997), are a technique for visualising enterprise architectures. They present architectural elements in the form of an easy-to-understand 2D ‘map’. A landscape map view of architectures provides non-technical stakeholders, such as managers, with a high-level overview, without burdening them with the technicalities of architectural drawings.
Many systems used by many processes realising various products and services comprise too much detail to display in a single figure. This is a typical example of where landscape maps can help. In Fig. 7.4, a landscape map is depicted that shows which information systems support the operations of our fictitious insurance company ArchiSurance. The vertical axis represents the company’s business functions; the horizontal axis shows its insurance products. An application rectangle covering one or more cells means that this particular function/product pair is supported by the application, e.g., contracting of a legal aid insurance is supported by the legal aid back-office system. The visualisation chosen makes it immediately obvious to the viewer that there is (possibly unwanted) overlap between applications, as is the case in the Car insurance application and the Legal Aid CRM system. Clearly, landscape maps are a richer representation than cross-reference tables, which cover only two dimensions. In order to obtain the same expressive power of a landscape map two cross-reference tables would be necessary; but even then, you would get a presentation that is not as insightful and informative as a landscape map.

The dimensions of the landscape maps can be freely chosen from the architecture that is being modelled. In practice, dimensions are often chosen from different architectural domains, for instance business functions, products and applications, etc. In most cases, the vertical axis represents
behaviour such as business processes or functions; the horizontal axis represents ‘cases’ for which those functions or processes must be executed. These ‘cases’ can be different products, services, market segments, or scenarios. The third dimension represented by the cells of the matrix is used for assigning resources like information systems, infrastructure, or human resources.

The visualisation of architecture models as landscape maps is based on architecture relations. The dimensions that are used in the landscape maps determine which relations are used. For instance, the landscape map in Fig. 7.4 relates business functions (Contracting, Claim Handling, etc.) to products (Home insurance, Travel insurance, etc.) to applications (Web portal, Car insurance application, etc.). The relation between business functions, products, and applications is not directly supported by relations in the underlying model. Rather, this needs to be inferred indirectly: a product comprises a number of business services, which are realised by business processes and functions, which use (the application services of) application components. For this inference, the formalisation of the underlying symbolic models and the rules for the composition of relations described in Chaps. 3 and 4 are indispensable.

For landscape maps to be of practical use, the visualisation must be intuitive and easy to understand. To a large extent, the choice of the axes and the ordering of the rows and columns determine the layout of a landscape map. If adjacent cells in the plane have the same value assigned, they can be merged to form a single shape. If there are no other criteria for ordering the axes such as time or priority, changes to the ordering can be used to optimise the layout of shapes in the plane, and also to limit their number. Various layout optimisation algorithms can be employed, and user manipulation of, for example, the order of rows and columns may also help in creating a pleasing visualisation.

Summarising, in developing the landscape map viewpoint, it has been fruitful to distinguish the operation on the model from the visualisation of the view, because they are completely different concerns. The same holds for the other viewpoints we have defined. To separate these concerns, views have to be distinguished from their visualisation.

7.3 Visualisation and Interaction

The distinction we make between a model and its visualisation naturally leads to the concept of interactive visualisation; that is, visualisation which can change the model due to interaction with a stakeholder. Interaction has traditionally been considered as something completely outside the model
and the view. Interaction is at least partly a visualisation issue: for example, when a user draws an object on the canvas of some tool. However, it can also partly be defined as part of the model and view, since the object the user draws may be put in the underlying model or view as well.

These two considerations have led to a new visualisation and interaction model for enterprise architectures in ArchiMate. Its goal is that interaction is separated from updating the model, or from its visualisation.

### 7.3.1 Actions in Views

The effect of a user interacting with the visualisation can be an update of the view. But where will this be defined? Clearly, the visualisation itself is ‘dumb’ and does not know about the semantics of the view. Hence, rules for changing the view cannot be tied to the visualisation and must be defined in the view itself. This is why we introduce the notion of actions in views. Consider for example a landscape map view, and a user who interacts with this view by moving an application to another business function. Does the relation between the interaction with the landscape map and the update of the model mean something? Obviously the relation between the move in the landscape map leads to an update of the underlying model or view, and thus means something.

In Sect. 6.2.3 we have identified a number of basic modelling actions, such as introducing, refining, abandoning, abstracting, and translating a concept in a model. These actions operate on the architecture model or view, not on its visualisation. However, most changes to a model will be conducted by a user who changes a visualisation of that model. Hence, we need to define the ways in which a user can manipulate these visualisations and the effects on the underlying model in terms of these basic modelling actions. We can then relate these actions to the manipulations of the visualisation by making the actions part of the view being visualised.

Thus, a clear separation of model and visualisation leads to a separation of concerns in tool building. An extremely generic visualisation engine can be constructed that does not need to know about the semantics of the models it displays. If we define the possible actions together with the views, a generic editor can be configured by this set of actions.

The actions in views should be defined in terms of the effects they have on elements of the underlying model. For example, consider a view of a business process model, and an action that merges two processes into a single process. Issues that are relevant for the action of merging processes are the effects of the merger: for example, the removal of processes, addi-
tion of a new process, transferring some relations from an old, removed process to a new process.

For each viewpoint, we define a set of actions. For example, for the landscape map viewpoint we define the move of an application to another cell, we define changing the columns and rows of the matrix, and we define the addition and deletion of applications. Moreover, we must determine for each action which parameters it needs as input, and define the consequences of executing the action.

When actions for each view have been defined, we can go one step further and define the relation between actions. One important relation is that one action may consist of a set of simpler actions. For example, consider an architect or stakeholder that wishes to change an existing landscape map. First the effects of this change on the underlying model need to be assessed. Some changes may be purely ‘cosmetic’ in nature, e.g., changing the colour of an object. Other changes need to be propagated to the underlying model by invoking one of the basic modelling actions of Sect. 6.2.3, e.g., if an object is added or deleted.

Mapping a seemingly simple change to the map onto the necessary modifications of the model may become quite complicated. Since a landscape map abstracts from many aspects of the underlying model, such a mapping might be ambiguous: many different modifications to the model might correspond to the same change of the landscape map. Human intervention is required to solve this, but a landscape map tool might suggest where the impact of the change is located.

In the example of Fig. 7.4, you may for instance want to remove the seemingly redundant Legal aid CRM system by invoking a ‘remove overlap’ operation on this object. This operation influences both the visualisation and the architectural model. The effects of the operation on the underlying model are shown in Fig. 7.5. First, you select the object to be removed, in this case the Legal Aid CRM system. The envisaged tool colours this object and maps it back onto the underlying object in the architecture. Next, the relations connecting this object to its environment are computed, possibly using the impact-of-change analysis techniques described in Chap. 8 (the second part of Fig. 7.5). Here, this concerns the relations of Legal Aid CRM to the Web portal and the Legal Aid back-office system. These relations will have to be connected to one or more objects that replace the objects that are to be removed. Since we have chosen a ‘remove overlap’ operation, the landscape tool computes with which other objects Legal Aid CRM overlaps, in this case the CRM system. The relations formerly connecting Legal Aid CRM are then moved to the other CRM system, unless these already exist (e.g., the relation with the Web portal).
Fig. 7.5. Editing a landscape map.
Naturally, this scenario presents an ideal situation with minimal user intervention. In reality, a tool cannot always decide how a proposed change is to be mapped back onto the model, and may only present the user with a number of options. For example, if the functionality of the Legal Aid CRM system overlaps with more than one other system, remapping its relations requires knowledge about the correspondence between these relations and the functions realised by these other systems.

Implementing a tool that realises this ‘actions in views’ concept is not a trivial task. In Chap. 10, we will describe the design of a prototype tool that provides a proof of concept of these ideas.

7.4 Creating, Selecting, and Using Viewpoints

It is interesting to note that both of the discussed frameworks of viewpoints (Sect. 7.1.3) do not provide an explicit motivation for their choice regarding the modelling concepts used in specific viewpoints. When using one of the two frameworks, architects will not find it difficult to select a viewpoint for the modelling task at hand. However, this ‘ease of choice’ is more a result of the limitation of the selections of options available (one is limited to the number of viewpoints provided by the framework) than the result of a well-motivated choice about the viewpoint’s utility towards the tasks at hand.

One should realise that a well-integrated set of viewpoints (such as the ArchiMate viewpoints) brings more (utility!) to a development project than the sum of its parts! Among other things, it allows views to be more easily related and integrated into a consistent whole. However, defining such an integrated viewpoint framework is an expensive undertaking. This means that even though a pre-existing (off-the-shelf) viewpoint framework may not be the ideal answer to an architect’s specific communication needs, the alternative strategy of defining a tailor-made viewpoint framework for each development project is likely to be too costly. Hence our attention to defining ‘ad hoc’ viewpoints relative to a predefined modelling language (i.e., meta-model) as a compromise between fixed viewpoints and free viewpoints.

7.4.1 Classification of Viewpoints

As we can see from the list of stakeholders in Sect. 7.1.2, an architect is confronted with many different types of stakeholders and concerns. To help the architect in selecting the right viewpoints for the task at hand, we
introduce a framework for the definition and classification of viewpoints and views. The framework is based on two dimensions, *purpose* and *content*. The following three types of architecture support define the purpose dimension of architecture views (Steen et al. 2004):

- **Designing**: Design viewpoints support architects and designers in the design process from initial sketch to detailed design. Typically, design viewpoints consist of diagrams, like those used in UML.
- **Deciding**: Decision support views assist managers in the process of decision making by offering an insight into cross-domain architecture relations, typically through projections and intersections of underlying models, but also by means of analytical techniques. Typical examples are cross-reference tables, landscape maps, lists, and reports.
- **Informing**: These viewpoints help to inform any stakeholder about the enterprise architecture, in order to achieve understanding, obtain commitment, and convince adversaries. Typical examples are illustrations, animations, cartoons, flyers, etc.

The goal of this classification is to assist architects and others to find suitable viewpoints given their task at hand, i.e., the purpose that a view must serve and the content it should display. With the help of this framework, it is easier to find typical viewpoints that might be useful in a given situation. This implies that we do not provide an orthogonal categorisation of each viewpoint into one of three classes; these categories are not exclusive in the sense that a viewpoint in one category cannot be applied to achieve another type of support. For instance, some decision support viewpoints may be used to communicate to any other stakeholders as well.

![Fig. 7.6. Elements of an enterprise architecture.](image-url)
For characterising the content of a view we define the following abstraction levels:

- **Details**: Views of the detailed level typically consider one layer and one aspect from the framework that was introduced in Chap. 5 (Fig. 7.6). Typical stakeholders are a software engineer responsible for the design and implementation of a software component or a process owner responsible for effective and efficient process execution. Examples of views are a BPMN process diagram and a UML class diagram.

- **Coherence**: At the coherence abstraction level, multiple layers or multiple aspects are spanned. Extending the view to more than one layer or aspect enables the stakeholder to focus on architecture relations like process–use–system (multiple layer) or application–uses–object (multiple aspect). Typical stakeholders are operational managers responsible for a collection of IT services or business processes.

- **Overview**: The overview abstraction level addresses both multiple layers and multiple aspects. Typically such overviews are addressed to enterprise architects and decision makers such as CEOs and CIOs.

In Fig. 7.7, the dimensions of purpose and abstraction level are visualised in a single picture, together with examples of stakeholders. Table 7.3 and Table 7.4 summarise the different purposes and abstraction levels.

![Fig. 7.7. Classification of enterprise architecture viewpoints.](image-url)
Table 7.3. Viewpoint purpose.

<table>
<thead>
<tr>
<th>Typical stakeholders</th>
<th>Purpose</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Designing</strong></td>
<td>Architect, software developer, business process designer</td>
<td>Navigate, design, support design decisions, compare alternatives</td>
</tr>
<tr>
<td><strong>Deciding</strong></td>
<td>Manager, CIO, CEO</td>
<td>Decision making</td>
</tr>
<tr>
<td><strong>Informing</strong></td>
<td>Employee, customer, others</td>
<td>Explain, convince, obtain commitment</td>
</tr>
</tbody>
</table>

Table 7.4. Viewpoint abstraction levels.

<table>
<thead>
<tr>
<th>Typical stakeholders</th>
<th>Purpose</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Details</strong></td>
<td>Software engineer, process owner</td>
<td>Design, manage</td>
</tr>
<tr>
<td><strong>Coherence</strong></td>
<td>Operational managers</td>
<td>Analyse dependencies, impact of change</td>
</tr>
<tr>
<td><strong>Overview</strong></td>
<td>Enterprise architect, CIO, CEO</td>
<td>Change management</td>
</tr>
</tbody>
</table>

The landscape map viewpoint described in Sect. 7.2.1 is a typical example of a decision support view, which give a high-level overview and can, for example, be used to identify redundancies or gaps in the application landscape of an enterprise.

The process illustration viewpoint described in Sect. 7.2.1 is an example of a viewpoint intended for ‘informing’ others. It depicts workflows in a cartoon-like fashion, easily readable for employees and managers. Process illustrations can be on the detailed, coherence, or overview abstraction level.

To assist the architect in designing an enterprise architecture, we present a set of basic design viewpoints in the next sections. These viewpoints are all diagrams for designing architectures. Some viewpoints are multiple-aspect and multiple-layer overviews at the ‘coherence’ level of abstraction, while others are at the ‘details’ level.
7.4.2 Guidelines for Using Viewpoints

To help you in selecting and using viewpoints for tasks at hand, we present a number of guidelines, based on our own experience and interviews with architects from practice.

In general, the use of an architectural viewpoint will pass through a number of phases. These phases roughly are:

1. **Scoping**: Select one or more appropriate viewpoints, select the (sub-)domain that needs to be represented or modelled, and determine the constraints that apply to the domain being modelled.

2. **Creation of views**: Create or select the actual content of the viewpoint, i.e., create or select a view conforming to the viewpoint used. This can pertain to the selection of a part of the larger (pre-existing) architecture model, or the creation or refinement of a part of the architecture model (in terms of a view).

3. **Validation**: Validate the resulting view. Do the stakeholders agree that the view is a correct representation of the actual or intended situation?

4. **Obtaining commitment**: If agreement has been reached among the key stakeholders involved, the next step will be to create commitment for the results. In other words, do the stakeholders commit themselves to the (potential) impact of what is described by the view?

5. **Informing**: Inform other stakeholders of the results. These stakeholders will be those members of the development community, whose explicit commitment has, in a conscious decision, been considered not to be crucial.

Note that these phases will not necessarily be executed in a linear order. Practical circumstances usually dictate a more evolutionary approach. The viewpoints to be used for architectural communication will have to support the activities of each of the phases. The guidelines resulting from the interviews are divided over them. They are discussed in the next sections.

7.4.3 Scoping

The importance of focusing on the concerns of stakeholders, and the extent to which a specific view(point) addresses these concerns, was confirmed by the outcomes of the interviews. When you communicate with business managers, you only need those views or models that enable a discussion of factors deserving special attention. Typically these are factors that have a high impact if they fail and also have a high risk of indeed failing. For communication with the actual software developers, on the other hand, more detailed models are crucial.
The selection of viewpoints should be done consciously and based on rational considerations. Furthermore, architects state that this decision, and its rationalisation, must be readily available. It is quite possible that a stakeholder (usually a technology-oriented one) will ask for more detail in a model than you can give, or want to give, in that particular phase of the project. An architect should be prepared to clarify better the goals of the particular model and phase, and why the requested details are not yet relevant (or even harmful).

Determining the constraints that should guide the ensuing creation phase is also considered to be important. Numerous IT projects suffer from the problem that designers have too much ‘design freedom’ when producing a model of a desired future system. This increases the risk of ending up with lengthy design processes. Limiting design freedom by means of architecture principles, a higher-level architecture, or any other means, reduces this risk considerably.

7.4.4 Creation of Views

During the creation of a view, in particular when it involves actual modelling, you should try to put a limit on the number of participants in a conversation. Graphical models may or may not be used in communication with stakeholders, but most actual modelling is done by individuals (or two people at most). Genuine group modelling sessions are very rare.

During the early stages of system design, it is often considered bad to ‘think’ in terms of ‘solutions’. However, when detailed modelling takes place in a cooperative setting, give informants some room to think in terms of ‘solutions’ even if pure requirements thinking (what, not how) does not officially allow for this. Most people just think better in terms of concrete solutions; it is a vital part of their creativity. Just be sure that requirements thinking is returned to in due course. In general, when you discuss models with stakeholders and informants, in particular when you try to establish a common understanding, you should discuss different scenarios and alternatives to the model being considered. Doing so leads to an exploration of the meaning and impact of the model taking shape, and also leads to improved mutual understanding.

The graphical notation that is part of a viewpoint should be approached flexibly when it comes to communicating with non-technical stakeholders. If people are not used to or prepared to deal with abstract graphical models, do not use them. Use other forms of visualisation, text, or tables. Iconised diagrams work particularly well. However, be prepared to point out
the relation between the alternative visualisation and your abstract models if asked to.

Even if graphical models play a big role in architecture, text is the chief form in which (written) communication takes place. Two main ways in which this occurs are:

- Graphical (partial!) models that are used to support textual descriptions (‘illustration by diagram’).
- Text explaining and elaborating on a graphical model (‘textual modelling’).

In fact, text is often better than a graphical model for conveying large amounts of detail.

Language studies have indeed shown how the specific form of a language does have an impact on what is expressed by means of the language (Cruse 2000). In the case of modelling languages, the modelling concepts offered by the language will, in general, influence the level of detail or abstraction that the resulting models will exhibit.

Finally, during a modelling session, several things may come to the fore that will influence the further process. External events may occur that are a threat to the process as a whole. Be prepared to stop modelling if executive commitment is withdrawn. It may be frustrating, but from a business perspective it may also be crucial. It is simply part of a flexible project setup. If the informants turn out to be less informed than expected, it is better to stop than to try and ‘make the best of it’ and produce an ill-conceived model.

In the field of agile development (Martin 2002, Rueping 2003, Ambler 2002), a refreshing perspective can be found on such considerations.

### 7.4.5 Validation

In validation of an architecture with stakeholders, a clear difference should be made between validation of content (qualitative validation, by modellers and experts) and validation in terms of commitment (by executives). Both are crucial, but very different. Obtaining (and validating) commitment is discussed in the next subsection.

Whether good mutual communication and understanding about a model is being reached is often a matter of intuition. If the people involved have a mutual feeling that ‘their thoughts are well in sync’, then dare to trust that feeling. However, if the opposite is the case, be prepared to invest in substantial discussion of concrete examples, or face the dire consequences of poor validation. If the required ‘level of agreement’ between participants is
high, an atmosphere of mutual trust and cooperation between these participants is crucial.

Validation is an activity that should be conducted in limited groups. ‘Feedback rounds’ involving a larger number of people, by e-mail or printed documentation, do not really work. If you want feedback that is worth something, find key people and discuss the models/views, preferably face to face. Make sure the ‘opinion leaders’ in an organisation agree to the model.

Also, you should take care that the languages used to express a view do not have a wrong connotation that may result in incorrect impressions about the scope and status of models. A language like UML cannot be used in a discussion with business people. Even though the language is suitable to express the models, the notation has an implementation-oriented connotation to this audience.

Furthermore, do not show a concrete view of the desired system too early on in the development process. The concreteness of the diagram may give the stakeholders a feeling that important decisions have already been made.

With regards to the last observation, an interesting statement on this issue can also be found in Weinberg (1988). He argues that when the design of a system, or a model in general, is still in its early stages, and different aspects are not yet clear and definite, the graphical notation used should also reflect this. He suggests using squiggly lines rather than firm lines, so as to communicate to the reader of a view that specific parts of the view are still open to debate. We use this principle in the Introductory viewpoint discussed in Sect. 7.5.1.

7.4.6 Obtaining Commitment

Obtaining commitment for a specific architectural design involves obtaining commitment for the impact of this design on the future system and its evolution, as well as the costs/resources needed to arrive at this future system. This means that the message that one needs to get across to the stakeholders involves:

− What are the major problems in the current situation?
− How bad are these problems (to the concerns and objectives of the stakeholders)?
− How will this improve in the new situation? (Benefits!)
− At what costs will these improvements come?
When discussing costs and benefits with stakeholders, make these costs and benefits as SMART (Specific, Measurable, Attainable, Realisable, and Time-bound) as possible. Make sure that the stakeholders agree, up front, with the criteria that are used to express/determine costs and benefits. It is their commitment that is needed. They will be the judge. Let them also decide what they want to base their judgement on! Create shared responsibility towards the outcomes.

Selecting the stakeholders that should be involved when obtaining commitment is also of key importance. Involving the wrong stakeholders, or leaving out important ones, will have obvious repercussions. At the same time, selecting a too large a group of stakeholders may bog down the process. Too much communication may be a bad thing; it may create unnoticed and uncontrolled discussion outside the main discussion, leading to twisted conceptualisations and expectations.

Though ideally ‘everyone’ should be heard, this is generally a practical impossibility. Therefore, choose your experts carefully. Aim for the opinion leaders, and also accept that you cannot please everyone. Realise that some people will not be perfectly satisfied, prepare for it, and deal with it.

People who actually make the decisions are usually those who are just outside the group of people who really know what is going on. Make sure that the former people are also involved and aware of what is happening.

Getting executive commitment may actually be dictated technologically. If their business is highly technological, business people do not see technology as secondary, and will only commit to something if they are assured that ‘their organisation will be able to run it’.

Sharing design decisions and their underlying considerations at a late stage has a negative impact on the commitment of stakeholders. Start building commitment early on in the process. This implies that the linear ordering of the ‘viewpoint use phases’ as provided at the start of this section should not be applied strictly.

Once agreement has been reached, you should document this explicitly. Models are never accepted as sufficient statements to base agreements and commitment on. Commitments and agreements also need to be spelled out separately, in text.

7.4.7 Informing Stakeholders

Once commitment from the opinion leaders has been obtained, other stakeholders may be informed about the future plans and their impact. In doing so, it still makes sense to concentrate on cost/benefit considerations when trying to ‘sell’ the new system. Below, we have gathered some ob-
servations that apply to the informing phase. However, due to their general communicative nature, some of these observations are also applicable to the creation, validation, and commitment phases.

Do not impose presumed architectural terminology on true business people. Use their terminology. Even a concept like ‘service’ is suspect because it is relatively technology oriented and often unknown by stakeholders that are strictly on the business side.

Models are particularly important in giving stakeholders a feeling that they are ‘part of the larger whole’. Often, just knowing where in the model ‘they can be found’ is important to stakeholders, even if they do not understand the fine points of the model.

Communication is the crucial factor in enterprise architecture. It will even pay off actually to employ some communication experts (think marketing, PR, even entertainment!) in larger projects. As a result, you will end up with stakeholders that are genuinely prepared to change the way they and their business work, not just with some interesting looking plans and models. Crucially, communication can be quite different for various stages of system development. Therefore, it is important to have a good communication strategy and a framework guiding you in this.

Even if people are willing to and able to read models thoroughly, text (spoken or written) needs to be added. Models alone never suffice.

7.5 Basic Design Viewpoints

The most basic type of viewpoint is the selection of a relevant subset of the ArchiMate concepts and the representation of that part of an architecture that is expressed in the concepts in this selection. This is sometimes called a ‘diagram’, akin to, for instance, the UML diagrams.

In Sect. 6.3.2, we introduced the following four metaphorical directions from which we can identify relevant model elements:

1. ‘inwards’, towards the internal composition of the element;
2. ‘upwards’, towards the elements that are supported by it;
3. ‘downwards’, towards its realisation by other elements;
4. ‘sideways’, towards peer elements with which it cooperates.

We also use these directions to identify possibly useful viewpoints.

For the ‘composition’ viewpoints, we start from the basic structure of our modelling language. In its elementary form, the generic meta-model that is behind the language consists of active structural elements such as actors, behavioural elements such as functions and processes, and passive informational elements such as business and data objects, which are pro-
essed by the active elements in the course of their behaviour (see also Fig. 7.6).

From this basic structure, we can deduce a first set of viewpoint types, containing three viewpoints that are centred around one specific type of concept:

1. active elements, e.g., the composition of a business actor from sub-actors, i.e., an organisation structure;
2. behaviour elements, e.g., the structure of a business process in terms of subprocesses;
3. passive elements, e.g., the information structure in terms of data objects.

Although these viewpoints take a specific type of concept and its structure as their focus, they are not limited to these concepts, and closely related concepts are also included.

For the ‘upwards’ support of elements in their environment, the active elements offer interfaces through which their services can be used. ‘Downwards’ services are realised by processes and functions, and application components are deployed on infrastructure elements. ‘Sideways’ cooperation is achieved through collaborations between active elements and their behaviour in the form of interactions, and flows of information and value that relate the elements. Passive elements often play a role in these relations, e.g., by being passed from one element to another, but are not the focus. Hence we concentrate on the relations between the active and behaviour elements.

Next to the design viewpoints resulting from these metaphorical directions, which focus on a limited part of an enterprise architecture, we also need to represent the whole architecture, but in a simplified form. Especially early in the design process, when we do not yet know all the details that are added later on, we want to express an architecture using a subset of the ArchiMate language denoted in an informal, simplified form. This helps to avoid the impression that the design is already fixed and immutable, which may easily arise from a more formal diagram. Furthermore, such a high-level overview is very useful in obtaining commitment from stakeholders at an early stage of the design (see also Sect. 7.4.6). To this end, we introduce the Simplified viewpoint.

In each of the viewpoint types, concepts from the three layers of business, application, and technology may be used. However, not every combination of these would give meaningful results; in some cases, for example, separate viewpoints for the different layers are advisable. Based on common architectural practice, our experiences with the use of ArchiMate models in practical cases, and on the diagrams used in other languages like UML, we have selected the most useful combinations in the form of a
‘standard’ set of basic viewpoints to be used with the ArchiMate concepts (Table 7.5).

Table 7.5. Design viewpoints.

<table>
<thead>
<tr>
<th>Early design</th>
<th>Cooperation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introductory, p. 173</td>
<td>Actor Cooperation, p. 175</td>
</tr>
<tr>
<td></td>
<td>Business Process Cooperation, p. 180</td>
</tr>
<tr>
<td></td>
<td>Application Cooperation, p. 182</td>
</tr>
<tr>
<td>Composition</td>
<td>Realisation</td>
</tr>
<tr>
<td>Organisation, p. 175</td>
<td>Service Realisation, p. 179</td>
</tr>
<tr>
<td>Business Function, p. 177</td>
<td>Implementation &amp; Deployment, p. 187</td>
</tr>
<tr>
<td>Business Process, p. 181</td>
<td></td>
</tr>
<tr>
<td>Information Structure, p. 182</td>
<td></td>
</tr>
<tr>
<td>Application Behaviour, p. 185</td>
<td></td>
</tr>
<tr>
<td>Application Structure, p. 186</td>
<td></td>
</tr>
<tr>
<td>Infrastructure, p. 186</td>
<td></td>
</tr>
</tbody>
</table>

Some of these viewpoints have a scope that is limited to a single layer or aspect: the Business Function and Business Process viewpoints show the two main perspectives on the business behaviour; the Organisation viewpoint depicts the structure of the enterprise in terms of its departments, roles, etc.; the Information Structure viewpoint describes the information and data used; the Application Structure, Behaviour, and Cooperation viewpoints contain the applications and components and their mutual relations; and the Infrastructure viewpoint shows the infrastructure and platforms underlying the enterprise’s information systems in terms of networks, devices, and system software. Other viewpoints link multiple layers and/or aspects: the Actor Cooperation and Product viewpoints relate the enterprise to its environment; the Application Usage viewpoint relates applications to their use in, for example, business processes; and the Deployment viewpoint shows how applications are mapped onto the underlying infrastructure.

In the next subsections, we will explain these design viewpoints in more detail and provide examples of each one. In these examples, we have made extensive use of the abstraction rule that can be applied on chains of structural relations in ArchiMate, which was explained in Sect. 5.7. Note that it is explicitly not the intention to limit the user of the ArchiMate language to these viewpoints; neither do we expect an architect to draw all these diagrams in a given situation! They are meant to assist the modeller in choosing the contents of a view, but combinations or subsets of these viewpoints could well be useful in specific situations.
It is important in the examples that these views exhibit considerable overlap, e.g., in Fig. 7.13, which shows the high-level business functions of our ArchiSurance example, there is a business function Customer Relations. This reappears in Fig. 7.18, which shows how the Handle Claim business process is related to a number of business functions. Different aspects of this business process are shown, for example, in Fig. 7.19 (its use of information), Fig. 7.16 (realisation of services by business processes), and Fig. 7.17 (its relations with other business processes), and there are many more of these overlaps between views. This shows that underlying these different views there is a single model, and each view is a projection of the relevant elements in that model. We will use two examples throughout the description of the basic design viewpoints to illustrate this coherence:

- The handling of insurance claims;
- The policy administration systems and infrastructure.

### 7.5.1 Introductory Viewpoint

The Introductory viewpoint forms a subset of the full ArchiMate language using a simplified notation. It is typically used at the start of a design trajectory, when not everything needs to be detailed, or to explain the essence of an architecture model to non-architects who require a simpler notation. Another use of this basic, less formal viewpoint is that it tries to avoid the impression that the architectural design is already fixed, an impression that may easily arise when using a more formal, highly structured, or detailed visualisation.

We use a simplified notation for the concepts (Fig. 7.8), and for the relations. All relations except ‘triggering’ and ‘realisation’ are denoted by simple lines; ‘realisation’ has an arrow in the direction of the realised service; ‘triggering’ is also represented by an arrow. The concepts are denoted with slightly thicker lines and rounded corners, which give a less formal impression. The example in Fig. 7.9 illustrates this notation.

On purpose, the layout of this example is not as ‘straight’ as an ordinary architecture diagram; this serves to avoid the idea that the design is already fixed and immutable. This conforms to the suggestion made in Weinberg (1988) to use squiggly lines rather than firm lines, to show to the reader of a view that specific parts of the view are still open to debate.
Fig. 7.8. Concepts and notation for the Introductory viewpoint.

Fig. 7.9. Example of an Introductory view which conforms to the viewpoint of Fig. 7.8.
7.5.2 Organisation Viewpoint

The Organisation viewpoint shows the structure of an internal organisation of the enterprise, department, or other organisational entity. It can be represented in the form of a nested block diagram, but also in more traditional ways like the organigram. An Organisation view is typically used to identify authority, competencies, and responsibilities within an organisation.

Fig. 7.10. ArchiSurance organisation structure.

In Fig. 7.10, we can see the high-level subdivision of ArchiSurance into a front and back office, and a finance department. Within the back office, there are three departments responsible for specific products, e.g., car, travel, or legal aid insurance, and the shared service centre for document processing. The front office comprises two departments that handle the relations with customers and intermediaries, respectively.

7.5.3 Actor Cooperation Viewpoint

The Actor Cooperation viewpoint focuses on the relations of actors with each other and their environment. A common example of this is what is sometimes called a ‘context diagram’, which puts an organisation into its environment, consisting of external parties such as customers, suppliers, and other business partners. It is useful in determining external dependencies and collaborations and shows the value chain or network in which the organisation operates. Another important use of this viewpoint is in showing how a number of cooperating (business and/or application) actors together realise a business process, by showing the flows between them.
The main roles involved in the insurance business are the customer, the insurer, the intermediary, and the customer’s bank. These cooperate in different settings. For example, closing an insurance contract involves the customer, insurer, and intermediary, whereas premium collection involves the insurer, the customer and the customer’s bank. The main collaborations of ArchiSurance, which fulfils the role of the insurer, are shown in Fig. 7.11.

![Collaborations of ArchiSurance and its partners.](image1)

**Fig. 7.11.** Collaborations of ArchiSurance and its partners.

![Information flows between ArchiSurance’s departments and partners in handling insurance claims.](image2)

**Fig. 7.12.** Information flows between ArchiSurance’s departments and partners in handling insurance claims.
If we look more closely at the relations between actors and roles, it is useful to focus on the information flows between them to identify, for example, important dependencies. In Fig. 7.12, we see the information flows that are associated with the Handle Claim business process that is used as an example throughout the description of these viewpoints. The types of business objects passed between the actors are put as annotations to the flow arrows; these correspond to the business objects used by the Handle Claim process shown in Fig. 7.19. If needed, we could also include the interfaces used in exchanging this information, e.g., telephone or e-mail.

### 7.5.4 Business Function Viewpoint

The Business Function viewpoint shows the main business functions of an organisation and their relations in terms of the flows of information, value, or goods between them. Business functions are used to represent what is most stable about a company in terms of the primary activities it performs, regardless of organisational changes or technological developments. Business function architectures of companies that operate in the same market therefore often exhibit many similarities. The Business Function viewpoint thus provides high-level insight into the general operations of the company, and can be used to identify necessary competencies, or to structure an organisation according to its main activities.

![Business Function Viewpoint Diagram](image)

**Fig. 7.13.** Business functions and flows of information and money.
In the example of Fig. 7.13, we can see the information flow associated with the handling of insurance claims. Claims are submitted to the Maintaining Customer Relations business function, processed by Claim Handling, and paid by Financial Handling. In the Business Process viewpoint (Sect. 7.5.5), we will see a more detailed depiction of this process. In Fig. 7.14, these business functions are mapped onto the responsible organisational units that were shown in Fig. 7.10.

### 7.5.5 Product Viewpoint

The Product viewpoint depicts the value this product offers to the customers or other external parties involved and shows the composition of one or more products in terms of the constituting (business or application) services, and the associated contract(s) or other agreements. It may also be used to show the interfaces (channels) through which this product is offered, and the events associated with the product.

A Product view is typically used in designing a product by composing existing services or by identifying which new services have to be created for this product, given the value a customer expects from it. It may then serve as input for business process architects and others that need to design the processes and IT systems that realise this product.
A typical insurance product of ArchiSurance is depicted in Fig. 7.15. The value to the customer of an insurance is typically the added security it provides. The services mentioned here are realised by various business processes, an example of which is given in Sect. 7.5.6.

### 7.5.6 Service Realisation Viewpoint

The Service Realisation viewpoint is used to show how one or more business services are realised by the underlying processes (and sometimes by application components). Thus, it forms the bridge between the Product viewpoint and the Business Process viewpoint. It provides a ‘view from the outside’ of one or more business processes.
Business services are realised by business processes. In Fig. 7.15, we saw the services that constitute the travel insurance product. The business processes that realise these services are shown in Fig. 7.16. For example, the Claim registration service is realised by the Handle Claim business process that we use as an example throughout this chapter.

### 7.5.7 Business Process Cooperation Viewpoint

The Business Process Cooperation viewpoint is used to show the relations of one or more business processes with each other and/or their surroundings. It can be used both to create a high-level design of business processes within their context and to provide an operational manager responsible for one or more such processes with insight into their dependencies. Important aspects of coordination are:

- causal relations between the main business processes of the enterprise;
- the mapping of business processes onto business functions;
- realisation of services by business processes;
- the use of shared data;
- the execution of a business process by the same roles or actors.

Each of these can be regarded as a ‘sub-viewpoint’ of the Business Process Cooperation viewpoint. Below, we give examples of some of the resulting views.

![Diagram of main business processes, triggers, and relations of Archi-Surance.](image)

**Fig. 7.17.** Some of the main business processes, triggers, and relations of Archi-Surance.
In Fig. 7.17, the most important business processes of ArchiSurence are depicted. It also shows their relations, e.g., the Collect Premium process needs to be preceded by the Close Contract process, since of course no premium can be collected before the insurance policy has been issued. This figure also shows the Handle Claim process that occurs in many of the other viewpoints. In Fig. 7.18, the Handle Claim business process of Fig. 7.19 is mapped onto the business functions of Fig. 7.13.

### 7.5.8 Business Process Viewpoint

The Business Process viewpoint is used to show the high-level structure and composition of one or more business processes. Next to the processes themselves, this viewpoint contains other directly related concepts such as:

- the services a business process offers to the outside world, showing how a process contributes to the realisation of the company’s products;
- the assignment of business processes to roles, which gives insight into the responsibilities of the associated actors;
- the information used by the business process.

Each of these can be regarded as a ‘sub-viewpoint’ of the Business Process viewpoint.

Fig. 7.19. The Handle Claim business process and its use of information.
In Fig. 7.19, the Handle Claim business process is shown, together with the information it uses. This shows in more detail which subprocesses are carried out in handling insurance claims.

### 7.5.9 Information Structure Viewpoint

The Information Structure viewpoint is basically identical to the traditional information models created in the development of almost any information system. It shows the structure of the information used in the enterprise or in a specific business process or application, in terms of data types or (object-oriented) class structures. Furthermore, it may show how the information at the business level is represented at the application level in the form of the data structures used there, and how these are then mapped onto the underlying infrastructure, e.g., by means of a database schema.

In Fig. 7.20, the most important business objects of ArchiSurance are shown. Some of these are used in the Handle Claim business process, as depicted in Fig. 7.19.

![Information model of ArchiSurance](image)

**Fig. 7.20.** Information model of ArchiSurance.

### 7.5.10 Application Cooperation Viewpoint

The Application Cooperation viewpoint shows the relations of a number of applications or components. It describes the dependencies in terms of the information flows between them, or the services they offer and use. This viewpoint is typically used to create an overview of the application landscape of an organisation.
This viewpoint is also used to express the coordination or orchestration (i.e., internal coordination) of services that together support the execution of a business process. By modelling the interdependencies between services, the coordination of the underlying applications is established in a more independent way. If this coordination is centralised and internal to the enterprise, we speak of ‘orchestration’; in the case of coordination between independent entities, the term ‘choreography’ is often used.

The front- and back-office applications of ArchiSurance are shown in Fig. 7.21. It is clear that the back office is structured according to the different types of products, whereas the front office is already more integrated. One of the applications shown is the Home & Away policy administration used in several other viewpoints as well.

Fig. 7.21. Applications and information flow of ArchiSurance.

Some of the connections between the ArchiSurance applications are shown in Fig. 7.22, which shows that ArchiSurance uses the Enterprise Service Bus concept to link its applications. In Fig. 7.23, we see in more detail how the Claim information service from the Home & Away Policy administration is used by the department’s Financial application, through an interface in which the message queuing service from the lower-level infrastructure is used (see also Fig. 7.28).
**Fig. 7.22.** Applications connected through the ArchiSurance Service Bus.

**Fig. 7.23.** Details of the connection between the Home & Away Policy administration and Financial application.

### 7.5.11 Application Usage Viewpoint

The Application Usage viewpoint describes how applications are used to support one or more business processes, and how they are used by other applications. It can be used in designing an application by identifying the services needed by business processes and other applications, or in designing business processes by describing the services that are available. Furthermore, since it identifies the dependencies of business processes upon applications, it may be useful to operational managers responsible for these processes.

In Fig. 7.24 it is shown how the Handle Claim business process uses the application services offered by several applications. Each of these services is realised by the behaviour of an application, an example of which is given in Fig. 7.25.
7.5.12 Application Behaviour Viewpoint

The Application Behaviour viewpoint describes the internal behaviour of an application or component, for example, as it realises one or more application services. This viewpoint is useful in designing the main behaviour of applications or components, or in identifying functional overlap between different applications.

Fig. 7.25. Behaviour of the Home & Away Policy administration in realising the Policy creation service.

Part of the behaviour of the Home & Away Policy administration is shown in Fig. 7.25. The individual application functions are chained together and collectively realise the Policy creation application service. The
communication with the Financial administration takes place in an interaction that realises the Policy information service.

### 7.5.13 Application Structure Viewpoint

The Application Structure viewpoint shows the structure of one or more applications or components. This viewpoint is useful in designing or understanding the main structure of applications or components and the associated data, e.g., to create a first-step work breakdown structure for building a system, or in identifying legacy parts suitable for migration.

![Home & Away Policy administration](image)

**Fig. 7.26.** Main structure of the Home & Away Policy administration.

Fig. 7.26 shows the main components that constitute the policy administration of ArchiSurance’s Home & Away department. It also depicts some of the important data objects used by these components. These data objects are realisations of the business objects of Fig. 7.20.

### 7.5.14 Infrastructure Viewpoint

The Infrastructure viewpoint comprises the hardware and software infrastructure upon which the application layer depends. It contains physical devices and networks, and supporting system software such as operating systems, databases, and middleware.

The physical infrastructure of ArchiSurance and its intermediaries is shown in Fig. 7.27.
7.5.15 Infrastructure Usage Viewpoint

The Infrastructure Usage viewpoint shows how applications are supported by the software and hardware infrastructure: infrastructure services delivered by the devices, system software, and networks are provided to the applications.

An example of this viewpoint is given in Fig. 7.28, which shows the use, by a number of back-office applications, of the messaging and data access services offered by ArchiSurance’s infrastructure.

This viewpoint plays an important role in the analysis of performance and scalability, since it relate the physical infrastructure to the logical
world of applications. It is very useful in determining the performance and quality requirements of the infrastructure based on the demands of the various applications that use it. In Chap. 8, we will describe a quantitative analysis technique that can be used to determine, for example, the load on the infrastructure, based on its use by applications (and their use by business processes).

### 7.5.16 Implementation & Deployment Viewpoint

The Implementation & Deployment viewpoint shows how one or more applications are deployed on the infrastructure. This comprises the mapping of (logical) applications and components onto (physical) artifacts like, for instance, Enterprise Java Beans, and the mapping of the information used by these applications and components onto the underlying storage infrastructure, e.g., database tables or other files. In security and risk analysis, Deployment views are used to identify critical dependencies and risks. Fig. 7.29 shows the mapping of logical application components of the Home & Away Policy administration (see Fig. 7.26) used in several of the other examples onto physical artifacts such as Enterprise Java Beans.

![Implementation of the Home & Away Policy administration](image)

**Fig. 7.29.** Implementation of the Home & Away Policy administration.
7.6 Summary

In the previous sections, we have advocated a viewpoint-oriented approach to enterprise architecture modelling, in which architects and other stakeholders can define their own views of the architecture. In this approach views are specified by viewpoints, which define abstractions on the set of models representing the enterprise architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns.

We have described the use of viewpoints in communication, and the distinction between an architecture model, a view of that model, and its visualisation and manipulation. We have presented guidelines for the selection and use of viewpoints, and outlined a number of viewpoints in the Archi-Mate language that can be used by architects involved in the creation or change of enterprise architecture models.
Guide to Enterprise IT Architecture

Col Perks
Tony Beveridge

Springer
7. Architectural Views  166

7.1 Introduction  166

7.2 The Role of Architectural Views  167
  Recommended Views  168

7.3 Business Process Domain View  169
  Introduction  169
  Role  169

7.4 Functional View  171

7.5 Security View  171
  Basic Concepts  172
  Generic Security Architecture View  173

7.6 Management View  175

7.7 Software Engineering View  175
  Data Intensive versus Information Intensive Software Systems  179
  Software Tiers  180

7.8 Data Management View  181
  Database Models  183
  Data Dictionary/Directory Systems  185
  Data Security  186

7.9 User View  186

7.10 System Engineering View  187
  Client/Server Model  187
  Master/Slave and Hierarchical Models  188
  Peer-to-Peer Model  190
  Distributed Object Management Model  190

7.11 Communications View  191
  Introduction  191
  Communications Infrastructure  191
  Communications Models  193
  Allocation of Services to Components  195

7.12 Summary  195
An individual system structure (say its application architecture) is a complex collection of components, building blocks, objects, hardware, networks, services, and nonfunctional requirements. Representing these aspects in a unified architecture can be difficult. This is complicated by the fact that an eclectic array of skills is required to specify, develop, and assure such an architecture.

This problem is multiplied for enterprise technical architecture development. All components of the IT environment are required to be modeled within the architecture to ensure that the end product (in this case, the organization’s IT environment) is complete, logical, reasoned, and meets the organization’s business requirements. The organization’s technical architecture can be considered an $n$-dimensional space, each dimension presenting a single component of the architecture, with the points on the dimension being a specific element of each component. The complexity of such a representation limits its ability to be understood (a significant aspect of the architectural approach being the ability to support effective communication of the architecture) by those who created it and possibly not even then. It affects the architecture’s ability to be assessed and assured by subject matter experts in the organization. Most of all, it affects the ability to deliver and maintain the architecture. We therefore need a mechanism to reduce the effects of these issues.

Architectural views provide us with such an approach. In general, views allow slices to be taken through the architecture at significant points to support increased understanding, assessment, and assurance
and heighten the chances for successful implementation and maintenance. Views can be taken at any point through the architecture—there is no right way to slice up the architecture. The most typical approaches include:

- Platform service views
- Quality views
- Functional views
- Project views

This chapter presents a number of architectural views that can be used to describe the technical architecture. Because this is their purpose, the bias is toward platform service views.

### 7.2 The Role of Architectural Views

Depending on the area of responsibility of the architect, an architecture may be viewed from different perspectives. For example, the architect responsible for computing perceives the architecture with a different focus than the Architect responsible for data management. The architect responsible for the overall system has yet another focus.

These different areas of focus, or slices, are called “views.”

A view therefore is a means of describing how an organization’s specific needs are embodied in the architecture. Views are used as a method of cross-checking that a proposed technical architecture will meet all aspects of the computational needs that will be imposed on it or that a current architecture is fit for its intended purpose.

Pertinent views are taken of both the existing system and the target system to establish which elements of the current system must be carried forward and which must be removed or replaced. The use of views for describing the current systems environment was presented briefly in Chapter 6.

The views presented in the following sections describe architecture concepts from different perspectives. Each of these views addresses components, interfaces, and allocation of services critical to the view.

TOGAF presents some recommended views, some or all of which may be appropriate in a particular architecture development. This is not an exhaustive set of views, and those shown may be supplemented by additional views as required.
**Recommended Views**

We have separated the architectural views into three levels:

- The functional view focuses on the functional aspects of the system; that is, on what the new system is intended to do. This can be built up from an analysis of the existing environment and the requirements and constraints affecting the new system.

- Implementation views focus on how the system is implemented and how that affects its properties:
  - The management view examines the architecture from the point of view of the service provider. It covers issues such as initial deployment, upgrading, availability, security, performance, asset management, fault, and event management of system components from the management perspective.
  - The security view focuses on the security aspects of the systems for the protection of information within the organization. It examines the systems to establish what information is stored and processed, how valuable it is, what threats exist, and how they can be addressed.
  - The builder’s view deals with aspects of interest to software developers. It considers what software development constraints and opportunities exist in the new architecture and looks at how development can be carried out in terms of both technology and resources.
  - The data management view deals with the storage, retrieval, processing, archiving, and security of data. It looks at the flow of data as it is stored and processed and at what components will be required to support and manage both storage and processing.
  - The user view considers the usability aspects of the IT environment.

- Physical views concentrate more on the location, type, and power of the equipment and software:
  - The computing view presents a number of different ways in which software and hardware components can be assembled into working systems. To a great extent, the choice of model determines the properties of the future organizational systems. It looks at technology that already exists in the organization, and what is available currently or will be in the near future. This reveals areas where new technology can contribute to the function or efficiency of the new architecture and how different types of processing platforms can support different parts of the overall system.
The communications view examines various ways of structuring communications facilities to simplify the business of network planning and design. It examines the networking elements of the architecture in the light of geographic constraints, bandwidth requirements, and so on.

These views should not be considered an exhaustive set but simply a starting point. In developing an organization-specific architecture, it is very likely that some of these views will not be useful, whereas others not given here will be essential. For example, other views might be developed in response to specific business requirements such as performance, operating cost, uptime, or contract management.

The comparison of the views of the existing systems and the target architecture (as explained during the development of the target architecture) will often yield indications of where changes in environment and system requirements and objectives will lead to implementation constraints and to recognition of elements being retained or eliminated. This comparison should also yield a better understanding of relationships between the architectures of the existing and target systems.

### 7.3 Business Process Domain View

#### Introduction

A business process domain is a logical grouping of business systems dedicated to a common purpose. Such systems may be geographically collocated, thus emphasizing their purpose, or they may be grouped by some other constraint such as a common systems availability target.

To demonstrate the responsiveness of the technical architecture to the business needs of the organization, among the various architecture views that are developed a business process domain view should be considered. This describes the technical architecture from the perspective of the enterprise’s key business process domains.

#### Role

A business process domain view is a set of functional views aligned with the business process structure of the enterprise. Business process domain views are used during architecture development as a means of verifying and demonstrating that the architecture being developed is addressing the business requirements. Specifically, they are an output of Phase C and are used in Phase D of the ADM.

Thorough and detailed application of the business scenario technique in the early stages of architecture development should provide an ex-
cellent foundation for a set of business process domain views. Business scenarios describe:

- A business process, application, or set of applications that can be enabled by the architecture
- The business and technology environment
- The people and computing components who execute the scenario
- The desired outcome of proper execution

Each business process domain view addresses components, interfaces, and allocation of services critical to the view. A typical structure is shown in Table 7.1.

Table 7.1. Business process domain views—description structure.

| **Introduction** | A description of the domain, its key applications and attributes that characterize the applications within the domain |
| **Business problem statement** | A short description of the important business problems relating to the domain and how they are addressed in the target architecture |
| **Applications deployed within the business process domain** | A table listing currently deployed applications |
| **Assumptions, constraints and guidelines** | General guidelines for developers, implementers, and system suppliers |
| **Domain structure** | Target architecture/business process domain mapping: A table highlighting the services applicable to this domain |
| | Application topology: A figure showing the relationships between the major application elements within this domain |
| **Domain service qualities** | A description of the service qualities that are important for each business process domain and how they are achieved in the target architecture |
| **Deployment guidance strategy** | Lessons learned with respect to deployment; the migration strategy for implementing the target architecture as it applies to this business domain; also any guidance for the implementation team responsible for deployment |
| **Future directions** | Any important directions identified for systems within the domain |
| **References** | Pointers to reference material |
7.4 Functional View

The functional view considers the functional aspects of systems; that is, what the systems are intended to do. This can be built up from an analysis of the existing environment and of the requirements and constraints affecting the systems.

The architecture development method (ADM) outlines the process of analyzing and describing the existing environment in its own terms (Phase B) and then restating it in TOGAF terms to obtain information about what exists already (Phase C).

The new requirements and constraints will appear from a number of sources, possibly including:

- Existing internal specifications and lists of approved products
- Business goals and objectives
- Business process reengineering activities
- Changes in technology

Business scenarios are an important technique that may be used prior to, and as a key input to, the development of the architecture to help identify and understand business needs and thereby derive the business requirements and constraints that the architecture development must address.

What should emerge from the functional view is a clear understanding of the functional requirements for the systems that will exist in the new architecture, with statements such as “Improvements in handling customer inquiries are required through wider use of Computer/Telephony Integration.”

7.5 Security View

The essence of security is the controlled use of information. This section provides a brief overview of how security protection is implemented in the components of information systems. Doctrinal or procedural mechanisms, such as physical and personnel security procedures and policy, are not discussed here in any depth.

Figure 7.1 depicts an abstract view of an information system architecture, which emphasizes the fact that information systems from the security perspective are either part of a local subscriber environment (LSE) or a communications network (CN). An LSE may either be fixed or mobile. The LSEs by definition are under the control of the using organization. In an open system distributed computing implementation, secure and nonsecure LSEs will almost certainly be required to interoperate.
**Basic Concepts**

The concept of an information domain provides the basis for discussing security protection requirements. An information domain is defined as a set of users, their information objects, and a security policy. An information domain security policy is the statement of the criteria for membership in the information domain and the required protection of the information objects. Breaking down an organization's information into domains is the first step in reducing the task of security policy development to a manageable size.

The business of most organizations requires that their members operate in more than one information domain. The diversity of business activities and the variation in perception of threats to the security of information will result in the existence of different information domains within one organization security policy. A specific activity may use several information domains, each with its own distinct information domain security policy.

Information domains are not necessarily bounded by information systems or even networks of systems. The security mechanisms implemented in information system components may be evaluated for their ability to meet the information domain security policies.

Information domains can be viewed as being strictly isolated from one another. Information objects should be transferred between two information domains only in accordance with established rules, conditions, and procedures expressed in the security policy of each information domain.

The concept of “absolute protection” is used to achieve the same level of protection in all information systems supporting a particular information domain. It draws attention to the problems created by interconnecting LSEs that provide different strengths of security protection. This interconnection is likely because interoperable systems may consist of an unknown number of heterogeneous LSEs. Analysis of minimum security requirements will ensure that the concept of absolute protection will be achieved for each information domain across LSEs.
Generic Security Architecture View

Figure 7.2 shows a generic architectural view that can be used as an aid for describing the makeup of security services and the implementation of security mechanisms. This view identifies the architectural components within an LSE. The LSEs are connected by CNs. The LSEs include end systems, relay systems, and local communications systems (LCSs), described below.

- **Relay System (RS).** The component of an LSE, the functionality of which is limited to information transfer and is only indirectly accessible by users (e.g., router, switch, message transfer agent). It may have functionality similar to an end system, but an end user does not use it directly. Note that relay system functions may be provided in an end system (ES).
- **Local communication system (LCS).** A network that provides communications capabilities between LSEs or within an LSE with all of the components under control of a LSE.
- **Communication Network (CN).** A network that provides inter-LSE communications capabilities but is not controlled by LSEs (e.g., commercial carriers).

The end system and the relay system are viewed as requiring the same types of security protection. For this reason, the view of security, the protection in an end system generally also applies to a relay system.

Implementing the necessary security protection in the end system occurs in three system service areas of TOGAF. They are operating system services, network services, and system management services.

Most of the implementation of security protection is expected to occur in software. The hardware is expected to protect the integrity of the end system software. Hardware security mechanisms include protection against tampering, undesired emanations, and cryptography.

A “security context” is defined as a controlled process space subject to an information domain security policy. The security context is therefore analogous to a common operating system notion of user process

![Figure 7.2. Generic security architecture view.](image-url)
space. Isolation of security contexts is required. Security contexts are required for all applications (e.g., end-user and security management applications). The focus is on strict isolation of information domains, management of end-system resources, and controlled sharing and transfer of information among information domains. Where possible, security-critical functions should be isolated into relatively small modules that are related in well-defined ways.

The operating system will isolate multiple security contexts from one another using hardware-protection features (e.g., processor state register, memory mapping registers) to create separate address spaces for each of them. Untrusted software will use end-system resources only by invoking security-critical functions through the separation kernel. Most of the security-critical functions are the low-level functions of traditional operating systems.

Two basic classes of communications are envisioned for which distributed security contexts may need to be established. These are interactive and staged (store and forward) communications.

The concept of a “security association” forms an interactive distributed security context. A security association is defined as all the communication and security mechanisms and functions that extend the protections required by an information domain security policy within an end-system to information in transfer between multiple end systems. Multiple security protocols may be included in a single security association to provide a combination of security services. IPSEC is an example of a technology that supports the forming of security associations.

For staged delivery communications (e.g., electronic mail), use will be made of an encapsulation technique (called a “wrapping process”) to convey the necessary security attributes with the data being transferred as part of the network services. The wrapped security attributes are intended to permit the receiving end system to establish the necessary security context for processing the transferred data.

Security management is a particular instance of the general information system management functions described by the TOGAF Systems Management Service. Information system security management services are concerned with the installation, maintenance, and enforcement of information domain and information system security policy rules in the information system intended to provide these security services. In particular, the security management function controls information needed by operating system services within the end-system security architecture. In addition to these core services, security management requires event handling, auditing, and recovery. Standardization of security management functions, data structures, and protocols will enable interoperation of security management application processes (SMAPs) across many platforms in support of distributed security management.
7.6 Management View

The management view acts as a check and balance on the difficulties and day-to-day running costs of systems built within the new architecture. Often, system management is not considered until after all of the important purchasing and development decisions have been made, and taking a separate management view at an early stage in architecture development is one way to avoid this pitfall. It is good practice to take the management view after the security view because in general the system management personnel are also responsible for system security.

Key elements of the management view are:

- Which system and network management services are required
- The likely quantity, quality and location of management and support personnel
- The ability of users to take on system management tasks
- The manageability of existing and planned systems in each of the required system and network management service areas
- Whether management should be centralized or distributed
- Whether security is the responsibility of system managers or a separate group, bearing in mind any legal requirements

In general stakeholders are concerned with assuring that the availability of systems does not suffer when changes occur. Managing the systems includes managing components such as:

- security components
- data assets
- software assets
- hardware assets
- networking assets

Key technical components categories that are the subject of the management view deal with change, either planned upgrades or unplanned outages. Table 7.2 lists specific concerns in each component category.

7.7 Software Engineering View

Building software-intensive systems is both expensive and time consuming. Because of this it is necessary to establish guidelines to help minimize the effort required and the risks involved. Most organizations will
Table 7.2. Operational concerns.

<table>
<thead>
<tr>
<th>Component category</th>
<th>Planned change considerations</th>
<th>Unplanned change considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Components</td>
<td>How does one propagate a security change throughout systems?</td>
<td>What should happen when security is breached?</td>
</tr>
<tr>
<td></td>
<td>Who is responsible for making changes, end users, or security stewards?</td>
<td>What should happen if a security component fails?</td>
</tr>
<tr>
<td>Data assets</td>
<td>How does one add new data elements?</td>
<td>What are your backup procedures and are all the system capabilities there to backup in time?</td>
</tr>
<tr>
<td></td>
<td>How does one import/export or load/unload data?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>How is backup managed while running continuously?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>How is data change propagated in distributed environment?</td>
<td></td>
</tr>
<tr>
<td>Software assets</td>
<td>How does one introduce a new application into the systems?</td>
<td>What do you want to happen when an application fails?</td>
</tr>
<tr>
<td></td>
<td>What procedures do you have to control software quality?</td>
<td>What do you want to happen when a resource of applications fails?</td>
</tr>
<tr>
<td></td>
<td>How does one propagate application changes in a distributed environment?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>How does one restrict unwanted software introduction given the internet?</td>
<td></td>
</tr>
</tbody>
</table>
Table 7.2. Operational concerns (continued).

<table>
<thead>
<tr>
<th>Component category</th>
<th>Planned change considerations</th>
<th>Unplanned change considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware assets</td>
<td>How do you assess the impact of new hardware on the system, especially network load?</td>
<td>What do you want to happen when hardware outages occur?</td>
</tr>
<tr>
<td>Networking assets</td>
<td>How do you assess the impact of new networking components?</td>
<td>What do you want to happen when networking outages occur?</td>
</tr>
<tr>
<td></td>
<td>How do you optimize your networking components?</td>
<td></td>
</tr>
</tbody>
</table>

develop custom software therefore it is important to define the frameworks under which this will occur. This view helps the architect to analyze the current methods used by the organization to develop software and aids in positioning architectural styles for future development. This is one of the most obvious interactions between enterprise and application architecture. In the role of an enterprise standards-setter the technical architecture is required to develop an opinion as to how individual software development projects should operate. This view helps in determining the factors to consider. The view also relates to the less abstract system engineering view.

Major points for the technical architecture to consider in developing a software engineering strategy are:

- Development approach
- Software modularity and reuse
- Portability
- Migration and interoperability
- Architectural models

The development approach consists of the many lifecycle models defined for software development (waterfall, prototyping, etc.). A consideration for the architect is how best to feed architectural decisions into the lifecycle model that is going to be used for development of the system.

- Prototyping or rapid prototyping may be used to determine the overall feasibility of the system and also to demonstrate the user interface to be provided with the system.
Iterative techniques may be combined with prototyping to incrementally develop the system in conjunction with potential system users.

As a piece of software grows in size, so the complexity and interdependencies between different parts of the code increase. Reliability will fall dramatically unless this complexity can be brought under control. Modularity is a concept by which a piece of software is grouped into a number of distinct and logically cohesive sub-units, presenting services to the outside world through a well-defined interface. Generally speaking, the components of a module will share access to common data, and the interface will provide controlled access to this data. Using modularity, it becomes possible to build a software application incrementally on a reliable base of pre-tested code. A further benefit of a well defined modular system is that the modules defined within it may be re-used in the same or on other projects, cutting development time dramatically by reducing both development and testing effort.

In recent years, the development of Object Oriented Programming Languages has greatly increased programming language support for modular development and code re-use. Such languages allow the developer to define "classes" (a unit of modularity) of objects that behave in a controlled and well-defined manner. Techniques such as inheritance, which enables parts of an existing interface to an object to be changed, enhance the potential for re-usability by allowing pre-defined classes to be tailored or extended when the services they offer do not quite meet the requirement of the developer. If modularity and software re-use are likely to be key objectives of new software developments, consideration must be given to whether the component parts of any proposed architecture may facilitate or prohibit the desired level of modularity in the appropriate areas.

Software portability, the ability to take a piece of software written in one environment and make it run in another, is important in many projects, especially product developments. It requires that all software and hardware aspects of a chosen technical architecture (not just the newly developed application) be available on the new platform. It will, therefore, be necessary to ensure that the component parts of any chosen architecture are available across all the appropriate target platforms.

Interoperability is always required between the component parts of a new architecture. It may also, however, be required between a new architecture and parts of an existing legacy system—for example during the staggered replacement of an old system. Interoperability between the new and old architectures may, therefore, be a factor in architectural choice.

Determining how software architectures are to be modeled represents
a key decision point for the technical architecture. There are numerous modeling techniques, many of which are related to software development methods (Rational Unified Process and UML for instance)

**Data Intensive versus Information Intensive Software Systems**

The technical architecture must be cognizant of the fact that one software engineering approach does not fit all situations. Size, problem domain, and complexity are key differentiators. While the objective is to “unify” development methods in an integrated manner—interoperability with the platform being a key driver—it continues to be important to consider different techniques to tackle differing development types.

From an abstract point-of-view it is possible to view software systems in two general categories. First, there are those systems that require only a user interface to a database, requiring little or no business logic built into the software. These systems can be called “Data Intensive.” Second, there are those systems that require users to manipulate information that might be distributed across multiple databases, and to do this manipulation according to predefined business logic. These systems can be called “Information Intensive.”

Data intensive systems can be built with reasonable ease through the use of 4GL tools. In these systems, the business logic is in the mind of the user, i.e., the user understands the rules for manipulating the data and uses those rules while doing his work.

Information intensive systems are different. Information is defined as “meaningful data,” i.e., data in a context that includes business logic. Information is different from data. Data is the tokens that are stored in databases or other data stores. Information is multiple tokens of data combined to convey a message. Typically, information reflects a model. Information intensive systems also tend to require information from other systems, and, if this path of information passing is automated, usually some mediation is required to convert the format of incoming information into a format that can be locally used. Because of this, information intensive systems tend to be more complex than others, and require the most effort to build, integrate, and maintain.

This view is concerned primarily with information intensive systems. In addition to building systems that can manage information, though, systems should also be as flexible as possible. This has a number of benefits. It allows the system to be used in different environments, for example, the same system should be usable with different sources of data, even if the new data store is a different configuration. Similarly, it might make sense to use the same functionality but with users who need
a different user interface. So information systems should be built so that they can be reconfigured with different data stores or different user interfaces. If a system is built to allow this, it enables the enterprise to reuse parts (or “components”) of one system in another.

Interoperability can only be achieved when information is passed, not when data is passed. Most information systems today get information both from their own data stores and other information systems. In some cases the web of connectivity between information systems is quite extensive. The United States Air Force technical architecture, for example, has a concept known as “A5 Interoperability.” This means that the required data is available anytime, anywhere, by anyone, who is authorized, in any way. This requires that many information systems are architecturally linked and provide information to each other. The technical architecture must consider the way current systems interoperate and provide meaningful direction (technology, methods, guidelines) as to how integration will “improve” and how new systems will be designed for integration.

**Software Tiers**

Physically, software architectures are either 2-tier or 3-tier. Each tier typically presents at least one capability.

In a two-tier architecture the user interface and business logic are tightly coupled while the data is kept independent. This allows the data to be independently maintained. The tight coupling of the user interface and business logic assure that they will work well together—for this problem in this domain. However, the tight coupling of the user interface and business logic dramatically increases maintainability risks while reducing flexibility and opportunities for reuse.

A three-tier approach adds a tier that separates (an amount of) the business logic from the user interface. This in principle allows the business logic to be used with different user interfaces as well as with different data stores. With respect to the use of different user interfaces, users might want the same user interface but using different COTS presentation servers, for example, Java Virtual Machine (JVM) or Common Desktop Environment (CDE). Similarly, if the business logic is to be used with different data stores, then each data store must use the same data model (“data standardization”), or a mediation tier must be added above the data store (“data encapsulation”).

An additional level of flexibility can be achieved through using a 5-tier scheme for software, extending the three-tier paradigm (see Figure 7.3). The scheme is intended to provide strong separation of the three major functional areas of the architecture. Since there are client and
server aspects of both, the user interface and the data store, the scheme then has 5 tiers.\textsuperscript{24}

The presentation tier is typically COTS-based. The presentation interface might be an X-server, Win32, etc. There should be a separate tier for the user interface client. This client establishes the look and feel of the interface; the server (presentation tier) actually performs the tasks by manipulating the display. The user interface client hides the presentation server from the application business logic.

The application business logic should be a separate tier. This tier is called the “application logic” and functions as a server for the user interface client. It interfaces to the user interface typically through call-backs. The application logic tier also functions as a client to the data access tier.

If there is a need to use an application with multiple databases with different schema, then a separate tier is needed for data access. This

\textsuperscript{24} We use the technical architecture term “tier” as apposed to the application architecture term “layering” to highlight the detail differences inherent between the two concepts.
client would access the data stores using the appropriate COTS interface and then convert the raw data into an abstract data type representing parts of the information model. The interface into this object network would then provide a generalized data access interface (DAI) that would hide the storage details of the data from any application that uses that data.

Each tier in this scheme can have zero or more components. The organization of the components within a tier is flexible and can reflect a number of different architectures based on need. For example, there might be many different components in the application logic tier (scheduling, accounting, inventory control, and so on) and the relationship between them can reflect whatever architecture makes sense, but none of them should be a client to the presentation server.

This clean separation of user interface, business logic, and information will result in maximum flexibility and componentized software that lends itself to product line development practices. For example, it is conceivable that the same functionality should be built once and yet be usable by different presentation servers (e.g., on PCs or UNIX boxes), displayed with different looks and feels depending on user needs, and usable with multiple legacy databases. Moreover, this flexibility should not require massive rewrites to the software whenever a change is needed.

### 7.8 Data Management View

Data management services may be provided by a wide range of implementations. Some examples include:

- Mega-centers providing functionally oriented corporate databases supporting local and remote data requirements
- Distributed database management systems that support the interactive use of partitioned and partially replicated databases
- File systems provided by operating systems, which may be used by both interactive and batch-processing applications

Data management services include the storage, retrieval, manipulation, backup, restart/recovery, security, and associated functions for text, numeric data, and complex data such as documents, graphics, images, audio, and video. The operating system provides file management services, but they are considered here because many legacy databases exist as one or more files without the services provided by a database management system (DBMS). The DBMS is the most critical component of any data management capability, and a data dictionary/directory system
is necessary in conjunction with the DBMS as a tool to aid the administration of the database. Data security is a necessary part of any overall policy for security in information processing.

A database management system (DBMS) provides for the systematic management of data. This data management component provides services and capabilities for defining, structuring, and accessing the data, as well as security and recovery of the data. A DBMS performs the following functions:

- Structures data in a consistent way
- Provides access to the data
- Minimizes duplication
- Allows reorganization, that is, changes in data content, structure, and size
- Supports programming interfaces
- Provides security and control

A DBMS must provide:

- Persistence—the data continue to exist after the application's execution has completed
- Secondary storage management
- Concurrency
- Recovery
- Data definition/data manipulation language (DDL/DML)

**Database Models**

The logical data model that underlies the database characterizes a DBMS. The common logical data models are listed below:

- **Relational DBMS.** A relational DBMS (RDBMS) structures data into tables that have particular properties. A collection of related tables in the relational model makes up a database. The mathematical theory of relations underlies the relational model—both the organization of data and the languages that manipulate the data. Edgar Codd, then at IBM, developed the relational model in 1973. It has been popular, in terms of commercial use, since the early 1980s.

- **Hierarchical DBMS.** The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

- **The Network DBMS.** The popularity of the network data model coincided with the popularity of the hierarchical data model. Some
data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model.

- **Object-oriented DBMS.** An object-oriented DBMS (OODBMS) must be both a DBMS and an object-oriented system. As a DBMS it must provide particular data management capabilities. OODBMSs typically can model tabular data, complex data, hierarchical data, and networks of data. The following are important features of an object-oriented system:
  - *Complex objects.* For example objects may be composed of other objects.
  - *Object identity.* Each object has a unique identifier external to the data.
  - *Encapsulation.* An object consists of data and the programs (or methods) that manipulate it.
  - *Types or classes.* A class is a collection of similar objects. Inheritance subclasses inherit data attributes and methods from classes.
  - *Overriding with late binding.* The method particular to a subclass can override the method of a class at run time.
  - *Extensibility.* For example a user may define new objects.
  - *Computational completeness.* A general-purpose language, such as Ada, C, C++, or Java is computationally complete. The special-purpose language SQL is not. Most OODBMSs incorporate a general-purpose programming language.

- **Flat file Systems.** A flat file system is usually closely associated with a storage access method. An example is IBM’s indexed sequential access method (ISAM). The models discussed earlier in this section are logical data models; flat files require the user to work with the physical layout of the data on a storage device.

- **Distributed DBMS.** A distributed DBMS manages a database that is spread over more than one platform. The database can be based on any of the data models just discussed above (except the flat file). The database can be replicated, partitioned, or a combination of both. A replicated database is one in which full or partial copies of the database exist on the different platforms. A partitioned database is one in which part of the database is on one platform and the other parts are on other platforms. The partitioning of a database can be vertical or horizontal.

  Whether the distributed database is replicated or partitioned, a single DBMS manages the database. There is a single schema (description of the data in a database in terms of a data model, such as relational) for a distributed database. The distribution of the data-
base is generally transparent to the user. The term “distributed DBMS” implies homogeneity.

A distributed, heterogeneous database system is a set of independent databases, each with its own DBMS, presented to users as a single database and system. “Federated” is used synonymously with “distributed heterogeneous.” The heterogeneity refers to differences in data models (e.g., network and relational), DBMSs from different suppliers, different hardware platforms or other differences. The simplest kinds of federated database systems are commonly called gateways. In a gateway, one vendor (e.g., Oracle) provides single-direction access through its DBMS to another database managed by a different vendor’s DBMS (e.g., IBM’s DB2). The two DBMSs need not share the same data model. For example, many RDBMS vendors provide gateways to hierarchical and network DBMSs.

There are federated database systems both on the market and in research that provide more general access to diverse DBMSs. These systems generally provide a schema integration component to integrate the schemas of the diverse databases and present them to the users as a single database, a query management component to distribute queries to the different DBMSs in the federation, and a transaction management component, to distribute and manage the changes to the various databases in the federation.

Data Dictionary/Directory Systems

The second component providing data management services, the data dictionary/directory system (DD/DS), consists of utilities and systems necessary to catalog, document, manage, and use metadata (data about data). The DD/DS is normally provided as part of a DBMS but is sometimes available from alternate sources. In the management of distributed data, distribution information may also be maintained in the network directory system. In this case, the interface between the DD/DS and the network directory system would be through the API of the network services component on the platform.

In current environments, data dictionaries are usually integrated with the DBMS, and directory systems are typically limited to a single platform. Network directories are used to expand the DD/DS realms. The relationship between the DD/DS and the network directory is an intricate combination of physical and logical sources of data.

A repository is a system that manages all of the data of an enterprise, which includes data and process models and other enterprise information. Hence, the data in a repository are much more extensive than those in a DD/DS, which generally defines only the data making up a database.
Data Security

The third component providing data management services is data security. This includes procedures and technology measures implemented to prevent unauthorized access, modification, use, and dissemination of data stored or processed by a computer system. Data security also includes data integrity (i.e., preserving the accuracy and validity of the data) and protecting the system from physical harm (including preventative measures and recovery procedures).

Authorization control allows only authorized users to have access to the database at the appropriate level. Guidelines and procedures can be established for accountability, levels of control, and type of control. Authorization control for database systems differs from that in traditional file systems because, in a database system, it is not uncommon for different users to have different rights to the same data. This requirement encompasses the ability to specify subsets of data and to distinguish between groups of users. In addition, decentralized control of authorizations is of particular importance for distributed systems.

Data protection is necessary to prevent unauthorized users from understanding the content of the database. Data encryption, as one of the primary methods for protecting data, is useful both for information stored on disk and for information exchanged on a network.

7.9 User View

The user view considers the usability aspects of systems and the environment in general. It should also consider impacts on the user such as skill levels required, the need for specialized training, and migration from current practice. The user view takes into account:

- The ease-of-use of the user interfaces and how intuitive they are
- Whether there is transparent access to data and applications, irrespective of location
- Ease of management of the user environment by the user
- Application interoperability through means such as drag and drop
- On-line help facilities
- Clarity of documentation
- Security and password aspects, such as avoiding the requirement for multiple sign-on and password dialogues
- Access to productivity (infrastructure) applications such as mail or a spreadsheet
This view of the technical architecture focuses on computing models that are appropriate for a distributed computing environment and on hardware/software and networking. To support the migration of legacy systems, the section also presents models that are appropriate for a centralized environment. The definitions of many of the computing models (e.g., host-based, master/slave, and three-tiered) historically preceded the definition of the client/server model, which attempts to be a general-purpose model. In most cases, the models have not been redefined in the computing literature in terms of contrasts with the client/server model. Therefore, some of the distinctions of features are not always clean. In general, however, the models are distinguished by the allocation of functions for an information system application to various components (e.g., terminals, computer platforms). These functions that make up an information system application are presentation, application function, and data management.

**Client/Server Model**

Client/server processing is a special type of distributed computing called cooperative processing because the clients and servers cooperate in the processing of a total application (presentation, functional processing, data management). In the model, clients are processes that request services, and servers are processes that provide services. Clients and servers can be located on the same processor, different multiprocessor nodes, or on separate processors at remote locations. The client typically initiates communications with the server. The server typically does not initiate a request with a client. A server may support many clients and may act as a client to another server. Figure 7.4 shows how the client/server model can be drawn following The Open Group technical reference model, showing how the various entities and interfaces can be used to support a client/server model whether the server is local or remote to the client. In these representations, the request–reply relationships would be defined in the API.

Clients tend to be generalized and can run on one of many nodes. Servers tend to be specialized and run on a few nodes. Clients are typically implemented as a call to a routine. Servers are typically implemented as a continuous process waiting for service requests (from clients). Many client/server implementations involve remote communications across a network. However, nothing in the client/server model dictates remote communications, and the physical location of clients is transparent to the server.
An application program can be considered to consist of three parts: data management, business logic, and presentation. In general, each of these can be assigned to either a client or server application, making appropriate use of platform services. This assignment defines a specific client/server configuration (or deployment topology).

**Master/Slave and Hierarchical Models**

In this model, slave computers are attached to a master computer. In terms of distribution, the master/slave model is one step up from the host-based model. Distribution is provided in one direction, from the master to the slaves. The slave computers perform application processing only when directed to by the master computer. In addition, slave processors can perform limited local processing, such as editing, function key processing, and field validation. A typical configuration might be a mainframe as the master with personal computers (PCs) as the slaves acting as intelligent terminals, as illustrated in Figure 7.5.

The hierarchical model is an extension of the master/slave model with more distribution capabilities. In this approach, the top layer is usually a powerful mainframe, which acts as a server to the second tier. The second layer consists of LAN servers and clients to the first layer as well as servers to the third layer. The third layer consists of PCs and workstations. This model has been described as adding true distributed pro-
cessing to the master/slave model. Figure 7.5 shows an example hierarchical model in the third configuration, and Figure 7.6 shows the hierarchical model represented in terms of the entities and interfaces of the technical reference model.

**Figure 7.5.** Host-based, master/slave, and hierarchical models.

**Figure 7.6.** Hierarchical model based on reference model.
Peer-to-Peer Model

In the peer-to-peer model there are coordinating processes. All of the computers are servers in that they can receive requests for services and respond to them, and all of the computers are clients in that they can send requests for services to other computers. In current implementations, there often are redundant functions on the participating platforms.

Attempts have been made to implement the model for distributed heterogeneous (or federated) database systems. This model could be considered a special case of the client/server model in which all platforms are both servers and clients. Figure 7.7 (A) shows an example peer-to-peer configuration in which all platforms have complete functions.

Distributed Object Management Model

In this model, the remote procedure calls typically used for communication in the client/server and other distributed processing models are replaced by messages sent to objects. The services provided by systems on a network are treated as objects. A requester need not know the details of how the object is configured. The approach requires:

- A mechanism to dispatch messages
- A mechanism to co-ordinate delivery of messages
- Applications and services that support a messaging interface

This approach does not contrast with client/server or peer-to-peer models but specifies a consistent interface for communicating between

\[\text{FIGURE 7.7. Peer-to-Peer and distributed object models.}\]
cooperating platforms. It is considered by some to be an implementation approach for client/server and peer-to-peer models. Figure 7.7 presents two distributed object model examples. Example B shows how a client/server configuration would be altered to accommodate the distributed object management model. Example C shows how a peer-to-peer model would be altered to accomplish distributed object management.

The Object Management Group (OMG), a consortium of industry participants working toward object standards, has developed an architecture—the Common Object Request Broker Architecture (CORBA)—that specifies the protocol a client application must use to communicate with an Object Request Broker (ORB), which in turn provides access to distributed object services (and the actual business objects). The ORB specifies how objects can transparently make requests and receive responses. In addition, Microsoft’s COM and Java’s J2EE provide similar distributed component services based on a similar model as CORBA but not the same technology (specifications).

### 7.11 Communications View

#### Introduction

Communications networks are constructed of end devices (e.g., printers), processing nodes, communication nodes (switching elements), and the linking media that connect them. The communications network provides the means by which information is exchanged. Forms of information include data, imagery, voice, and video. Because automated information systems accept and process information using digital data formats rather than analog formats, the TOGAF communications concepts and guidance will focus on digital networks and digital services. Integrated multimedia services are included.

The communications view describes the communications architecture with respect to geography, discusses the Open Systems Interconnection (OSI) reference model, and describes a general framework intended to permit effective system analysis and planning.

#### Communications Infrastructure

The communications infrastructure may contain up to three levels of transport—local, regional/metropolitan, and global—as shown in Figure 7.8. The names of the transport components are based on their respective geographic extent, but there is also a hierarchical relationship among them. The transport components correspond to a network management structure in which management and control of network re-
sources are distributed across the different levels. Generally, available (cost-effective) bandwidth reduces by orders of magnitude as the geographic boundaries extend.

The local components relate to assets that are located relatively close together geographically. This component contains fixed communications equipment and small units of mobile communications equipment. Local area networks (LANs), to which the majority of end devices will be connected, are included in this component. Standard interfaces will facilitate portability, flexibility, and interoperability of LANs and end devices.

Regional and metropolitan area networks (MANs) are geographically dispersed over a large area. A regional or metropolitan network could connect local components at several fixed bases or connect separate remote outposts. In most cases, regional and metropolitan networks are used to connect local networks. However, shared databases, regional processing platforms, and network management centers may connect directly or through a LAN. Standard interfaces will be provided to connect local networks and end devices.

Global or wide area networks (WANs) are located throughout the world, providing connectivity for regional and metropolitan networks in the fixed and deployed environments. In addition, mobile units, shared databases, and central processing centers can connect directly to the global network as required. Standard interfaces will be provided to connect regional and metropolitan networks and end devices.

![Communications infrastructure diagram](image-url)
**Communications Models**

The geographically divided infrastructure described earlier forms the foundation for an overall communications framework. These geographic divisions permit the separate application of different management responsibilities, planning efforts, operational functions, and enabling technologies to be applied within each area. Hardware and software components and services fitted to the framework form the complete model.

The open systems interconnection (OSI) reference model, portrayed in Figure 7.9, is the model used for data communications in TOGAF. Each of the seven layers in the model represents one or more services or protocols (a set of rules governing communications between systems) that define the functional operation of the communications between user and network elements. Each layer (with the exception of the top layer) provides services for the layer above it. The OSI model specifically aims at establishing open systems operation and implies standards-based implementation. It strives to permit different systems to accomplish complete interoperability and quality of operation throughout the network.

The seven layers of the OSI model are structured to facilitate independent development within each layer and to provide for changes independent of other layers. Stable international standard protocols in conformance with the OSI reference model layer definitions have been published by various standards organizations. This is not to say that the only protocols that fit into TOGAF are OSI protocols. Other protocol standards such as SNA or TCP/IP can be described using the OSI seven-layer model as a reference.

The OSI standards are essentially defunct from an industry perspective. Although a number still remain (X.500 and X.400, for instance), very few organizations are implementing the entire OSI stack; predominately, TCP/IP has filled this niche. However, the layered concept is still very much in vogue, and OSI is now probably better known as a euphemistic representation of the generic seven-layered model.

Infrastructure and business applications, as defined in TOGAF, are above the OSI reference model protocol stack and use its services via the applications layer.

A communications system based on the OSI reference model includes services in all of the relevant layers, the infrastructure and business application software that sits above the Application layer of the OSI model, and the physical equipment carrying the data. These elements may be grouped into architectural levels that represent major functional capabilities, such as switching and routing, data transfer, and the performance of applications.
These architectural levels are:

- The transmission level (below the physical layer of the OSI model) provides all of the physical and electronic capabilities that establish a transmission path between functional system elements (e.g., wires, leased circuits, interconnects).
- The network switching level (OSI layers 1 through 3) establishes connectivity through the network elements to support the routing and control of traffic (e.g., switches, controllers, network software).
- The data exchange level (OSI layers 4 through 7) accomplishes the transfer of information after the network has been established (end-to-end, user-to-user transfer) involving more capable processing elements (e.g., hosts, workstations, servers).
- In the TRM, OSI application layer services are considered to be part of the application platform entity because they offer standardized interfaces to the application programming entity.
- The applications program level (above the OSI) includes the infrastructure and business applications (non-management application programs).

The communications framework is defined to consist of the three geographical components of the communications infrastructure (local, regional, and global) and the four architectural levels (transmission, network switching, data exchange, and application program), and it is depicted in Figure 7.10. Communications services are performed at one
or more of these architectural levels. Figure 7.10 also identifies the relationship of TOGAF to the communications architecture.

**Allocation of Services to Components**

The communications infrastructure consists of the local, regional, and global transport components. The services allocated to these components are identical to the services of the application program, data exchange, network switching, or transmission architecture levels that apply to a component. Data exchange and network-switching level services are identical to the services of the corresponding OSI reference model layers. Typically, only network-switching and transmission services are allocated to the regional and global components, which consist of communications nodes and transmission media. All services may be performed in the local component, which includes end devices, processing nodes, communications nodes, and linking media. Transmission, switching, transport, and applications are all performed in this component.

### 7.12 Summary

The ability to take various slices through an organization’s architecture and present them in different ways is central to developing and understanding the overall architecture. The technique of using architectural views supports the slicing and dicing of the current and target architec-

![Communications framework](image-url)
tures. This chapter considered a number of TOGAF-specific architectural views, encompassing all aspects of the platform service continuum. We have already seen the use of a number of these views in the previous chapter describing the current environment.

In the next chapter, we continue the assessment of the current environment using further TOGAF-specific techniques to describe the environment in a way that allows us to develop further the understanding necessary to begin the specification of the target architecture.
A computer-based information system is made up of a number of different elements:

- The *data* element. Data is the fundamental representation of facts and observations. Data is processed by a computer system to provide the information that is the very reason for the computer’s existence. As you will see, data can take on a number of different forms.

- The *hardware* element. Computer hardware processes the data by executing instructions, storing data, and moving data and information between the various input and output devices that make the system and the information accessible to the users.

- The *software* element. Software consists of the system and application programs that define the instructions that are executed by the hardware. The software determines the work to be performed and controls operation of the system.

- The *communication* element. Modern computer information systems depend on the ability to share processing operations and data among different computers and users, located both locally and remotely. Data communication provides this capability.

The combination of hardware, software, communication, and data make up the *architecture* of a computer system. The architecture of computer systems is remarkably similar whether the system is a playstation, a personal computer that sits on your lap while you work, an embedded computer that controls the functions in your cell phone or in your car, or a large mainframe system that is never actually seen by the hundreds of users who access it every day.

Even more remarkably, the basic architecture of computer systems has changed surprisingly little over the last fifty-five years. The latest IBM mainframe computer executes essentially the same instruction set as the mainframe computer of 1965. The basic communication techniques used in today’s systems were developed in the 1970s. As new as it might seem, the Internet will celebrate its fortieth anniversary in 2010. All of
this is surprising considering the growth of computing, the rapid change of technology, and the increased performance, functionality, and ease of use of today’s systems. This makes the study of computer architecture extremely valuable as a foundation upon which to understand new developments in computing as they occur.

Computer system architecture is the subject of this textbook. Each element of the system is addressed in its own section of the text, always with an eye to the system as a whole.

Part I is made up of two chapters that presents an overview of systems, and of the computer system in particular.

Chapter 1 addresses a number of issues, including

- The ways in which a knowledge of computer architecture enhances our abilities as computer users and professionals
- A simplified view of typical computer system architectures
- The basic components that make up a computer system
- The fundamental operations that are performed by computer systems

Chapter 1 concludes with a brief architectural history of the computer.

An encompassing theme throughout this text is that of systems and system architecture. The words “system” and “architecture” appear throughout this book: we talk about information systems, computer systems, operating systems, file systems, software architecture, I/O architecture, network architecture and more. You will probably take a course in System Analysis and Design sometime in your college career.

Although most people have an instinctive understanding of what a system is, it is more important for us as system professionals to understand the concepts of systems and system architecture at a deeper level than the average person. Chapter 2 offers careful definitions and examples of the concept of systems and system architecture, both generally and in the specific context of the computer systems that are the focus of this book.
1.0 INTRODUCTION

It is nearly impossible today to escape the immediate reach of computers and computer-based systems. There is probably a cell phone in your pocket or on your desk and, perhaps, an iPod as well. For many of you, your laptop or desktop computer is sitting nearby as you read this paragraph. And that’s not all. Your car probably has several embedded computers controlling various functions. Even your microwave oven and the machine that launders your clothes depend on computers to function properly. As you are probably aware, most of these machines can talk to each other, using the Internet or some other networking technology.

Indeed, the jargon of computers has become a part of common daily language. You can open a newspaper and find references to expressions such as “2 GB DDRAM” or “WXGA LCD display” or “2 MB level 2 cache” or “Wi-Fi” in articles and advertisements. (In a way, it’s scary!) The ad in Figure 1.1, taken from a Sunday newspaper flier, is typical of recent computer ads.

You’ll notice that this computer features a “Core 2 Duo Processor” CPU, 2 GB of DDR2 SDRAM memory, a 16× DVD ±RW Drive, and a 160 GB SATA hard drive. It also contains a 256 MB PCI Express graphics card among other things. But how good a system is this? Are these features important to the user? Is this the right combination of features that you need in your computer to have the computer perform the work that you wish to get done? Are there features missing that we need? Is a Core 2 Duo processor the best CPU for us? Perhaps we are paying too much for the performance that we need. Or maybe we need more. And what does “Core 2 Duo” mean, anyway? What I/O ports might you need to assure a satisfactory long-term investment of computers for your organization? Is a 16× DVD ±RW drive adequate for your work? What if you have to burn a lot of disks? What other information about this system would allow you to make a more informed decision? (For example: Hey—where’s the networking capability?)

Some of the expressions used in these articles and ads are obvious from the context. Other references may be more obscure. Presumably, everyone today knows what a “display monitor” is. But how many people know the meaning and significance of the terms “cache memory” or “multitasking” or “PCI Express bus”? Yet all these expressions have appeared recently in daily newspaper advertisements with the assumption that people would understand the meaning of the ad.

Despite the jargon, there is obviously no need to understand the inner workings of most modern computer-based systems to operate them adequately. Indeed, in many cases the presence of the computer is hidden from us, or embedded, and its operation invisible to us as users.

Even as experienced users, we can run standard software packages on a personal computer without understanding exactly how they work; we can program a computer in a high-level language without understanding the details of how the machine
executes the individual instructions; we can design and implement Web pages without understanding how the Web browser gets its pages from a Web server or how the Web server creates those pages; we can purchase a computer system from a salesperson without understanding the specifications of the system.

And yet, there is something missing. Perhaps the package doesn’t do exactly what we want, and we don’t understand the machine well enough to risk fooling around with the package’s options. Perhaps if we understood the system better we might have written and configured the program to be faster and more efficient. Perhaps we could create Web pages that load faster and work better. Perhaps the salesperson did not sell us the optimum system for our job. Or perhaps it’s nothing more than a sense of excitement that’s missing. But that’s important, too!

You are reading this book because you are a student studying to become a computer professional, or maybe you are simply a user wanting a deeper understanding of what the computer is all about. In either case, you will most likely be interacting with computer systems for the rest of your life. It’s nice (as well as useful) to know something about the tools of the trade. More important, understanding the computer system’s operations has an immediate benefit: it will allow you to use the machine more effectively.
As a user, you will be aware of the capabilities, strengths, and limitations of the computer system. You will have a better understanding of the commands that you use. You will understand what is taking place during the operation of the programs that you use. You will be able to make informed decisions about your computer equipment and application programs. You will understand more clearly what an operating system is, and how to use it effectively and to your advantage. You will know when it is preferable to do a job manually, and when the computer should be used. You will understand the most efficient way to “go online,” and what benefits might be gained from a home network. You will improve your ability to communicate with system analysts, programmers, and other computer specialists.

As a programmer, it will allow you to write better programs. You will be able to use the characteristics of the machine to make your programs operate more effectively. For example, choosing the appropriate data type for a variable can result in significantly faster performance. Soon you will know why this is so, and how to make the appropriate choices.

Computers can perform integer calculations incorrectly if the integers exceed a certain size, but they do not necessarily warn the user of the error. You will learn how this can occur, and what can be done to assure that your programs generate correct results.

You will discover that some computers will process nested loops much more quickly if the index variables are reversed. A rather surprising idea, perhaps, and you’ll understand why this is true.

You will understand why programs written in a compiled language like C++ usually run much faster than those written in interpreted program languages like BASIC or scripting languages like JavaScript.

As a systems architect or system analyst, you will be responsible for the design and implementation of systems that meet an organization’s information technology (IT) needs, recognizing that the differences in the cost and capabilities of the components that you select may have significant impact on the organization. With the knowledge gained here you will be in a better position to determine and justify the set of computer system components and the system architecture that are appropriate for a particular job and to determine the tradeoffs with other possible system architectures.

You’ll be able to assist management in making intelligent decisions about system strategy: should the company adopt a large mainframe/virtual machine system approach for its Web servers, or would a system consisting of a network of off-the-shelf blade servers provide better performance at lower cost? You’ll be better prepared to analyze the best way to provide appropriate facilities to meet the needs of your users. In an era of fast-changing technology, you’ll be more able to differentiate between simple technological obsolescence that does not affect the organization’s requirements significantly and major advances that suggest a real need to replace older equipment.

When selecting computers, you would like to purchase the computer that best meets the needs of the organization’s applications and the users. You must be able to read and understand the technical specifications in order to compare different alternatives and to match the system to the users’ needs. This book will teach you what you need to know to specify and purchase a system intelligently. You’ll know the differences between various CPU technologies and the advantages and disadvantages of each. You will learn what peripheral hardware is appropriate for your organization’s files and the trade-offs between different file system formats, what is required to build an intranet, and what the speed and
size limitations of a particular system are. You’ll be able to compare the features of Windows and Linux knowledgeably and decide which ones are important to you. You’ll be able to apply your basic understanding of computers to new technologies such as virtual machines as they appear. You’ll learn to understand the jargon used by computer salespeople and judge the validity of their sales claims.

As a system administrator or manager, your job is to maximize the availability and efficiency of your systems. You will need to understand the reports generated by your systems and be able to use the information in those reports to make changes to the systems that will optimize system performance. You will need to know when additional resources are required, and be able to specify appropriate choices. You will need to specify and configure operating system parameters, set up file systems, manage system and user PC upgrades in a fast-changing environment, reconfigure networks, provide and ensure the robustness of system security, and perform many other system management tasks. The configuration of large systems can be very challenging. This text will give you an understanding of operating system tools that is essential to the effective management of systems.

As a Web services designer, you will be able to make intelligent decisions to optimize your Web system configurations, page designs, data formatting and scripting language choices, and operating systems to optimize customer accessibility to your Web services.

In brief, when you complete this book, you will understand what computer hardware and software are and how programs and data interact with the computer system. You will understand the computer hardware, software, and communication components that are required to make up a computer system and what the role of each component in the system is.

You will have a better understanding of what is happening inside the computer when you interact with the computer as a user. You will be able to write programs that are more efficient. You will be able to understand the function of the different components of the computer system and to specify the computer system you need in a meaningful way. You will understand the options that you have as a system administrator or Web services designer.

In an era in which technology changes extremely rapidly, the architecture of the computer system rests on a solid foundation that has changed only slightly and gradually over the last sixty years. Understanding the foundations of computer system architecture makes it possible to flow comfortably with technological change and to understand changes in the context of the improvements that they make and the needs that they meet. In fact, interviews with former students and with IT executives and other IT professionals clearly indicate that a deep understanding of the basic concepts presented here is fundamental to long-term survival and growth in the field of information technology and IT management.

This type of understanding is at the very foundation of being a competent and successful system analyst, system architect, system administrator, or programmer. It may not be necessary to understand the workings of an automobile engine in order to drive a car, but you can bet that a top-notch race car driver knows his or her engine thoroughly and can use it to win races. Like the professional race car driver, it is our intention to help you to use your computer engine effectively to succeed in using your computer in a winning way. The typical end user might not care about how their computer system works, but you do.

... These are the goals of this book. So let’s get started!
1.1 THE STARTING POINT

Before we begin our detailed study of the architecture of computer systems, let us briefly review some of the fundamental principles and requirements that guide computer system design and operation.

In a simple scenario, you use your laptop or desktop personal computer to word process a document. You probably use a mouse to move around the document and to control the features of the word processor software application, and you use the keyboard to enter and modify the document text data. The word processor application program, together with your document, appears on a screen. Ultimately, you might print the document on a printer. You store the document on a disk or some other storage device.

The fundamentals of a typical computer system are readily exposed in this simple example. Your mouse movements and clicks and your keyboard entry represent input to the system. The computer processes the input and provides output to the screen, and, perhaps, to a printer. The computer system also provides a storage medium of some sort, usually a hard disk, to store the text for future access. In simplest terms, your computer receives input from you, processes it, and outputs results to the screen. Your input takes the form of commands and data. The commands tell the computer how to process the data.

Now consider a second, slightly more complex example. Your task in this example is to access a Web page on the Internet. Again, your input to the computer is via mouse and keyboard. When you type the Web page URL, however, your computer sends a message to another computer that contains Web server software. That computer, in turn, sends a Web page file that is interpreted by the browser on your computer and presented on your screen. You are probably already aware that HyperText Transfer Protocol (HTTP) is used as a standard for Web message exchanges.

The elements of this example differ only slightly from the first example. Your command inputs tell a Web browser software application on your computer what processing is to take place; in this case, your desire to access a Web page. The output from your computer is a message to a Web server on the remote computer requesting the data that represents the Web page. Your computer receives the data as input from the network; the Web browser processes the data and presents the Web page output on the screen. Figure 1.2 illustrates the layout for this example.

The major differences between this and the first example are the source of the input data and the fact that network connectivity is required between the two computers. Instead of the keyboard, the input data to be processed by your Web browser comes from a communication channel. (Note that the exact nature of the channel is not important for this discussion.) In both cases, your computer receives data input to process, and control input that determines how the data is to be processed, performs the processing, and provides output.

These two examples contain all of the key elements found in any IT system, large or small.

- An IT system consists of one or more computer systems; multiple computer systems are connected together using some type of network interconnectivity. As a matter of necessity, network interfaces must conform to standard agreements, known as protocols, for messages to be understood by both computers during a message exchange between a pair of computers. The network itself can take on a
variety of forms, provided that the interface requirements are met, and are determined by such characteristics as performance, convenience, and cost.

The work performed by an individual computer system within the IT system can be characterized by input, processing, and output. This characterization is often represented by the **Input-Process-Output (IPO) model** shown in Figure 1.3. Storage is also represented within this model. Alternatively, storage can be interpreted as output to be saved for use as future input. Storage is also used to hold the software programs that determine the processing operations to be performed. The ability to store programs and data on a temporary, short-term, or long-term basis is fundamental to the system. In Chapter 2, Section 2.2, we will show that all IT systems can ultimately be characterized by the same basic IPO model at all levels, from a single computer to a complex aggregation of computers, although the complexity of large systems may obscure the model and make it more difficult to determine the actual inputs, outputs, and processing
FIGURE 1.4
A simplified IT Computer System Layout

- Intranet web server
- Database server
- Apps. server
- Internet web server
- Firewall
- To Internet

- Sales
  - Order entry
  - Service
  - Web design
  - Support
  - Printer

- Finance
  - Financial planning
  - Accounts receivable
  - Credit
  - Accounting

- Marketing
  - Research & planning
  - Advertising

- Order Fulfillment
  - Purchasing
  - Inventory
  - Warehousing
  - Shipping
  - Printer
The components of an individual computer system consist of processing hardware, input devices, output devices, storage devices, application software, and operating system software. The task of the operating system software is to provide overall control of the individual system, including management of input, output, and file storage functions. The medium of exchange, both with users and between computers within a larger system, is data. (Note that the messages between computers in the second example are a form of data.) Figure 1.4 is a simple illustration of computer systems embedded in a larger IT system.

Figure 1.5 summarizes the basic operations that are performed during computer processing. These operations, in turn, can be reduced to the primitive operations that are also familiar to you from your understanding of programming languages. The primitive processing operations common to high-level programming languages are shown in Figure 1.6.

1.2 COMPONENTS OF THE COMPUTER SYSTEM

As noted in the previous section, there are three components required for the implementation of a computerized input-process-output model:

1. The computer hardware, which provides the physical mechanisms to input and output data, to manipulate and process data, and to electronically control the various input, output, and storage components.

2. The software, both application and system, which provides instructions that tell the hardware exactly what tasks are to be performed and in what order.

3. The data that is being manipulated and processed. This data may be numeric, it may be alphanumeric, it may be graphic, or it may take some other form, but in all cases it must be representable in a form that the computer can manipulate.

In modern systems, input entry, output display, and storage of the data and software used for processing often take place at a location different from the computer where the

**FIGURE 1.5**

Basic Computer Operations

- Input/output
- Basic arithmetic and logical calculations
- Data transformation or translation (e.g., program compilation, foreign language translation, file updating)
- Data sorting
- Searching for data matches
- Data storage and retrieval
- Data movement (e.g., movement of text or file data to make room for insertion of additional data)
actual processing occurs. In many installations, actual processing is distributed among computer systems, with particular results passed to the individual systems that require them. Therefore, we must also consider a fourth component:

4. The communication component, which consists of hardware and software that transport programs and data between interconnected computer systems.

The hardware and system software components make up the architecture of the computer system. The communication component connects individual computer systems together. The data component, and also the application software, while fundamental to the operation of the computer system, are supplied to the computer system by the user, rather than being a part of the architecture of the computer system itself. (It is useful to note, however, that application software and data structure \textit{are} often considered as part of the overall system architecture when one considers the architecture from the perspective of the organization. We explore this issue briefly in Chapter 2. Note, however, that the focus of this book is primarily on computer \textit{system} architecture, rather than on organizational system architecture.)

\textbf{The Hardware Component}

The most visible part of the computer system is obviously the hardware that makes up the system. Consider the computer system upon which you write and execute your programs. You use a keyboard and mouse to provide \textit{input} of your program text and data, as well as for commands to the computer. A display screen is commonly used to observe \textit{output}. A printer is frequently available as an alternative output to the screen. These are all physical components.

Calculations and other operations in your program are performed by a \textit{central processing unit (CPU)} inside the computer. \textit{Memory} is provided to hold your programs and data while processing is taking place. Other input and output devices, such as a disk and SD plug-in cards, are used to provide long-term storage of your program and data files. Data and programs are transferred between the various input/output devices and memory for the CPU to use.

The CPU, memory, and all the input, output, and storage devices form the \textit{hardware} part of a computer system. The hardware forms the tangible part of the system. It is physical—you can touch it, which is what the word “tangible” means. A typical hardware
block diagram for a computer is seen in Figure 1.7. In addition to the input and output devices shown in this diagram, Figure 1.8 lists some other input and output devices that are frequently seen as part of computer systems. The diagram in Figure 1.7 actually applies equally well to large mainframe computers, small personal computers, and even devices with computers embedded in them, such as PDAs, iPods, GPSs, and cell phones. Large and small computers differ primarily in speed, capacity, and the selection of peripheral devices provided. The basic hardware components and design are very similar.

Conceptually, the CPU itself is often viewed as a composition of three primary subunits:

1. The **arithmetic/logic unit (ALU)** where arithmetic and Boolean logical calculations are performed.
2. The **control unit (CU)**, which controls the processing of instructions and the movement of internal CPU data from one part of the CPU to another.
3. The **interface unit**, which moves program instructions and data between the CPU and other hardware components.

(In modern CPUs, the actual implementation is usually modified somewhat to achieve higher performance, although the basic concepts are carefully preserved. More about that later, in Chapter 8.)
FIGURE 1.8
Other Common Input/Output Devices

- Bar Code Scanners
- Optical Character Recognition Scanners
- Image Scanners
- RFID Readers
- Video and Audio Capture Devices
- TV Tuners
- Video Cameras
- SD, SmartCard, etc. Card Readers
- Fingerprint and Face Readers
- Touch Screens
- Graphics Tablets
- X-Y Plotters

The interface unit interconnects the CPU with memory and also with the various I/O (input/output) modules. It can also be used to connect multiple CPUs together. In many computer systems, a bus interconnects the CPU, memory, and all of the I/O components. A bus is simply a bundle of wires that carry signals and power between different components. In other systems, the I/O modules are connected to the CPU through one or more separate processors known as channels.

The main memory, often known as primary storage, working storage, or RAM (for random access memory), holds programs and data for access by the CPU. Primary storage is made up of a large number of cells, each numbered and individually addressable. Each cell holds a single binary number representing part of a data value or part of an instruction. The smallest addressable size of the cell in most current computers is 8 bits, known as a byte of memory. Eight bits of memory can only hold 256 different patterns, so neighboring cells in memory are nearly always combined to form groupings with a larger number of bits. In many systems, for example, 4 bytes of memory combine to form a 32-bit word. Modern computers address memory at least 4 bytes (a “32-bit” computer) or 8 bytes (a “64-bit” computer) at a time to take advantage of larger instruction and data groupings.

The amount of primary storage determines the maximum number of instructions and data words that can be loaded into memory from a peripheral device at one time. For example, a computer with 2 gigabytes (GB), actually 2,147,483,648 bytes, of memory would not be able to execute a program that requires 2.7 GB for its instructions and data unless some means is provided within the computer to load the program in sections as each section of the program is needed.

The amount of primary storage provided in a typical computer has increased rapidly as computer technology improves. Whereas 64 kilobytes (KB) of memory was considered a large amount in 1980, even the least expensive personal computers today usually have 2 gigabytes (GB) of memory or more. Large computers may provide many gigabytes of primary storage. There are programs on the market that require hundreds of megabytes (MB) of memory to execute. The inexpensive availability of increased amounts of memory have allowed the design of very sophisticated programs that would not have been possible just a few years ago.

The same is true for secondary storage. Even small personal computers provide hard disks with storage measured in tens or hundreds of gigabytes. The storage of images and video, in particular, requires tremendous amounts of storage capacity. It is not uncommon to see arrays of hard disks, even on some personal computers, providing trillions of bytes (specified as terabytes) of long-term storage.

1 Kilobyte actually equals 1024 bytes. Thus, 1 MB = 1024 × 1024 = 1,048,576 bytes × 2048 = 2,147,483,648 bytes.
The instructions that form a particular program are stored within the primary storage, then brought into the central processing unit and executed. Conceptually, instructions are brought in and executed one at a time, although modern systems overlap the execution of instructions to some extent. Instructions must be in primary storage in order to be executed. The control unit interprets each instruction and determines the appropriate course of action.

Each instruction is designed to perform a simple task. Instructions exist to perform basic arithmetic, to move data from one place in the computer to another, to perform I/O, and to accomplish many other tasks. The computer’s power comes from the ability to execute these simple instructions at extremely high speeds, measured in millions or billions or trillions of instructions executed per second. As you are already aware, it is necessary to translate high-level language programs into the language of the machine for execution of the program to take place. It may require tens or even hundreds of individual machine instructions to form the machine language equivalent of a single high-level language statement. Program instructions are normally executed sequentially, unless an instruction itself tells the computer to change the order of processing. The instruction set used with a particular CPU is part of the design of the CPU and cannot normally be executed on a different type of CPU unless the different CPU was designed to be instruction set compatible. However, as you shall see, most instruction sets perform similar types of operations. As a result, it is possible to write programs that will emulate the instruction set from one computer on a computer with a different instruction set, although a program written for the original machine may execute slowly on the machine with the emulator.

The data that is manipulated by these instructions is also stored in memory while being processed. The idea that the program instructions and data are both stored in memory while being processed is known as the stored program concept. This important concept is attributed primarily to John von Neumann, a famous computer scientist. It forms the basis for the computer architecture that is standard to nearly every existing computer.

**The Software Component**

In addition to the hardware requirement, your computer system also requires software. Software consists of the programs that tell the computer what to do. To do useful work, your system must execute instructions from some program.

There are two major categories of software: system software and application software. System software helps you to manage your files, to load and execute programs, and to accept your commands from the mouse and keyboard. The system software programs that manage the computer are collectively known as an operating system, and differ from the application programs, such as Microsoft Word, or Firefox, or the programs that you write, that you normally run to get your work done. Windows and Linux are the best known examples of an operating system. Others include Unix, Mac OS X, Sun Solaris, and IBM z/OS.

The operating system is an essential part of the computer system. Like the hardware, it is made up of many components. A simplified representation of an operating system is shown in Figure 1.9. The most obvious element is the user interface that allows
you to execute programs, enter commands, and manipulate files. The user interface accepts input from a keyboard and, in most modern systems, a mouse, touch screen, or other pointing device. The user interface also does output presentation on the display. On some systems, the output display might be simple text, but more likely the display includes a graphical user interface with a windowing system, and various gadgets for manipulating the windows.

The operating system’s **application program interface (API)**, acts as an interface for application programs and utilities to access the internal services provided by the operating system. These include file services, I/O services, data communication services, user interface services, program execution services, and more.²

Many of the internal services are provided by the **kernel** module, which contains the most important operating system processing functions. The remaining services are provided by other modules that are controlled by the kernel. The kernel manages memory by locating and allocating space to programs that need it, schedules time for each application to execute, provides communication between programs that are being executed, manages and arranges services and resources that are provided by other modules, and provides security.

The file management system allocates and manages secondary storage space and translates file requests from their name-based form into specific I/O requests. The actual storage and retrieval of the files is performed by the I/O drivers that comprise the I/O component. Each I/O driver controls one or more hardware devices of similar type.

The network module controls interactions between the computer system and the network(s) to which it is attached.

The operating system software has nearly always been stored on a hard disk, but on some smaller systems, especially lightweight laptops and embedded systems such as cell phones and iPods, a solid state disk or SD card may be used instead. On a few systems the operating system is actually provided as a network service when the system is turned on. In either case, the bootstrap or IPL (Initial Program Load) program in the operating system is stored within the computer using a type of memory known as **ROM**, or **read-only memory**. The bootstrap program provides the tools to test the system and to load the remainder of the operating system from the disk or network. Although the physical medium where the software is stored can be touched, the software itself is considered intangible.

Together, the hardware and system software provide a working computer system environment. Application software, communication support, and user data complete the picture.

²The same term (API) is also sometimes used to describe the services provided by one application to another. For example, Amazon and Google are among many companies whose application software provides API tools to allow users to extend the functionality of the original software.
The Communication Component

Very few modern computers or computer-based devices (which includes cell phones, iPods, and automobile computers, to name just a few possibilities) operate independently. Instead, they are tied to other computers directly, by modem, or through a network connection of some sort. The computers may be located physically close to each other, or they may be separated, even by thousands of miles. To work together, computers must have means to communicate with each other. The communication component requires both hardware and software to achieve this goal. Additional hardware components physically connect computers together into multiprocessing systems, or clusters, or networks, or, via telephone, satellite, or microwave, to computers at other remote locations. A communication channel provides the connection between computers. The channel may be a wire cable, a fiber-optic cable, a telephone line, or a wireless technology, such as infrared light, cellular phone, or radio-based technology such as Wi-Fi or Bluetooth. Special I/O hardware, consisting of a device such as a modem or network interface card (NIC) within the computer, serves as an interface between the computer and the communication channel. There may be additional hardware within the channel itself.

The communication component also requires additional software within the operating system of each computer to make it possible for each computer to understand what the other computers that they are connected with are saying. This software establishes the connections, controls the flow of data, and directs the data to the proper applications for use.

The Computer System

Our general description of the computer is valid for all general-purpose computer systems, and also for most devices with computers embedded in them, regardless of brand name or size. In more general terms, every computer system consists of a CPU, or central processing unit, where all the processing takes place; memory to hold the programs and data while they are being processed; and some form of input and output, usually one or more keyboards and flat-screen display devices plus one or more forms of long-term storage, usually disks, CDs or DVDs, and USB or SD plug-in memory. Most modern computer systems provide more than one CPU (or “core”) within the computer system. A single CPU can process only one instruction at a time; the use of multiple CPUs can increase processing speed by allowing instructions that do not affect each other to be executed in parallel.

The validity of our general description is true regardless of how complex or simple the computer system may seem.

As a specific example, the large z10 EC IBM mainframe computer shown in Figure 1.10 can provide complex Web services to thousands of users at a time. IBM mainframes can have dozens of CPUs working together, with up to 1.52 terabytes (TB) of primary storage. They are capable of executing instructions at a rate of tens of billions of instructions per second! The powerful z/OS operating system can keep track of hundreds or thousands of simultaneous users and divides the time among them to satisfy their differing requirements. Even in its smallest configuration, the z10 EC Model S64 system, which is the largest current model at this writing, provides at least 16 GB of memory and processes instructions at
the rate of several billion instructions per second. In addition to the CPU, there are many large I/O devices—including tape drives and high speed printers—and disks that store many billions or trillions of characters. The computer alone weighs over 5000 pounds/2200 kilograms!

In contrast, the laptop PC shown in Figure 1.11 is designed for personal use. Everything is self-contained in one package. This system only has 2 GB of primary RAM storage and operates at a small fraction of the speed of the z10 EC. A hard drive is one of many storage options. The entire system, complete with display screen, built-in webcam, multiple network connections, and battery, weighs about three pounds (1.4 kilograms, if you prefer).

Although these two systems seem very different, the difference is actually one of magnitude, not of concept. The large system operates much faster, can support much more memory, and handles more input and output much faster. It has operating system software that allows many users to share this larger resource. Nonetheless, the fundamental system architecture is remarkably similar in both cases. Even the actual processing performed by the CPU is similar.

In fact, today’s CPU operates in the same fundamental way as its CPU counterpart of fifty-five years ago, even though the construction is very different. Since computers all operate so similarly, regardless of size or type, it is not difficult today to transfer data between these different systems, allowing each system to do part of the processing for higher overall efficiency. This concept is known as distributed computing. The fact that different types of computers can work together, share files, and communicate successfully is known as open computing. Communication technology fulfills the requirements that make open and distributed computing possible.

Computers are sometimes divided into categories: mainframe computers, minicomputers, workstations, and personal computers, but these categories are less significant than they once were. The capability of today’s personal computer far exceeds the capabilities of a mainframe computer of just a few years ago.
The Sun Ultra 40 computer is an example of a workstation that is frequently used as though it were a minicomputer, or even a small mainframe. Rather than attempting to categorize a particular computer, it is usually more productive to describe its capabilities in comparison to other systems being discussed or considered.

1.3 THE CONCEPT OF VIRTUALIZATION

The word “virtual” appears frequently throughout the computer literature in many different contexts. To name a few applications of the word that appear in this text, there are virtual computers, a Java Virtual Machine, virtual memory, and virtual networks. Sometimes, a synonymous word, logical, is used instead: in networking we have logical connections. Virtual storage consists of a relationship between logical memory and physical memory.

It is not important at this point that you understand any of the specific concepts mentioned above. (In fact, we realize that you probably don’t.) Since the words virtual and logical represent a number of important concepts, however, we introduce them here.

In optics, a virtual image is the reflection that you see when you stand in front of a regular mirror. (See, for example, the cartoon at the beginning of Chapter 18.) You know that the image isn’t real. For one thing, it’s behind the wall that the mirror is mounted on. For another, you can’t touch it. In early, time-shared computing, a large central computer commonly supplied computing services to users at terminals located remotely from the computer. In a sense, it seemed as though the user had access to a computer that was all her own. Starting in the early 1970s, IBM offered the VM (virtual machine) operating system to support this concept.

The American Heritage Dictionary offers two applicable definitions of virtual that together describe the usage of the word in modern computing:

- Existing or resulting in essence or effect though not in actual fact, form, or name
- Created, simulated, or carried on by means of a computer or computer network

Wikipedia defines virtualization as “a broad term that refers to the abstraction of computer resources”.

In essence, virtual and logical are used to refer to something that appears as though it is something different. Thus, the Java Virtual Machine (JVM) uses software to simulate a real computer that works well with the Java programming language, even though the actual computer executes a different set of instructions than the JVM. A logical connection in networking offers the appearance of a direct communication link for passing data between two computers, even though the actual connection might involve a complex series of interconnections involving many computers and other devices and a variety of software to make it all look simple. The virtualization of a computer allows a single computer to appear as a multiplicity of computers, each with its own operating system and hardware resources.

1.4 PROTOCOLS AND STANDARDS

Standards and protocols are of great importance in computer systems. Standards are agreements among interested parties, often manufacturers, to assure that various system components will work together interchangeably. Standards make it possible to build
a computer with components from different suppliers, for example, knowing that a graphics card will plug properly into a connector on a motherboard and that the image representations will be consistent between the connector, the CPU, memory, and the display monitor.

Standards apply to every aspect of computing: hardware, software, data, and communications, the voltage of a power supply, the physical spacing of pins on a connector, the format of a file, the pulses generated by a mouse. Computer language standards, such as Java and SQL, allow programs written on one type of computer to execute properly and consistently on another, and also make it possible for programmers to work together to create and maintain programs.

Similarly, data format and data presentation standards, such as the GIF and JPEG image format standard, the Unicode text format standard, and the HTML and XML Web presentation standards allow different systems to manipulate and display data in a consistent manner.

Standards can arise in many different ways. Many standards occur naturally: a proprietary data format (PDF) belonging to a single vendor becomes a de facto standard due to the popularity of the product. The PDF print description language is an example of such a standard. The format was designed by Adobe Corporation to provide a way of communicating high-quality printed output between computers and printers. Other standards are created because of a perceived need in an area where no standard exists. Often a committee will form to investigate the requirements and create the standard. The MPEG-2 and MPEG-4 standards, which establish the means for the transmission and processing of digital video images, occurred in this way. The committee that designed the standard, made up primarily of motion picture engineers and video researchers, continues to develop the standard as improved techniques evolve. The JPEG photographic standard and MP3 sound standard are other examples of standards that were developed formally. Similarly, each version of HTTP has been formalized after many years of discussion by parties interested in Web communication. A nonstandard protocol or data format is limited in use to its supporters and may or may not become a standard, depending on its general acceptance. For example, DVD videos encoded in the proprietary DivX format will play on some DVD players, but not on others.

Protocols define the specific agreed-upon sets of ground rules that make it possible for a communication to take place. Except for special applications, most computers perform their operations such that each hardware or software computer unit will understand what other computer units that they are connected with are saying. Protocols exist for communications between computers, for the communications between various I/O devices and a computer, and for communications between many software programs. A protocol specification defines such communication features as data representation, signaling characteristics, message format, meanings of messages, identification and authentication, and error detection. Protocols in a client-server system assure that requests are understood and fulfilled and that responses are interpreted correctly.

Since the use of a proprietary protocol would be limited to those with permission to use it, protocols are almost always eventually standardized. Although not always the case, protocols that are not standardized tend to die out from lack of use. In fact, international standards are often created to ensure that the protocols are universally compatible. As an example, HTTP, HyperText Transfer Protocol, guides communication between Web
servers and Web browsers on the Internet. The movement of data through the Internet is controlled by a suite of protocols called TCP/IP (Transmission Control Protocol/Internet Protocol). Storage devices communicate with a computer using a protocol called SATA. There are thousands of such protocols.

New protocols and other standards are proposed and created and standardized as the need arises. XML, RSS, and SIP are all examples of protocols developed recently to meet new demands. Satellite telecasting, near-universal telephone communication, wireless communications, and the Internet all demonstrate powerful and useful technologies made possible by protocols and standards. Indeed, the Internet is a measure of the success to which protocols that govern intercommunication between computer hardware and software have been standardized throughout the world. Discussions of various protocols and standards will occur regularly throughout this book.
“Now, this is just a simulation of what the blocks will look like once they’re assembled.”

C. Covert Darbyshire/The Cartoon Bank
2.0 INTRODUCTION

In this book we discuss systems: computer systems, operating systems, file systems, I/O (sub)systems, network systems, and more. Each of these same systems is also an element with a major role in the information technology systems that form the backbone of modern organizations. Indeed, these elements—computer hardware, software, data, and communication—together represent the infrastructure of every IT system. If we are to understand the various types of systems that are the focus of this book, it is important that we first understand the concept of “system” itself, and, then, equally important, the basic architectures of the IT systems that use these elements. Only then is it possible to see clearly the role of the various system elements in the larger IT picture as we visit each in turn.

Use of the word “system” is obviously not unique to IT. In our daily lives, too, we often use the word “system” to describe things in everyday language. Our homes have electrical systems, plumbing systems, heating and air conditioning systems, and maybe for some, even, home theatre systems. There are ignition, braking, fuel, exhaust, and electrical systems in our cars. Our cities have water systems, sewer systems, and transportation systems, to name a few. Philosophers and social scientists talk about social systems and linguistic systems. The economy deals with banking systems, financial systems and trading systems, and, for that matter, economic systems. The word “system” even appears in the names of thousands of companies.

So it seems as though everyone knows what a system is, but what is a system? We use the word “system” intuitively, without thinking about the meaning of the word, so we obviously have an intuitive understanding of what a system is. IT professionals, however, spend their careers analyzing, designing, developing, implementing, upgrading, maintaining, administering, and using systems everyday. It is therefore important that we have a deeper, more formal understanding of system concepts.

In this chapter, we consider the concept of a system from an IT perspective. We investigate the characteristics and composition of systems, explain the meaning of system architecture, and show the fundamental role of systems, particularly various types of IT systems, in business. We offer examples of different types of IT systems, and show how IT systems work together to accomplish tasks and solve problems. We show how systems can themselves be composed of subsystems, where the subsystems also fit the definition of systems.

After you have studied this chapter, you should have a clear understanding of what a system is, what kinds of systems are used in IT, the purpose and goals for each of these systems, and how these systems fit together and interact with each other and with their environment. You’ll understand the concept of system architecture. This discussion will set the stage for the remainder of the book, which considers individually and collectively the specific computer-based systems and subsystems that constitute the primary tools and components of business information technology.
The most important characteristic that is shared by all of the systems mentioned above, and, indeed, by all systems, is that each is built up from a set of components that are linked together to form what we think of as a single unit. The house plumbing system, for example, consists of sinks, faucets, toilets, a hot water heater, bathtubs or showers, valves, and more, all connected together by pipes. An IT system consists of groups of computer hardware, various I/O devices, and application and system software, connected together by networks.

Often, the system is intended to serve a purpose or to produce results. The purpose of the house plumbing system is to allow the residents of the home access to water to wash, bathe, and drink. The purpose of an IT system is to allow organizations to process, access, and share information. The results of a successful IT system are documents, information, improved business processes and productivity, profits, strategic plans, and the like. This is, in fact, the “output” of the IPO model described in Chapter 1. In general, though, there is no requirement that a system serve a specific, definable purpose. The fact that the set of components may be considered as a single unit is sufficient to satisfy the concept of a system. The solar system is an example of a system where the purpose is unspecified.

There is also no requirement that the components of a system be physical. The links between components can also be physical or conceptual. In fact, the system itself may be conceptual, rather than physical. The number system is an example of a conceptual system. Computer operating systems are also conceptual, rather than physical. Business systems are also conceptual, although some of the components that they contain may be physical. The words tangible and intangible are sometimes used in place of physical and conceptual, respectively. Intangible or conceptual components and systems include ideas, methods, principles and policies, processes, software, and other abstractions. If, for example, the components in a system represent steps (intangible) in a multistep process, the links may represent the need to complete one step before the next is begun (also intangible).

Figure 2.1 illustrates a number of different systems to show you some of the possibilities. Figure 2.1(a) is a model of a home plumbing system. This is a physical system. The components are plumbing fixtures, linked by pipes. Figure 2.1(b) is a simplified representation of the solar system. The sun and planets are physical; the links in this system are conceptual, specifically, the distance of each planet from the sun, interplanetary and solar gravity, orbital relationships, the distances between planets at a particular point in time, and other attributes. Figure 2.1(c) is a diagram of a home networking system. The links in this case are a mixture of physical wires and (intangible) wireless connections. Sometimes the nature of the links is important only in terms of providing the proper interface connections to the components. Figure 2.1(d) is a simplified diagram of part of the inventory control portion of a sales system. The relationships between the components in this case are temporal (i.e., related to time). For example, inventory from a previous sale must be deducted from stock before we process the next order; otherwise we can’t promise delivery of goods on the new order because we don’t know if we still have sufficient goods in stock to fill the order.

With these pictures and ideas about systems in mind, we will define a system as follows:

A system is a collection of components linked together and organized in such a way as to be recognizable as a single unit.
FIGURE 2.1(a) Plumbing System Diagram

- Drain
- Water supply system: Air chamber, Relief valve, Main service pipe from water supply
- Water meter
- Main shut-off
- Shutoff: 3/4” to 1” supply pipe
- Shutoff: Main service pipe from water supply
- Shutoff: Drain
- Main shut-off
- Shutoff: Water supply system
PART ONE AN OVERVIEW OF COMPUTER SYSTEMS

FIGURE 2.1(b)
The Solar System

Mercury  Earth  Jupiter  Neptune
Venus  Mars  Saturn  Uranus
Sun

FIGURE 2.1(c)
A Typical Home Network System

Phone line or cable  DSL or cable modem  Wireless router

Network-Attached Storage (NAS)  Network-Ready Printer
A general representation of a system is shown in Figure 2.2.

The linked components that constitute a system also define a boundary for the system. Anything outside the boundary represents the environment that the system operates or presents itself within. The environment may interact with and affect the system in various ways. The reverse is also true. The interface between the system and its environment is an important characteristic of the system. If the interface is well-defined, it is often possible to replace the existing system with a different system, as long as the interface between the system and the environment remains constant. This idea can have important implications when designing IT systems. For example, in a particular IT installation, a single large computer may be functionally the same as a network of small computers. When we define inputs and outputs for a system, the environment is the source of the input and also the receiver of the output.

As an example of the relationship between a system and its environment, consider the rather simplistic view of an e-business system illustrated in Figure 2.3. The organization represented by this illustration purchases goods from suppliers and makes them available for sale. (The value-adding component in the figure consists of various operations that make it worthwhile to buy from this organization, rather than directly from the supplier. For example, Amazon.com makes it possible to buy a wide variety of books from one source, rather than having to place separate orders from a number of different suppliers.) The environment for this system consists of customers who purchase from the system, suppliers
to the system, governments who control the legal aspects of the business and collect taxes, employees and prospective employees, external support personnel (such as repair people), financial resources, and others. The primary interfaces for this system are system input from suppliers and system output to purchasers; however, there are additional, more subtle interfaces to be considered, including legal, cultural, and financial interactions with the system. For example, sensitive cultural and language issues that offend potential customers on a website might have an important impact on an organization’s sales.

When analyzing a system, the components of the system may be treated as irreducible or they may themselves be representable as systems. When considered in the context of a particular system, these components would be viewed more accurately as subsystems. A business IT system, for example, might have marketing, manufacturing, purchasing, inventory, finance, and accounting subsystems, among others. Even these components might be expanded. The marketing subsystem might be further broken down into sales, development, and advertising components, as one possibility. The level of detail to be considered depends on the context in which the system is being considered, discussed, evaluated, or used. The division of a system or subsystem into its components and linkages is called decomposition. Decomposition is inherently hierarchical. The ability to decompose a system hierarchically into subsequent sets of components and subsystems is an important property of systems.
The fundamental properties, and the patterns of relationships, connections, constraints, and linkages among the components and between the system and its environment are known collectively as the architecture of the system. Some people choose to differentiate the architecture of a system from the organization of a system. The assumption is that the architecture is fundamental to the meaning and value of the system whereas the organization is one of possibly many combinations of components and linkages that meets the requirements of the architecture. The difference is subtle and often unimportant.

It is common to represent systems and their components by models or drawings on paper or objects within a computer program. These representations are abstractions. They represent the real system but are not actually the real system. (For example, the solar system does not fit conveniently inside a computer!) It should be obvious to you that all of the illustrations of systems in Figures 2.1, 2.2, and 2.3 are abstractions.

The primary reason for humans to group components into systems and to represent them as abstractions is to simplify understanding and analysis, particularly if the individual components are numerous and complex. We can study the relationships between the different components without the distraction created by the details of individual components. We can decompose, isolate and study individual components when required. We can study the interactions between the environment and the system as a whole. Effectively, our analyses are simplified by eliminating factors that are not relevant in the context of our interests. In a large network of computers, for example, we may be concerned primarily
with the flow of data between computers. The details of the individual computers are unimportant. In general, dealing with models at the system level allows us to isolate and focus on the specific elements of interest more easily, by treating other elements collectively.

To escape our fixation on information technology systems for an instant, consider, just for fun, the solar system that we’ve used previously as an example. If we are studying the Milky Way galaxy, it is convenient and sufficient to treat the solar system as a single irreducible component in the galaxy. We might be interested in the location and movement of our Sun in the galaxy, for example, but the structure of the planets is irrelevant to our study in this case. On the other hand, if we are interested in studying the effects of the tides on a sea shore where we are planning to vacation, we will have to expand the “Earth” component and look at the specific effects of the moon and other nearby objects as part of our analysis.

Consider, too, the role played by decomposition and the ability to isolate and study individual components. A complex system may be divided into relatively independent components and analyzed by different individuals, each a specialist in their own area. Thus a plumber can create a home water system component without concern for the details of the electrician’s efforts. They can work together on the linkages that concern both of them, for example, the wiring for the boiler in a hot water heating system. The system architect coordinates the different efforts. The role of an IT system architect is similar: to work with finance experts on the finance component, marketing experts on the marketing component, and so forth.

When the goal of a project is to implement a system of some type, it is sometimes convenient to view the components of a system as modules that can be implemented independently, then linked together to produce the final result. This technique can simplify analysis, design, assembly, upgrading, and even repair. It also supports collaboration during the design process, since individual components can be designed by different individuals using specifications for the interfaces.

For example, a cell phone might consist of a computer control module, a memory module, a display module, an audio input/output module, a radio transmitter/receiver module, a keypad/text input module, and a wireless networking module. Each component might have been developed by a different team. These modules, designed, constructed, and manufactured as individual assemblies, properly interfaced, wired together, and mounted into a case, constitute the design of a typical cell phone. They also represent the components that might appear in the system diagram for a cell phone. The same approach might be taken with a computer system, with a central processor module, a graphics display module, an audio module, a network module, a hard drive controller module, and so on. Figure 2.4, for example, shows the basic system hardware components that make up an iPhone.

It is also important to realize that there may be many different representations for a system, to reflect the various uses of the system model. Returning to our IT roots for an example, the representation of the business system shown in Figure 2.5(a) is a traditional hierarchical organization chart. The components are departments that perform various functions within the business. In contrast, a partial model of the same business shown in Figure 2.5(b) represents the application architecture of an IT system within this business. Take another look at Figure 1.4 for still another representation of a business. As another simple example, you could represent a house by the physical appearance of its exterior, by the function and layout of its rooms, or by the various subsystems, electrical, plumbing, heating, and so on that the house requires. Presumably, each of these representations would be useful to a different participant. In fact, we would expect an architect to provide all of these for use by the owner, the builder, and the various contractors.
FIGURE 2.4

iPhone Components

Flash memory
CPU
Main circuit boards
Communications
GSM cell, WiFi, EDGE
Battery
Display (rear)

Courtesy Christopher Harting.

FIGURE 2.5(a)

Business Organization Chart

Corporate exec. management

Marketing and sales
  - Sales
  - Advertising
  - Planning
  - Order Fulfillment

IT
  - System planning & development
  - System Administration
  - User support

Human Resources
  - Employment
  - Organizational development
  - Contracts

Finance
  - Accounting
  - Financial planning
  - Purchasing
  - Auditing & control
2.2 IT SYSTEM ARCHITECTURES

The use of system concepts is particularly applicable when discussing the various types of IT systems. In general, the goal of IT systems is to assist organizations to meet the strategic needs of the enterprise. Not surprisingly, IT systems are frequently complex, and the ability to separate them naturally into subsystems or components of manageable size simplifies understanding of the system as a whole. The analysis, design, and implementation of IT systems must take place at different levels of detail and frequently require collaboration among many analysts and designers. This corresponds well with the ability to decompose systems into components, hierarchically, which allows us to concentrate at the appropriate
levels of detail during each step along the way. This approach is known as a top-down approach. The top-down approach allows us to focus on the specific areas of interest without the distraction of details that are irrelevant for the level that we’re studying. In this way, a system architect can analyze and study the IT system as a whole, encapsulating the computer systems, software systems, network architecture, and Web architecture that represent components, and focusing instead on the large picture: the purpose of each component and the requirements for the interfaces and linkages that connect and integrate them. With the IT system architecture firmly established, we can consider the individual business functions, computer systems, and networks that will link them together. For IT system analysis, this is often sufficient, at least superficially, assuming that the system architects actually understand the conditions and constraints imposed by details at the lower levels.

Although there are other, equally valid, approaches to IT system analysis and design, and many other important considerations as well, this approach suits the purposes of this book well because it allows us to establish general requirements for IT systems and then to show how the specific capabilities and characteristics of computer hardware, operating systems, networks, and data fulfill those requirements.

With these ideas in mind, let us return to the simple word processing example from Chapter 1 and reconsider it from a system architecture perspective. Recall that in this example you are sitting at your computer typing text into a word processor. We noted that the computer accepted input from your mouse and keyboard, processed it according to rules established by the application software, and produced output, which appeared on a display. From the system perspective, we can, for now, treat the whole computer, keyboard, display, printer, storage, software, and all as a single component. You’re the relevant part of the environment for this discussion. Forgetting the issue of control for now, the system has an input and an output. Both of these interface with you, the environment. The data for this interface is alphanumeric text in human-readable form. Other input data to the document might include graphics drawn with the mouse, photographic images from a digital camera, bar codes, or music from an iPod or other audio source. We described this scenario earlier, in Chapter 1, as input-process-output.

A system this simple is unlikely to meet all the needs of even the smallest enterprise or, even, the least computer-literate individual. But it does serve as a starting point to recognizing the value of a system approach to the understanding of information technology.

**Distributed Processing Systems**

Realistically, modern IT system architectures generally rely on multiple computers connected by networks of communication channels to achieve their goals. In all but the smallest organizations, input data is collected from locations scattered throughout the organization, stored, processed, and distributed to other locations within the organization. Since modern computer hardware and networking equipment is plentiful and inexpensive, it is practical to distribute computing capability to everyone who needs it. Furthermore, the availability of the Internet and alternative structures, such as satellite communications, make global data communication practical. Web access, organization intranets, e-mail capability, analysis tools, such as Microsoft Excel, and document preparation tools are widely available and are considered essential business tools throughout most organizations. Collaboration
between different organizations, particularly in the area of automated business-to-business purchasing and sales, is commonplace.

Therefore, when envisioning effective IT systems, designers typically must create system architectures that are capable of supporting large numbers of user workstations who will have ready access to the organizational information that they need. The system must be able to reliably store and protect large amounts of organizational data. For many organizations, customers outside of the organization may also need access to the system to get information and to make purchases.

Consider a few typical simple scenarios:

- A global fast food chain collects data each day from each of its restaurants worldwide to establish sales figures and determine sales trends. This allows the company to determine which locations are most productive and which locations need assistance, which items sell best and which need to be modified or replaced, and so on.

- A large travel organization conducts much of its business online, using travel agents located all over the world. It maintains Web servers that have immediate access to large databases of client information and travel information, as well as continual and instant access to airline and hotel reservation systems to determine current airfares, seat availability, and hotel room availability. All of this information must be immediately accessible to every agent and must be current at every instant. Even brief system failures are very costly.

- A large Web-based retail sales organization sells large quantities of a wide variety of merchandise. (Think Amazon or Wal-Mart.) Orders initially come in to a central facility, where they are billed. Inventory is stored in warehouses in various countries and local regional areas to expedite delivery and reduce delivery costs. The system must be able to distribute orders to the various regional facilities efficiently; it must also maintain appropriate levels of goods at each warehouse to match sales and must be able to locate goods and arrange shipping in response to orders as they come in.

  Inventory replenishment is handled by an automated purchasing IT system component that is integrated with the IT systems of the suppliers. Purchase order data is passed from the retailer to a supplier, which triggers order placement, billing and shipment components in the supplier’s systems. Web technology is commonly used to satisfy the need for data and communication compatibility between the systems.

- Even conventional business order processing is inherently distributed within an organization. A purchase order, for example, might be entered into the system by a salesperson in the sales department; the order is checked by order fulfillment for inventory, then distributed to the accounting department for a credit check and billing, and sent to the warehousing area for packaging and shipping. Back orders and inventory replenishment are sent to the purchasing department. For planning and marketing purposes, data will be collected into a central location and processed into sales figures, inventory planning and purchase requirements data, and the like. In a large organization, these functions might be widely scattered over a city, country, or even the world.
The emphasis in each of these scenarios is the flow and processing of data within an organization or between organizations or between an organization and its environment. The system architecture representation of such operations is called application architecture. Application architecture is primarily concerned with the activities and processing of application programs and the communications between them. Since the application architecture addresses the fundamental business needs of the organization, the application architecture is typically the primary consideration in IT system design. Therefore, the system requirements and constraints set by the application architecture have major impact on the hardware architecture and network architecture requirements for the system. Within the application architecture realm the selection and layout of computer systems and communication networks is of concern primarily to the extent that it adequately supports the application software and its functionality. However, additional factors such as scalability, convenience, information availability, data security, system administration, power and space requirements, and cost may also play important roles in computer and network architectural designs.

CLIENT-SERVER COMPUTING  There are a variety of possible application architectures that can satisfy the requirements of modern organizations. Most, however, are based on different applications of a simple technological concept, the client-server model.

In a client-server configuration, a program on a client computer accepts services and resources from a complementary program on a server computer. The services and resources can include application programs, processing services, database services, Web services, file services, print services, directory services, e-mail, remote access services, even computer system initial startup service. In most cases, the client-server relationship is between complementary application programs. In certain cases, particularly for file services and printer sharing, the services are provided by programs in the operating system. Basic communication and network services are also provided by operating system programs.

Basic client-server architecture is illustrated in Figure 2.6. Notice that the link between client and server is essentially irrelevant within the application architecture view of the system. The “cloud” in the figure is intended to indicate only that there is a link of some kind between the client and the server. The link can be a network connection, an intranet.
or Internet connection, or some sort of direct connection. In fact, a single computer can act as both client and server, if desired. (A situation where this is the case is described in Chapter 16.)

The client-server model describes the relationship and behavior of programs in one or two computers under particular prescribed circumstances. It is important to understand that the client-server model does not require any special computer hardware. Furthermore, networking software within the operating system of each computer routinely provides basic communication capabilities. The only “special” software required is the software within the complementary application programs that provides the communications between the programs. The requests and responses take the form of data messages between the client and server that are understood by both application programs. As an example, slightly simplified, the HTTP request message sent to a Web server by a Web browser requesting a Web page consists of the word GET followed by a URL. If the request is successful, the message returned by the server contains the HTML text for the page.

From the description and the figure you can see that the Web browser–Web server application described as an example in Chapter 1 fits the description of a client-server application. We will return to this example momentarily.

A typical use of the client-server concept within an organization is shown in Figure 2.7. In this case, a number of clients are sharing a number of servers, showing both the shared server nature of client-server computing, as well as to show that there may be multiple servers offering different services on the same network. Notice, also, that the server computer labeled S2 in the figure is running two different server applications. Since computers are capable of running multiple tasks concurrently, this is a possible scenario. The only limitations to running multiple applications on a single server are the potential slowdowns that may result from the load on the server computer and the traffic on the network to that server. Overall, there is a multiple-multiple relationship between clients and servers: a server can serve multiple clients, and a client can request services from multiple servers.

The use of client-server processing as a basis for IT system architecture has a number of advantages:

- Providing services on a single computer or on a small number of computers in a central location makes the resources and services easy to locate and available to everyone who needs them, but also allows the IT administrators to protect the resources and control and manage their use. The consistency of files and data can be managed and assured.

  For example, client-server technology can ensure that every user requesting a particular program from a server will receive the same version of the program. As another example, suppose a program has a license that limits the number of simultaneous users. The program server can easily limit distribution of the program appropriately.

- The amount of data to be stored, processed, and managed may be extremely large. It is more efficient to equip a small number of computers with the power needed than to require powerful computers at every station.

- Typically, humans request information from knowledgeable sources as they need it. Thus, the client-server approach is naturally consistent with the way humans acquire and use information.
The most familiar example of the use of client-server technology is the Web browser–Web server model used in intranets and on the Internet. In its simplest form, this model is an example of two-tier architecture. Two-tier architecture simply means that there are two computers involved in the service. The key features of this architecture are a client computer running the Web browser application, a server computer running the Web server application, a communication link between them, and a set of standard protocols, in this case, HTTP, for the communication between the Web applications, HTML for the data presentation requirements, and, usually, the TCP/IP protocol suite for the networking communications.

In the simplest case, a Web browser requests a Web page that is stored as a pre-created HTML file on the server. More commonly, the user is seeking specific information, and a custom Web page must be created “on the fly”, using an application program that looks up the required data in a database, processes the data as necessary, and formats it to build the desired page dynamically.

Although it is possible to maintain the database and perform the additional database processing and page creation on the same computer as the Web server, the Web server in a
large Internet-based business may have to respond to thousands of requests simultaneously. Because response time is considered an important measure by most Web users, it is often more practical to separate the database and page processing into a third computer system. The result, shown in Figure 2.8, is called a **three-tier architecture**. Note that, in this case, the Web server machine is a client to the database application and database server on the third computer. CGI, the *Common Gateway Interface*, is a protocol for making communication between the Web server and the database application possible. (In the figure, we have placed the page creation application software on the database machine, but it could be located on the Web server instead if doing so would balance the loads on the two machines better.) In some situations, it is even desirable to extend this idea further. Within reason, separating different applications and processing can result in better overall control, can simplify system upgrades, and can minimize scalability issues. The most general case is known as an **n-tier architecture**.

Client-server architecture is a distributed processing methodology, in which some of the processing is performed on the client system and some is performed on the server system. To see this more clearly, consider the distribution of processing between the client and server in a database application, in which the client requests specific information from a database stored on a database server.

At one extreme, the client application provides little more than a request form and a means to display the results. All of the processing is performed on the server. This might be appropriate if there is little computing power in the client. Certain so-called “thin” clients or “end-user” terminals might meet this criterion, but this situation is increasingly rare. Because this extreme case puts the entire processing load on the server, the system designer will have to specify a more powerful computer for the server; additionally, the requirements of the database server may limit the capability of the server computer system to perform other tasks or to scale for increased usage.

At the other extreme, the database server application simply accesses data from the database and passes all of the data to the client. The client application performs all of the processing. This relieves the load on the server, and it is reasonable to assume that modern client computers would be able to handle most database processing tasks relatively easily. However, the potential transfer of large amounts of raw data from the server to

**FIGURE 2.8**

Three-Tier Database Architecture

*CGI: Common Gateway Interface*
the client for processing may put an extra burden on the network instead, requiring the
system designer to specify higher speed network components at potentially higher cost and
additional implementation difficulty.

A well-designed system analysis will consider the different factors, the complexity
of the applications, expected network traffic, usage patterns, and the like. The optimum
solution is likely to fall somewhere in the middle, with some pieces of applications on the
server, others on the client.

One of the strengths of client-server architecture is its ability to enable different
computer hardware and software to work together. This provides flexibility in the selection
of server and client equipment tailored to the needs of both the organization and the
individual users. One difficulty that sometimes arises when different computers have
to work together is potential incompatibilities between the application software that
resides on different equipment. This problem is commonly solved with software called
middleware. Middleware resides logically between the servers and the clients. Typically,
the middleware will reside physically on a server with other applications, but on a large
system it might be installed on its own server. Either way, both clients and servers send
all request and response messages to the middleware. The middleware resolves problems
between incompatible message and data formats before forwarding the messages. It also
manages system changes, such as the movement of a server application program from one
server to another. In this case, the middleware would forward the message to the new
server transparently. The middleware thus assures continued system access and stability. In
general, the use of middleware can improve system performance and administration.

WEB-BASED COMPUTING  The widespread success of the World Wide Web has
resulted in a large base of computer users familiar with Web techniques, powerful
development tools for creating Web sites and Web pages and for linking them with
other applications, and protocols and standards that offer a wide and flexible variety
of techniques for the collection, manipulation, and display of data and information. In
addition, a powerful website is already a critical component in the system strategy of
most modern organizations. Much of the data provided for the website is provided by
architectural components of the organization’s systems that are already in place.

Not surprisingly, these factors have led system designers to retrofit and integrate Web
technology into new and existing systems, creating modern systems which take advantage
of Web technology to collect, process, and present data more effectively to the users of the
system.

The user of a Web-based system interacts with the system using a standard Web
browser, enters data into the system by filling out Web-style forms, and accesses data
using Web pages created by the system in a manner essentially identical to those used
for the Internet. The organization’s internal network, commonly called an intranet, is
implemented using Web technology. To the user, integration between the intranet and the
Internet is relatively seamless, limited only by the security measures designed into the system.
This system architecture offers a consistent and familiar interface to users; Web-enabled
applications offer access to the organization’s traditional applications through the Web.
Web technology can even extend the reach of these applications to employees in other parts
of the world, using the Internet as the communication channel.
Since Web technology is based on a client-server model, it requires only a simple extension of the $n$-tier architecture to implement Web-based applications. As an example, Figure 2.9 shows a possible system architecture to implement Web-based e-mail. Note the similarity between this example and the three-tier database application shown in Figure 2.8.

Many organizations also now find it possible and advantageous to create system architectures that integrate parts of their systems with other organizations using Web technology and Web standards as the medium of communication. For example, an organization can integrate and automate its purchasing system with the order system of its suppliers to automate control of its inventory, leading to reduced inventory costs, as well as to rapid replacement and establishment of reliable stocks of inventory when they are needed. Internet standards such as XML allow the easy identification of relevant data within data streams between interconnected systems, making these applications possible and practical. This type of automation is a fundamental component of modern business-to-business operations.

**PEER-TO-PEER COMPUTING**  
An alternative to client-server architecture is **peer-to-peer architecture**. Peer-to-peer architecture treats the computers in a network as equals, with the ability to share files and other resources and to move them between computers. With appropriate permissions, any computer on the network can view the resources of any other computer on the network, and can share those resources. Since every computer is essentially independent, it is difficult or impossible to establish centralized control to restrict inappropriate access and to ensure data integrity. Even where the integrity of the system can be assured, it can be difficult to know where a particular file is located and no assurance that the resource holding that file is actually accessible when the file is needed. (The particular computer that holds the file may be turned off.) The system also may have several versions of the file, each stored on a different computer. Synchronization of different file versions is difficult to control and difficult to maintain. Finally, since

---

**FIGURE 2.9**

Three-Tier Web-Based E-Mail Architecture

*SMTP: Simple Mail Transfer Protocol
*CGI: Common Gateway Interface
data may pass openly through many different machines, the users of those machines may be able to steal data or inject viruses as the data passes through. All of these reasons are sufficient to eliminate peer-to-peer computing from consideration in any organizational situation where the computers in the network are controlled by more than one individual or group. In other words, nearly always.

There is one exception: peer-to-peer computing is adequate, appropriate, and useful for the movement of files between personal computers or to share a printer in a small office or home network.

Peer-to-peer technology has also proven viable as an Internet file sharing methodology outside the organizational structure, particularly for the downloading of music and video. The perceived advantage is that the heavy loads and network traffic associated with a server are eliminated. (There are legal ramifications, also, for a server that is sharing copyrighted material illegally.) This technique operates on the assumption that the computer searching for a file is able to find another computer somewhere by broadcasting a request across the Internet and establishing a connection with a nearby computer that can supply the file. Presumably, that computer already has established connections with other systems. All of these systems join together into a peer-to-peer network that can then share files. One serious downside to this approach, noted above, is the fact that the computers in an open, essentially random, peer-to-peer network can also be manipulated to spread viruses and steal identities. There are several serious documented cases of both.

An alternative, hybrid model uses client-server technology to locate systems and files that can then participate in peer-to-peer transactions. The hybrid model is used for instant messaging, for Skype and other online phone systems, and for Napster and other legal file download systems.

Although there have been research studies to determine if there is a place for peer-to-peer technology in organizational computing, the security risks are high, the amount of control low, and the overall usefulness limited. The results to date have been disappointing.

The Role of the System Architect

In Section 2.1, we suggested that there are different ways of viewing systems. From the discussion within this section, you can see that the IT system architect must consider the system from the perspectives of application architecture, data architecture, network architecture, and computer architecture. Each of these addresses different aspects of the IT system as a whole. For example, our consideration of different general application architectures—client-server, web-based architecture, peer-to-peer architecture—ignored the networking that links the various computers together. Similarly, we attempted to minimize the effects due to the specifics of individual computer systems when exploring the various requirements of a system from the perspective of application architecture.

Ultimately, it is the responsibility of the system architect to assess the particular needs of an organization and create a system that meets those needs while attempting to achieve an optimum balance of computer power, network capability, user convenience, and budget. To do so, the architect will consider each aspect of the system: application architecture, network requirements, specification of computer systems, and data requirements, just as the architect designing a house considers flow of people through the house, overall use of
space and room layout, individual room layouts, mechanical systems, and aesthetic design as different views of the overall architecture of the house.

Although the infrastructure design as defined by the computer hardware, system software, and communication channels is subordinate to the fundamental business requirements that determine a basic IT system architecture, the system architect must understand the features and constraints that establish the feasibility and desirability of a particular infrastructure configuration.

**Google: A System Architecture Example**

So far, we have considered basic system concepts and simple system architectures as examples. Most IT business systems operate primarily within an organization, with limited collaboration with other, partnered organizations and carefully controlled public access. At the opposite extreme are massive systems that are widely open to the public. Google offers a primary example of such a system.

The primary mission of Google is to provide powerful, fast search capability of material on the Internet for billions of users all over the world. Income to the organization is provided from advertising that is targeted to each user based on the specific nature of the user’s search. The design of Google’s IT system architecture is obviously fundamental to Google’s ability to achieve its mission and to meet reasonable income goals. In keeping with the focus of this book, our primary interest is in the computer and network architectures that Google uses to meet its system requirements; however we will use this example to explore the relationship between the basic system requirements, the IT system architecture created to meet those requirements, and the specific computer and network architectures that evolved from the system architecture.

Some of the basic requirements that the Google IT system must satisfy include the following:

- It must be capable of responding to millions of simultaneous requests from all over the world with pertinent, ranked search results and appropriately targeted advertising. Most desirably, the results and advertising would be matched in language, geographic suitability, and culture as much as possible to the location of the user.

- The system must be able to troll the Internet systematically and thoroughly to retrieve data and to organize the data in such a way as to make it readily available for response to user requests. There must be a processing mechanism to establish a ranking of the results to a request.

- The system must respond to requests with a reliability as near to 100 percent as is technically possible. Individual hardware and software component failures within the system must not affect system performance adversely.

- The system must be easily scalable to handle ever-increasing numbers of requests and must be cost effective.

At the application level, the requirements identify three specific processing tasks that the system must fulfill:

1. The system must accept search requests from users, identify and rank matches, create a Web page, and serve it to the user.
2. The system must collect data—lots of data! This task “crawls the Web”, identifies the search terms (every significant word) on every Web page it encounters, and maintains an index database connecting each term to the corresponding page. It likewise stores every Web page in a Web page database and assigns a ranking value to each entry.

3. The system must manage advertisements, identify appropriate advertisements in response to user search requests, and make the advertisements available to the Web page creation application mentioned in 1.

For this discussion we will focus on the processing of search requests. When a user types the Google URL www.google.com into her browser, the Web browser uses a service called Domain Name Service (DNS) to identify the IP address of the Web server to which the request is to be sent. Because Google must be able to handle several million requests per hour, Google provides a number of alternative IP addresses representing different sites to which the request may be redirected. Based on the approximate location from which the request was sent, the request is routed by DNS to a Google data center near that location. Google maintains more than forty separate data centers around the world to serve user requests.

A simplified system diagram of the application architecture for a Google data center is shown in Figure 2.10. All of the data centers are architecturally identical, differing only in such details as the number of processors and the hardware specifications for each processor. Each data center processes requests independently. Multiple copies of all of the

![Google Data Center Search Application Architecture](image-url)
index word data and Web page data are stored locally at every data center, and updated from master data at regular intervals.

A request enters the system from the Internet and is distributed to a Google Web server for processing. A request consists of words and phrases. There are many separate Web servers available so that many requests can be processed in parallel. The words are passed to a spell checker, to an ad server, and to a pool consisting of a large number of index servers.

The spell checker checks each word and considers possible alternatives if it believes that the user may have intended something different. When appropriate, the output of the spell checker will become part of the response sent to the user. ("Did you mean . . . " is familiar to most Google users.) The ad checker searches for words in the advertising database that match the user’s request and adds the corresponding advertisement(s) to the material that will be used to create the response page.

The index servers look up each word from the request in the index database and compile a list of matching pages for each word. The list is then adjusted for multiple words and phrases and sorted in order of relevance, based on Google’s ranking algorithms. This list is then passed back to the Web server.

Next, the Web server calls upon the document servers to look up each matching page in the Web page database. The document servers return a URL, a title, and a short snippet of text for each document to the Web server. Finally, the Web server creates an HTML document from the spelling, ad, and matching page results and returns the page to the user’s Web browser.

Although the application processing just described is relatively straightforward, the implementation of this system presented a number of challenges to the system architects, The index and document databases are both massive in size. Many searches will result in a large number of “hits”; each hit must be evaluated and ranked. Each hit requires retrieval and processing of a separate page from the document database. All of this processing must occur very quickly. And the numbers of searches occurring simultaneously may also be extremely large.

Google’s system architects responded to these challenges by recognizing that each search could be processed independently on a separate computer, except for certain bottlenecks. For example, each search request arriving from the Internet could be steered by a computer to a different Web browser. They also observed that the major bottleneck was the time required to access the databases on disks, which had to be shared among all the searches taking place. Since the data in the databases never changed as a result of a search, however, they reasoned that the databases could also be replicated and accessed in parallel.

A simplified hardware representation of their solution is shown in Figure 2.11. Groups of up to eighty computers are connected together in a network, then these networks, up to sixty-four of them, are, themselves, connected together to form a larger network, sort of like a miniature Internet of up to 5,120 computers. (There are additional switches and connections built in for reliability that are not shown in the diagram.) Each computer acts as a server, with different computers assigned to different pieces of the application architecture. Each data center is equipped similarly.

Although the computers are manufactured specifically for Google, they are essentially inexpensive commodity PCs, similar to standard, medium power, non-state-of-the-art,
off-the-shelf PCs. Each computer has a fairly large, but still off-the-shelf, hard disk. The index and document databases are divided up among the hard disks on many computers. (Google calls these partial databases shards.) This design allows different searches to access different parts of the databases simultaneously. There are multiple copies of each database, so that the failure of a PC or hard disk does not affect the overall ability of the system to conduct searches. Each computer runs standard Linux operating system software, but the application software was specially written by Google programmers.

Overall, this design allows a large number of searches to progress in parallel. The use of inexpensive PC hardware makes the solution cost-effective. The system can be scaled easily by adding more computers. Finally, the failure of a PC does not result in failure and, in fact, has minimal effect on the performance of the system overall. Thus, this solution meets the original requirements admirably. It is worth noting that a fundamental understanding of computer infrastructure was key to the system architects’ solution.

This discussion provides a simple overview of the Google system. Hopefully you found even this brief look at the Google system interesting and informative. There are a number of other considerations in the Google system architecture that we have glossed over for now. However, to understand the Google architecture better, it is first necessary to continue our exploration of the hardware, software, and network components that make up the Google system, as well as every other IT system. We will return for a more in-depth discussion of the Google system architecture in Supplementary Chapter 2.
When working with large concepts with defined boundaries, it is often easiest to think of them in terms of systems. A system can be defined as a collection of components, linked together and organized in such a way as to be recognizable as a single unit. The components themselves may also be recognized as subsystems, to be further reduced into components, when appropriate. The area outside the boundaries of a system is its environment. The system affects and is affected by various elements of the environment. In many situations, the environment supplies inputs to the system and receives outputs from the system. The patterns of relationships, connections, constraints, and linkages among the components of a system and between a system and its environment are known collectively as the architecture of the system.

Information technology systems are systems that support the strategy and operations of organizations. The technological components of an IT system include computer hardware, application software, operating system software, networks, and data. Other components include personnel, policies, and more.

There are a number of different ways of viewing an IT system, including application architecture, network architecture, software architecture, and hardware architecture. The general architecture for an IT system includes all of these considerations.

Nearly all modern IT systems rely on distributed processing. Data comes from many sources and information is required by users distributed throughout an organization and beyond. The most common application architecture to support distributed processing is client-server architecture, in which server computer systems provide various services—Web, database, file, print, processing—to client computer systems. Client-server systems are convenient for users and offer centralized control for the organization. Client-server architecture is commonly organized in tiers, ranging from two-tier to n-tier. The alternative architecture to client-server computing, peer-to-peer computing, is used outside of organizations as a means for sharing files over the Internet, but is of limited use in organizational settings due to difficulties in establishing stable data sources, security risks, and lack of central control. It is also possible to create a hybrid architecture, with features from both client-server and peer-to-peer computing.

A specific type of client-server architecture, Web-based computing, predominates the IT scene, primarily because users are generally familiar with the use of Web browsers, the technology is standardized and already in use in most organizations, and good development tools for designing Web pages and accessing data are readily available. Both intranets and the Internet provide user access.

Protocols are the means used to communicate between computers. IT system protocols of interest to us include network protocols such as TCP/IP, I/O protocols such as USB and PCI-Express, and application protocols such as HTTP. Standards make it possible for different system components to work together. Most modern standards are global. There are standards that are defined by interested groups and de facto standards that arise from common usage.

The first step in IT system analysis and design is about finding an appropriate architecture for a particular business situation. The task can be difficult and challenging. It is easy to see why system architects need a deep understanding of the computer system...
and network components that comprise the modern IT system to make the appropriate
design, selections, and tradeoffs.

Hopefully this short but concentrated chapter has prepared you for the remainder of
the book, which considers in detail the data, computer system hardware, operating systems,
and networks that make up the technological infrastructure of an IT system.

FOR FURTHER READING

Surprisingly, there are few books that discuss system concepts and system architecture in
a truly general way. Most books that claim to be about system architecture are actually
specific to a particular field, usually the field of information systems. One general book
about systems is by Laszlo [LASZ96]. Some IS systems design and analysis textbooks
provide a brief introduction to general system concepts. (Unfortunately, many don’t!) One example of a book that provides a good introduction to system concepts is Stampf [STAM05]. Chapter 1 of Stampf covers the topics in this chapter well. Wikipedia offers
other references under the topic system.

KEY CONCEPTS AND TERMS

<table>
<thead>
<tr>
<th>abstraction</th>
<th>interface</th>
<th>subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>application architecture</td>
<td>intranet</td>
<td>system</td>
</tr>
<tr>
<td>architecture</td>
<td>middleware</td>
<td>three-tier architecture</td>
</tr>
<tr>
<td>client-server (model)</td>
<td>n-tier architecture</td>
<td>top-down approach</td>
</tr>
<tr>
<td>decomposition</td>
<td>peer-to-peer architecture</td>
<td>two-tier architecture</td>
</tr>
<tr>
<td>environment</td>
<td>shared server</td>
<td></td>
</tr>
</tbody>
</table>

READING REVIEW QUESTIONS

2.1 What are the most important ideas, keywords, and phrases that are stated in the
definition of a system?
2.2 Explain the relationships among the following words: system, environment, boundary, interface.
2.3 Explain the following statement about systems: “Decomposition is inherently
hierarchical.”
2.4 Explain what is meant by the architecture of a system.
2.5 What does the top-down approach allow a system architect to do that might be
more difficult otherwise?
2.6 What is the primary concern of application architecture? Give an example of
application architecture, either your own, or one from the examples in the book. Explain how this example fulfills the features and requirements of the concept of application architecture.
2.7 Most modern computing in organizations is based on client-server models. Explain
why this tends to be the case. Give an example of client-server computing that you
are familiar with and explain the characteristics of your example that fulfill the concept of client-server computing.

2.8 Web-based system architecture is a popular approach to many organizational systems because it offers a number of advantages to the users and to the organization over other types of systems. Discuss the primary advantages to this approach.

2.9 What are the principal responsibilities of a system architect?

2.10 Many system architects base their IT system designs on an $n$-tier architecture, where $n$ is a number with value 2 or greater. Explain the difference between a single-tier architecture and an $n$-tier architecture. What are the main advantages claimed for an $n$-tier architecture?