21 The Operational Viewpoint

Definition	Describes how the system will be operated, administered, and supported when it is running in its production environment
Concerns	Installation and upgrade, functional migration, data migration, operational monitoring and control, configuration management, performance monitoring, support, and backup and restore
Models	Installation models, migration models, configuration management models, administration models, and support models
Problems and Pitfalls	Lack of engagement with the operational staff, lack of backout planning, lack of migration planning, insufficient migration window, missing management tools, lack of integration into the production environment, and inadequate backup models
Stakeholders	System administrators, developers, testers, communicators, and assessors
Applicability	Any system being deployed into a complex or critical operational environment

Considerable effort is spent defining the architecture and design of today's large systems. However, it is rare in our experience to find a system for which comparable consideration is given as to how the system will be controlled, managed, and monitored. The aim of the Operational viewpoint is to identify a system-wide strategy for addressing the operational concerns of the system's stakeholders and to identify solutions that address these.

For a large information system, the Operational view focuses on concerns that help ensure that the system is a reliable and effective part of the commissioning enterprise's information technology environment. For a product development project, the Operational view is more generic and illustrates the *types* of operational concerns that customers of the product are likely to encounter, rather than the concerns of a specific site. This view also identifies the solutions to be applied throughout the product implementation to resolve these concerns.

Of all the views you create for your AD, the Operational view is often the one that is least well defined and needs the most refinement and elaboration during the system's construction. This is simply because many of the details that the Operational view considers are not fully defined until design and construction is well under way. However, considering the issues described in this chapter as early as possible will save you a lot of time and effort later.

CONCERNS

Installation and Upgrade

Installation and upgrade can range from the development team installing and configuring software elements on customer-specific hardware to the ultimate users of the system obtaining hardware and software from a number of sources and performing installation, integration, and configuration themselves.

The other major area of variability is whether this is a pure installation or whether a previous version of your system is already installed, making the installation of the current version actually an upgrade. Upgrade can be significantly more complex than installation due to the need to respect existing data, configuration settings, the state of running elements, and so on. However, the use of iterative development approaches means that upgrade, rather than installation, is the norm, so you need to master it.

As an architectural concern, installation is less about the design of detailed procedures and plans and more about ensuring that the system can be installed or upgraded in a way that is acceptable to stakeholders. This involves working with technical specialists to understand the installation processes, software developers to ensure their elements can be easily and reliably installed, and system administrators to assure a practical, low-risk installation approach.

Functional Migration

Functional migration is the process of replacing existing capabilities with the ones provided by your system. This usually means migrating users of an older system to use your new system. Your migration approach may comprise one or more of the following:

- A *big bang* where the migration occurs irrevocably at a single point in time (often over a weekend)

- A *parallel run* where new and old versions of a system are used side by side until confidence in the new system is high enough to allow switching off the old one
- A *staged migration* where parts of a process or an organization are moved to a new system one by one, to manage the risk and cost of the migration activity

As an architectural concern, migration is centered on two issues—*risk* and *cost*. The big bang approach, for example, can be the cheapest because it requires no replication of resources, but it can be extremely risky because there is no easy recovery route if the migration goes wrong. Other approaches can be extremely expensive (because they require duplication of resources and the implementation of costly processes to ensure that systems run together in lockstep) but reduce risk.

Data Migration

Most if not all system developments involve some element of data migration—that is, loading data from existing systems into the new one(s). A goal of a data migration exercise is almost always to automate as much as possible, particularly where large volumes of data are involved. Where migrated data is very old, of variable quality, or poorly modeled, data migration may be extremely complex.

Data migration software is typically viewed as utility software with a limited life, rather than as a system requiring long-term support. This does not mean that it is of any lesser quality, but it may consist of a collection of automated software, semiautomated procedures, and manual intervention to deal with exceptions (such as missing data or data in unexpected formats). This also adds to the complexity of the process.

Nowadays, systems that manage hundreds of gigabytes or terabytes of data are not uncommon, and this presents its own migration challenges. Massive data stores are far more likely to include data that does not conform to business rules and therefore requires exceptional processing (possibly manual intervention). It can take days or weeks to extract data from or load data into massive data stores, and it is important that you do not underestimate the time required to reorganize databases, create indexes, and so on.

Fortunately, you can also make use of a wide range of Extraction, Transformation, and Load (ETL) tools that will help you automate this process. Many ETL tools allow you to define transformation rules visually and provide facilities for accessing a wide range of different physical formats, performing standard transformations, and monitoring and analyzing the results.

Another frequently overlooked problem occurs when you are migrating data from a live system that is continuing to be updated while you migrate from it, as discussed in this example.

EXAMPLE A government tax office has a very large database of taxpayers that it is migrating into a new system. The database is updated through end-user screens in tax offices throughout the country.

The architect predicts that extracting all of the data from the database will take between three and five days. The data must be sorted, which will take another day, and will then be loaded into the new system, which will take ten days. Finally, indexes must be created on the new system, which takes another day. The overall elapsed time to migrate is over two weeks, during which time the original system is estimated to have received 100,000 updates, as shown in Figure 21–1.

Special code has to be written to capture these updates as they occur and to apply them to the new system once the bulk of the data has been migrated into it, so that the extract is complete.

In short, data migration may be a significant piece of work in its own right, and you should manage it the same way as any other development project, with requirements, design, build and test, and acceptance—and, of course, architecture. Many of the architectural principles described in this book apply equally well to such a migration subproject, although the success criteria are different. In a migration project, it is the successfully migrated

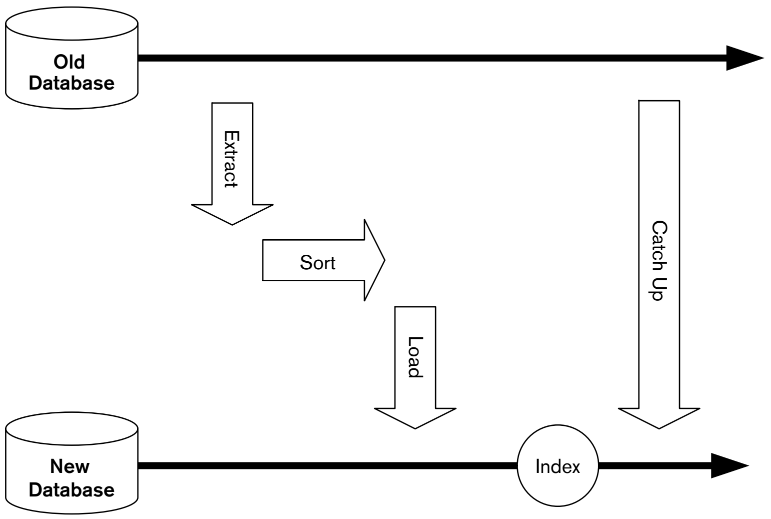


FIGURE 21–1 DATA MIGRATION FROM A LIVE SYSTEM

data that has to be accepted, rather than the software, which will be discarded once migration is complete.

Operational Monitoring and Control

Once a system is running in its production environment, it will require some amount of routine monitoring, to ensure that it is working correctly, and some routine control operations, to keep it working that way (startup, shutdown, transaction resubmission, and so on).

Some systems need little monitoring or control—for example, a file server that needs only direct operational control when it fails or fills up. Others may need quite a lot—for example, a large financial reconciliation system that accepts data feeds from a variety of sources and may need routine monitoring and control to identify and rectify communication link and data reconciliation failures.

The amount of monitoring and control needed depends on the likely number and variety of unexpected operational conditions the system is likely to encounter in production. However, the development of monitoring and control facilities can be a major effort in itself, so you may have to balance stakeholder needs in this area against cost and time. You also need to consider the system's deployment environment to make sure that the solutions you identify are appropriate.

Configuration Management

Many of the elements that make up your deployment environment will have their own configuration parameters. Databases, operating systems, middleware products, and of course your own software elements may all require detailed, specific configuration for the system to operate correctly. You may also need to make coordinated sets of changes to these configurations on a regular basis (the canonical example being a switch from online to batch mode and back again every 24 hours). Managing a number of separate element configurations can rapidly become complex enough to be a major source of operational risk for the system.

The discipline of configuration management aims to address this problem. Configuration management encompasses the processes and technologies to group, modify, and track element configuration parameters in a reliable and predictable manner.

The process of operational configuration management tends to be a fairly specialist job, handled by the system administration groups that run the production systems. From the architectural perspective, addressing this concern involves understanding the operational configuration your system requires and ensuring that it is possible to achieve it in a way that will be accepted by the interested stakeholders.

Performance Monitoring

The process of understanding and improving system performance is known as *performance engineering*, which we discuss in Chapter 25. However, the basis of all performance engineering work is *measurement*, so performance monitoring is an important concern for most systems. Your system needs to be able to capture, present, and store accurate quantitative performance information.

Production system administrators are often the first people who need to recognize and respond to a performance problem. You need to involve them in this process as early as possible to make sure they can work with the proposed solution.

In Chapter 25, we discuss in more detail the kinds of metrics needed for performance engineering and how you can capture and report them.

Support

End users, support staff, and maintainers have an interest in the type and level of support needed, who will provide that support, and the channels through which it will be delivered. As well as the system itself, support may be needed for the associated hardware infrastructure (computers, printers, and the network).

Backup and Restore

As we saw in our description of the Information viewpoint in Chapter 17, data is an extremely valuable asset to any organization and should be protected and "insured" the same way that its other assets are. Processes to do this should be carefully designed, built, and executed and should also be regularly tested to ensure they are still working correctly.

EXAMPLE One of us visited a trade organization many years ago that ran its membership database on a stand-alone UNIX system. The system administrator faithfully backed up the database to tape every night but, unfortunately, because the output of the process was not captured to a log file, nobody realized that the tape drive was broken and no data was being written. Only when the inevitable happened and a disk failed did the soon-to-be-unemployed system administrator realize that he had a shelf full of blank tapes. The organization had to recreate its membership database from paper records, which was a slow, painful, and costly process.

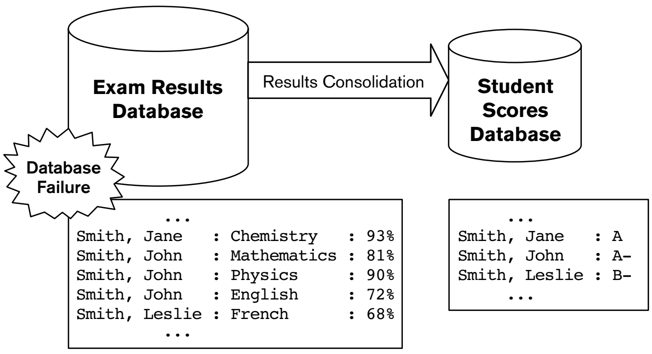
You shouldn't forget the restore side of the equation, either. As a minimum, restoring data should leave it in a transactionally consistent state (i.e., with all updates entirely committed to the restored database or not recovered at all). You will need to consider the amount of data lost as a result of the restore; at a minimum, this will be any transactions that are active at the time of failure but may include a lot more, particularly if backups can be done only when the system is offline.

If your data is distributed, this problem becomes much harder. Although a failure in any one program will affect only that program's data, the data may then become inconsistent with the rest of the system; you need to develop strategies to deal with this. Typically, the solution involves recovering or recreating the lost data, either manually or (preferably) automatically. In some cases, it may be more appropriate to revert the rest of the system to the older state.

A significant complication for backup and restore planning is the fact that, in many situations, transactional consistency must extend across a system's entire distributed data set, as shown in the next example.

EXAMPLE A university maintains academic records for all its students in a number of databases. The main database stores results for each exam taken by each student, while a consolidated database turns these into an overall score for each student based on exam success, as shown in Figure 21–2.

A database corruption means that the Exam Results database has to be restored from its latest clean backup, which is almost three months old, and the results for the last three months rekeyed into it. Although the Student Scores database is unaffected by the corruption, special actions must be taken to prevent this manually recovered data from filtering into the Student Scores database and corrupting its data. As a result, it could take several weeks to repair the damage due to the corruption of one database.



```
         ...
Smith, Jane   : Chemistry   : 93%
Smith, John   : Mathematics : 81%
Smith, John   : Physics     : 90%
Smith, John   : English     : 72%
Smith, Leslie : French      : 68%
         ...
```

```
      ...
Smith, Jane   : A
Smith, John   : A-
Smith, Leslie : B-
      ...
```

FIGURE 21–2 EXAMPLE OF BACKUP AND RECOVERY

TABLE 21–1 STAKEHOLDER CONCERNS FOR THE OPERATIONAL VIEWPOINT

Stakeholder Class	Concerns
Assessors	Functional migration, data migration, and support
Communicators	Installation and upgrade, functional migration, and operational monitoring and control
Developers	Operational monitoring and control and performance monitoring
Support staff	Functional migration, data migration, and support
System administrators	All concerns
Testers	Installation and upgrade, functional migration, data migration, monitoring and control, and performance monitoring
Users	Support

If your system contains data distributed over a number of data stores, you must ensure that your Operational view takes this into account when considering backup and recovery.

Stakeholder Concerns

Typical stakeholder concerns for the Operational viewpoint include those listed in Table 21–1.

MODELS

The Operational view consists of models that illustrate how the system will be put into production and kept running effectively once it is there.

Bear in mind that for most enterprise systems, each of these models can be quite large and involved. When this is the case, it's sensible to summarize the model in the AD and reference a fuller model in another document, to avoid making the AD too large and unwieldy.

Installation Models

Moving a system from its development environment to its production environment is a critical part of the system's lifecycle. Your AD needs to demonstrate that it is possible for a system built using this architecture to be installed (and upgraded) in a practical way.

The installation model should discuss installation and/or upgrade as needed for your system. This model needs to help the reader understand:

- What needs to be installed or upgraded to move the system into production
- What dependencies exist between the various groups of items to be installed and upgraded
- What constraints exist on the process to perform the installation and/or upgrade for the system
- What would need to be done to abandon and undo the installation and/or upgrade if something goes seriously wrong

The AD doesn't need to include a complete installation and upgrade plan—that information goes into a different document produced later in the development lifecycle. Instead, the installation model provides your view of the requirements and constraints the architecture imposes on installation and upgrade. The installation model in your initial AD is likely to contain only an overview of your installation strategy (because the details of what needs to be installed aren't fully known at that time), but you will be able to elaborate and refine this model as construction of the system progresses.

Notation The best notation to use for an installation model really depends on the situation and what the primary stakeholders (the system administrators) are familiar with. In our experience, an approach using text and tables is often the best way to communicate this information.

Simple lists work well for laying out and defining the elements of the installation problem. In simple cases, cross-reference tables can describe dependencies, while more involved dependencies are usually effectively addressed with the use of dependency diagrams.

Activities

Identify the Installation Groups. Start by considering what elements of your architecture need to be installed and/or upgraded, and identify groups of them that can be handled together. For each group, define which elements it contains and the approach that will be used to install or upgrade that group.

Identify Any Dependencies. Technical dependencies often exist between different parts of a complex system during installation, so that the installation process has to proceed in a specific order. Identify the dependencies that exist between your installation groups to reveal these constraints.

Identify Any Constraints. Consider the overall installation process and the different ways it could be achieved. Other than the ordering dependencies you considered in the previous activity, does your architecture impose any further constraints on the process? (For example, do you need to start one element after it is installed so it can generate code or data needed to install the next one? Do you need to restart one of the machines during installation?)

Design the Backout Approach. Consider what would need to be done to undo any of the installation tasks you have identified. In particular, identify anything that would be complex or time-consuming to undo.

EXAMPLE This example shows an installation model for a rental-tracking system, based on the results of the activities just described.

Installation Groups

- *Win32 Desktop Client*: Contains all of the software in the `W32CLIENT` component. Installation shall be via InstallShield automatic installer, remotely executed via management tool.
- *Database Schema*: Contains all DBMS schema definitions and data abstraction stored procedures. To be packaged as simple SQL scripts and installed using a custom written Perl script.
- *Web Interface*: Contains the server-resident user interface components (the `WEBINTERFACE` component). Installation will be by manual administrative action, copying files into IIS directories according to written instructions.
- *Rental-Tracking Service*: Contains the .NET assemblies that implement the services called by the Web and Win32 interfaces (the `RENTALTRACKER` component). Installation will be by manual administrative action, copying files into IIS directories according to written instructions.
- *Reporting Engine*: Contains the .NET assemblies that implement the summary reporting engine. Installation will be by manual administrative action, copying files into IIS directories according to written instructions.

Dependencies

- Win32 Desktop Client, Web Interface, Rental-Tracking Service, and Reporting Engine depend on Database Schema.
- Win32 Desktop Client and Web Interface depend on Rental-Tracking Service.
- Web Interface depends on Reporting Engine.

Constraints

- *Win32 Desktop Client*: A restart of the client machine will be required during this installation process.

Backout Strategy

This is the first release of the software, so backout is reasonably straightforward and consists simply of uninstallation. For each installation group, the following action will be required.

- *Win32 Desktop Client*: Run the installer with an `uninstall` flag.
- *Database Schema*: A custom Perl script will be supplied to remove all objects created during the installation.
- *Web Interface*: Manual administrative action will be required. The supplied instructions will list the files to be removed.
- *Rental-Tracking Service*: Manual administrative action will be required. The supplied instructions will list the files to be removed.
- *Reporting Engine*: Manual administrative action will be required. The supplied instructions will list the files to be removed.

Migration Models

If a migration process is required, the migration model needs to illustrate the strategy that will be used. Again, a complete plan is not called for in the AD, but rather a succinct definition of the strategies to be employed. This model should allow its reader to understand:

- What overall strategies can (or will) be employed to migrate information and users to the system
- How the new system will be populated with information from the existing environment
- How information in the new and old environments will be kept synchronized (if required)
- How operation could revert to the old system if serious problems emerge with the new one

As with the installation model, the migration model should focus on the requirements and constraints that the current architecture places on the detailed migration process that will be developed later.

NOTATION A migration model is usually documented by using text and tables because no suitable, widely accepted graphical notations are available. Some informal diagrams may help illustrate data migration and synchronization, and particularly complex data migration may require some form of data model to illustrate the transformations involved.

ACTIVITIES

Establish Possible Strategies. Assess your architecture and the existing system(s), and establish which migration strategies (i.e., big bang, parallel run, staged migration) are possible, how each would work, and the tradeoffs among them.

Define the Primary Strategy. In some situations, you will simply define the options and someone else will decide which approach to use (e.g., if you're developing a product and different customers want to migrate different ways). In other cases, you will be tasked with the responsibility of defining which strategy best meets the needs of your stakeholders and making it happen.

Design the Data Migration Approach. Having identified a strategy to use, you need to decide how to populate the system with all of the information in the existing system(s). This doesn't mean you need to spend days mapping fields between databases, but it does mean you have to understand the problem well enough to choose an appropriate approach for the data migration and determine how long it is likely to take and what tasks and resources might be required.

Design the Information Synchronization Approach. Some situations call for information to be synchronized between the old system(s) and the new one. This is particularly the case when using the parallel run migration strategy because information in the old system(s) may continue to be updated after the new system goes live. If synchronization is required, it may be unidirectional (just into the new system) or bidirectional (information changes need to be migrated from new to old as well as old to new). Your task is to identify an overall approach that will allow the required degree of synchronization to be performed within the operational constraints of your environment.

Identify the Backout Strategy. Being able to back out to an existing system (if available) is an attractive risk-reduction option for live operation. The problem is that it isn't always clear how such a backout would work, or if it is even possible (e.g., reverse data migration may not be practical due to the design of the new system). You need to decide whether a backout strategy involving the old system(s) is required and, if so, how it could work.

Configuration Management Models

You may need to create a configuration management model if your system requires complex, regular reconfiguration (e.g., reconfiguring parts of the system to handle different types of workloads according to a calendar-based schedule). This model must explain to its reader:

- The groups of configuration items in the system and how each is managed
- The dependencies that exist among the configuration groups

- The different configuration value sets required for routine system operation (and why each is required)
- How the different sets of configuration values will be applied to the system

The aim is to create a model of the system configuration management problem (rather than identifying lots of individual configuration values). This allows those responsible for system configuration management to understand the problem and plan their solution for it.

Like the installation model, this model is unlikely to be complete in your early AD but can be elaborated and refined as construction of the system progresses and the details of the configuration items required become known.

NOTATION This model is often quite simple and best documented by using text and tables. In more complex cases, it is usually best treated primarily as a data model, and a data-modeling notation, such as entity-relationship diagrams or UML, is a useful addition to the textual description.

ACTIVITIES

Identify the Configuration Groups. Consider all of the configuration values your system requires and break them into cohesive groups with as few intergroup dependencies as possible. This allows you to abstract the problem of managing the individual values to the level of managing large groups as a single unit (think of them as clumps of values). Name each group, explain its purpose, and explain how the configuration group would be managed (how values are defined, collected, applied, and so on).

Identify Any Configuration Group Dependencies. Having identified the groups of configuration information in your system, you can clearly identify and record any dependencies among them. For example, if changing the configuration of your database management system means reconfiguring the operating system, or adding more instances of one of your elements means changing the application server configuration, record these as intergroup dependencies. Identifying these dependencies allows you to start understanding the problem of reconfiguring your system in production.

Identify Configuration Value Sets. Consider your system during its routine operational lifecycle, and establish how many configurations your system will need during it. Define the characteristics of each value set, and identify the configuration groups that change between different configurations. For each set you identify, define its purpose and when it needs to be applied. This allows assessment of the operational impact of your architecture's configuration needs.

Design the Configuration Change Strategy. Once you have identified the configuration your system needs and the changes you need to make to it, design an approach to achieve this in your intended production environment.

Again, rather than focusing on the minutiae of the administration process, you need to identify a practical overall approach that the production administrators of your system will accept.

EXAMPLE This example shows a configuration management model for the rental-tracking system we described earlier.

Configuration Groups

- *DBMS Parameters*: The SQL Server 2000 parameters that control the initialization, operation, and performance characteristics of the database. These are managed via SQL scripts, applied by database administrators.
- *IIS Parameters*: The IIS parameters that control the initialization, operation, and performance characteristics of the server. These are managed by using a set of JScript scripts that will be supplied with the system.
- *Reporting Engine Options*: The Reporting Engine parameters that control what is summarized and when summaries are computed. These are managed as a set of configuration files read by the component.

Configuration Dependencies

- When the IIS Parameters are set to allow more connections, the DBMS Parameters must be changed to allow for the possible increase in load.
- If the Reporting Engine Options are set for more aggressive summary activity, the DBMS Parameters must be set to allow for an increased amount of data cache being required.

Configuration Sets

- *Standard*: Normal configuration for planned system workload of up to 1,200 concurrent users with the Reporting Engine producing level 1 summary statistics every 6 hours.
- *High Volume*: Configuration to be applied when high client volume is expected. Increases capacity to 2,000 concurrent users and switches off routine operation of the Reporting Engine.
- *Month End*: Configuration to be applied during the last 2 days of the month, limiting concurrent usage to 800 users and allowing the Reporting Engine to run continually to produce complete summary statistics.

Configuration Change Strategy

The configuration sets will be applied as follows.

- The DBMS will be manually reconfigured first, by the database administrator running a single script that sets the parameters for the desired configuration set. (This could involve a DBMS restart.)
- Next, the Reporting Engine Options will be changed by altering the Engine's configuration file parameter and restarting it.
- Finally, the IIS configuration set will be applied by an administrator running the appropriate JScript script and restarting the IIS engine.

Administration Models

When your system is running in its production environment, it will require some degree of administration to monitor it and keep it running smoothly. The administration model is the section of your AD where you define the operational requirements and constraints of your architecture and the facilities it provides for administrative users.

The administration model must define the following items.

- *Monitoring and control facilities*: In order to support your system's administrators, you may need to provide some monitoring and control facilities as part of your architecture. This may involve custom utilities and features and/or integration into an existing management environment. It can be as simple as a basic message log or as complex as a full-blown integration with a management or monitoring infrastructure. You need to clearly define the facilities you are going to provide, how these address the problem, and whether any limitations in these facilities could constrain their applicability or usefulness.
- *Required routine procedures*: You should also review the architecture you have designed and identify any administrative work that needs to be performed on a regular basis or that may be required in exceptional circumstances. Depending on the system, this can be as basic as a weekly backup and a monthly health check, or it can be a set of complex procedures performed on a round-the-clock basis to keep a critical high-volume system running at peak efficiency. For each procedure, you need to define its purpose, when it is performed, who performs it, and what is involved in performing it. In most cases, you should cross-refer to the relevant monitoring and control facilities provided.
- *Likely error conditions*: Any complex system can suffer unexpected failures due to internal or external faults. From simple situations such as

disks filling up to sudden failures of the underlying network causing a cascade of problems, many error conditions that occur can require administrative intervention to rectify them. Some of these conditions are independent of your architecture and are caused by underlying platform failures. The administrators of your system will probably already be experts at diagnosing and recovering from these failures. However, you cannot expect them to understand the possible error conditions that are unique to your architecture, and you need to explain these carefully to help administrators understand the conditions they may need to recover from. Your description should include when the condition can occur, how to recognize it (referencing relevant monitoring facilities provided), how to rectify it (referencing relevant control facilities provided), and possible further failures the condition could trigger.

- *Performance monitoring facilities*: A specialist subset of system monitoring is the ability to monitor the performance of the system. The difference between operational monitoring and performance monitoring tends to be how the data is used. Operational monitoring usually reports by exception and produces little or no output data when everything is going well. In contrast, performance monitoring facilities are usually designed so that performance information can be extracted and analyzed routinely to allow system performance to be tracked over time. We talk more about performance activities in general in Chapter 25. In an administration model, you need to explain the types of performance measures you will make available and how administrators or developers will extract and analyze the information when required.

An important point to note is the strong degree of cross-reference between the administrative facilities you define in this model and the common design model in the Development view. In the Operational view, you define the facilities you will provide for the administrative stakeholders. The Development view needs to define the common processing required across all of the system's elements in order to actually achieve those facilities.

Notation The primary customers of the administration model are system administrators, who may not be software developers by training. We have found that the right notation for this model is nearly always text and tables augmented with a few informal diagrams where needed. Extensive use of more formal notation such as UML is less appropriate for this model.

Activities

Identify the Routine Maintenance Required. Consider your system running in its production environment and create a list of the types of operational tasks that will need to be performed to keep the system running smoothly. For each task type, define who needs to perform it, when it needs to be performed. and how it should be performed.

Identify Your Likely Error Conditions. Analyze your architecture by considering its primary usage scenarios, and work out what is likely to go wrong during the operational lifecycle (elements failing, data stores filling up, systems running out of memory or other runtime resources). Make sure you think about the ones related to administration and maintenance as well as the ones that end users would be aware of—it is often harder to plan for failures during larger-scale administrative scenarios (such as data maintenance). Identify the classes of error conditions that can occur, what causes them, and how they can be rectified to get the system running again. You should also try to estimate the likely availability impact of the failure to ensure that you can recover the system in a time acceptable to your stakeholders. You may want to consider the error conditions that could occur if routine maintenance *isn't* performed, so that the importance of this maintenance is understood.

Specify Any Custom Utilities. Routine and exceptional procedures may require system-specific utilities to allow administrators to perform them efficiently. Such utilities can range from very simple database or operating system scripts to significant pieces of software in their own right. Consider whether any such utilities are required, and specify any you need.

Identify the Key Performance Scenarios. Some of your architectural usage scenarios will be much more important than others from a performance perspective (look for the scenarios that support time-critical business processes, involve a high workload, are executed very frequently, or are required by key stakeholders). Extract these scenarios from the overall system usage scenarios.

Identify the Performance Metrics. Consider the key performance scenarios and identify metrics that will allow you to measure the performance achieved for each and to analyze where the system spends most of its time and resources. In order to abstract the problem, it may be more useful to identify classes of metrics rather than individual ones. Make sure that you record what each metric or class identified actually means and what it is used for.

Design the Monitoring Facilities. Having established the operational tasks and performance metrics required, you can design monitoring facilities to be used across the system to provide routine system monitoring and error condition recognition and to gather performance metrics from the system's elements. This design will be at the outline level, to be fleshed out later during the development increments of the lifecycle. However, at this stage you should provide enough detail to clarify what needs to be done in each system element to provide the administration facilities required.

EXAMPLE This example shows an administration model for the rental-tracking system.

Monitoring and Control

The monitoring and control facilities are as follows.

- *Server Message Logging*: All server components will write information, warning, and error messages to the Windows Event Log of the machine they are running on.
- *Client Message Logging*: The client software will log messages if an unexpected error is encountered. The log will be written to the hard disk of the client machine for later manual retrieval.
- *Startup and Shutdown*: No system-specific startup and shutdown facilities will be provided because the software will run in the context of the IIS and SQL Server 2000 servers, and their facilities are considered to be sufficient.

Operational Procedures

Routine operational procedures are as follows.

- *Backup*: Operational data in the SQL Server database will need to be backed up. This will involve backing up the transaction logs every 15 minutes and backing up the application's databases every day. Details of this procedure will be left to the database administrators.
- *Pruning of Summary Information*: The Reporting Engine does not remove the summary reporting information it creates. This information is left in place and is available to users of the Win32 client interface. Database administrators will need to monitor the performance of the Reporting Engine and the management reporting aspects of the Win32 client components and manually prune the summary information when its volume starts to impact performance. A written procedure will be supplied to explain how the pruning should be performed.

Error Conditions

The error conditions that administrators should be expected to handle are as follows.

- *Database Out of Log Space*: If transaction volume rises above a certain point, it is possible that the transaction log will fill. This will cause the system to suspend operation. Database administrators will need to recognize log space problems and manually back up the logs to free space. If this happens routinely, the backup interval for the transaction logs should be reduced.

- *Database out of Data Space*: If the database runs out of data space, the system will stop operating. Again, database administrators will need to recognize this condition and either prune the summary information (see above) or add more data space to the system. A written estimate of the amount of space required for various volumes of workload will be provided.
- *IIS Failure*: If the IIS server fails, the system will completely fail, and Win32 clients will lose contact with the server. Administrators need to recognize this condition and restart IIS. The system will recover automatically once IIS is restarted. The Win32 clients will automatically reconnect once the server is available again.

Performance Monitoring

No application-specific performance monitoring facilities are planned. System performance monitoring should be achieved by using the following facilities.

- *SQL Server 2000 Counters*: The SQL Server 2000 product allows a wide range of performance counters to be collected via the Windows 2000 Server's Performance Monitor application. These counters should be used to assess the volume of workload on the database and the time taken for the application's transactions to complete.
- *IIS/ASP.NET Counters*: IIS Server and ASP.NET produce a wide range of performance counters to be collected via the Windows 2000 Server's Performance Monitor application. These counters should be used to assess the number of Web requests being serviced and how long it is taking to service them.
- *.NET Counters*: The .NET runtime allows a wide range of performance counters to be collected via the Windows 2000 Server's Performance Monitor application. These counters should be used to establish the amount of non-Web-request workload that the application is performing and how long it is taking to perform the operations.

Support Models

Once your system is running in production, at least some of the system's stakeholders are likely to need help using or operating it, and other parties will need to provide assistance to them. The support model should provide a clear abstraction of the support that will be provided, who will provide the support, and how problems can be escalated between parties when searching for a resolution. This means defining the following in your support model.

- *Groups needing support*: The model must clearly define the groups of stakeholders that will require support, the nature of support they need, and the appropriate mechanisms for delivering that support.
- *Classes of incidents*: The model must also define what sort of support incidents are likely to be encountered and what sort of response is reasonable to expect in each case. The definition of each class of incident should clearly state the characteristics of an incident in that class, typically in terms of operational, organizational, or financial impacts.
- *Support providers and responsibilities*: Each type of support incident needs to be handled by at least one support provider, who must accept responsibility for resolving the incident. The model should capture who the support providers are and their responsibilities for incident resolution.
- *Escalation process*: A serious incident often requires a number of different support providers to resolve the situation because it is too complex or specialized for a single provider to handle. Your model should define how incidents are escalated between support providers and the responsibilities of each when this happens. This will help ensure that incident resolution does not stall because of confusion over responsibilities or a lack of expertise by a particular provider.

As with the other models for the Operational view, the focus of the support model should be to provide an overview of the support problem and a strategy for its solution rather than the definition of detailed procedures.

NOTATION This model needs to be understood by a number of different technical and nontechnical stakeholder groups. The majority of the model should normally be a text-and-tables definition of the support to be provided, with some flow diagrams (such as UML activity diagrams) where required to make the information flow and decision-making processes clear.

ACTIVITIES

Identify the Supported Groups. Identify the groups of stakeholders who will need support, the type of support they will need, and the possible avenues through which that support could be provided.

Identify the Support Providers. Decide who will be providing support to your stakeholders. For each provider (which will probably be an organization), define the support they provide and how they provide it.

Identify Any Incidents Requiring Support. Consider the types of incidents that could trigger the need for assistance by each of your groups of supported stakeholders, and characterize the incident type by likely frequency and severity.

Map the Providers, Incidents, and Groups. Decide which support providers will resolve which incident types for which stakeholder groups, and ensure that each provider can offer suitable support.

Plan the Escalation. Consider your groups of support providers, and identify which of them may need to escalate problems to other support providers. Define the escalation paths that should be used between providers and the responsibilities of each provider when this happens.

EXAMPLE This example shows a support model for the rental-tracking system.

Supported Groups

- *Web Users*: People using the Web interface to book or manage their rentals may need support if there are problems with the site or if they have difficulties using the Web interface. Few assumptions may be made about this group, and the primary support channel should be e-mail, with telephone backup.
- *Win32 Users*: Internal users using the Win32 client may need help with a range of problems including usage issues, system problems, and PC support. Their primary support channel is assumed to be telephone, although they may be prepared to receive support via e-mail.
- *Windows Administrators*: The administrators of the server machines are technically sophisticated and will require assistance only in unexpected failure scenarios. They will need to receive immediate assistance via telephone as well as query resolution via e-mail.
- *Database Administrators*: The database administrators are technically sophisticated and will require assistance only with unfamiliar database behavior. They will need to receive immediate assistance via telephone as well as query resolution via e-mail.

Support Providers

- *Web Services Help Desk*: This organizational group is responsible for resolving all support incidents raised by users of the Web interface. They provide support via e-mail and telephone, 6 days per week, 20 hours per day.
- *IT Help Desk*: This organizational group is responsible for resolving all support incidents raised by users of the Win32 client interface. They provide support via e-mail and telephone and can often provide direct assistance at the end user's desk as well. Support is provided during normal business hours.
- *DBA Group*: This organizational group is responsible for resolving all support incidents related to database management systems. They provide support via e-mail and telephone. Support is normally provided during normal business hours, with the option of using on-call staff outside this period.
- *Windows Administrators*: This organizational group is responsible for resolving all support incidents related to IIS, .NET, and Windows 2000 Server and underlying hardware. They provide support via e-mail and telephone. Support is provided during normal business hours, with the option of using on-call staff outside this period.
- *Microsoft Support*: This is an external organization (the Microsoft Corporation's Support division) that is responsible for assisting with the resolution of support incidents caused by a fault or usage problem with the SQL Server 2000, Windows 2000 Server, or IIS products. They provide support via e-mail, newsgroups, Web sites, fax, and telephone. Support is provided 24 hours per day, every day.
- *Development Team*: This is the organizational group that developed the system originally and maintains it on an ongoing basis. They are responsible for resolving any incident that other support providers cannot resolve. They provide support via e-mail, telephone, and site visits during normal business hours, with the ability to reach on-call staff during other times.

Support Incidents and Resolution

- *Web Usage Difficulties*: This class of support incident covers any situation where a user of the Web interface is having problems using the system that are not caused by failure or malfunction of a system component. These incidents should be resolved in a single interaction with the Web Services Help Desk, either by phone or e-mail. The impact on the organization should be minimal.

- Win*32 Usage Diffi*culties: This class of support incident covers any situation where a user of the Win32 client interface is having problems using the system that are not caused by failure or malfunction of a system component. These incidents should be resolved in a single interaction with the IT Help Desk, either by phone or e-mail. The impact on the organization should be minimal.
- *End-User System Errors*: This class of support incident covers any situation where a user of the system encounters a problem caused by failure or malfunction of a system component. These incidents should be resolved within 1 working day. The user should interact entirely with staff of the IT or Web Services Help Desk, who will manage problem resolution and deal with other support providers as required. The impact on the organization should be moderate and should not threaten business operations beyond inconvenience.
- *Slow End-User Performance*: This class of support incident covers any situation where end users complain of unacceptably slow performance. These incidents should be resolved within 3 working days. The user should interact entirely with members of the IT or Web Services Help Desk, who will manage problem resolution and deal with other support providers as required. The impact on the organization should be moderate and should not threaten business operations beyond inconvenience.
- *Database Corruption*: This class of support incident covers any situation where the database system reports internal corruption. These incidents should be resolved within 2 hours (although, realistically it is recognized that they could return; however, the original incident should be resolved within 2 hours). The DBA Group is responsible for recognizing and resolving these situations. The impact on the organization should be moderate, but business operations will be interrupted while the problem is resolved.
- *Database Failure*: This class of support incident covers any situation where the database system needs to be recovered from backups. These incidents should be resolved within 4 hours. The DBA Group is responsible for recognizing and resolving these situations. The impact on the organization may be severe during this period, with business operations being interrupted for the whole period of the incident, but should not continue beyond the resolution of the incident.
- *IIS or Windows 2000 Server Failure*: This class of support incident covers any situation where the IIS Server, underlying operating

system, or underlying hardware suffers a failure. These incidents should be resolved within 1 hour. The Windows Administrators are responsible for recognizing and resolving these situations. The impact on the organization may be severe during this period, with business operations being interrupted for the whole period of the incident, but should not continue beyond the resolution of the incident.

Escalation

The escalation process is as follows.

- Users of the Web interface will report problems to the Web Services Help Desk.
- Users of the Win32 client interface will report problems to the IT Help Desk.
- The Help Desks will report system problems to the Windows Administrators.
- The Windows Administrators will report database problems to the DBA Group.
- The Windows Administrators will report other problems to the Development Team.
- The Windows Administrators, DBA Group, and Development Team will report problems with Microsoft software to the Microsoft Support organization.

In each case, the organization accepting the incident must provide the reporter with a unique identifier for the incident and record the reporter's description of it. If the problem is not immediately resolved, the organization accepting the incident must provide the reporter with information on resolution status within 75% of the target resolution time.

PROBLEMS AND PITFALLS

Lack of Engagement with the Operational Staff

In many organizations, a gulf exists between the development staff members who build systems and the operational staff members who deploy and administer them. This can be a significant problem if you want to achieve a smooth, incident-free system rollout.

RISK REDUCTION

- The best solution to this problem is to engage early with the operational groups, stressing how valuable their contribution is. Operational staff often have a legitimate grievance with software developers because systems are frequently passed on to them with very little thought given to operational requirements.
- Use an explicit Operational view to help avoid this situation.

Lack of Backout Planning

Many systems we've seen don't have real backout plans. In fact, many commercial software products don't have graceful recovery mechanisms to cope with situations like failed upgrades. Without a good backout plan, you are relying on a perfect rollout for your entire system, which experience suggests is somewhat optimistic.

RISK REDUCTION

- Make sure that your system can be backed out of its production environment by defining a clear procedure and reviewing it.

Lack of Migration Planning

Many information systems replace a manual system, a previous automated system, or an earlier version of themselves, but many systems are developed without a good migration plan. Migration planning isn't glamorous or, in many cases, even interesting, but without it, you are unlikely to achieve a smooth system deployment.

RISK REDUCTION

- Make sure you understand the migration needs of your architecture as early as possible.
- Address migration needs in your AD.

Insufficient Migration Window

In our experience, data migration *always* takes longer than anticipated, typically because the data does not conform to the level of quality and consistency expected of it and because of the problems associated with handling and manipulating large volumes of data.

RISK REDUCTION

- Consider how you will deal with data errors and inconsistencies.
- Develop processes for accepting migrated data, and make sure your stakeholders have bought into them.
- In your hardware sizing models, factor in the storage requirements for transitional data.
- Include adequate elapsed-time contingency in your migration plan.
- Factor in the time needed to reorganize databases, create indexes, and so on.
- Where you are migrating data from live systems and the migration time is substantial, create strategies for reconciling any data updates made during the migration period.

Missing Management Tools

Most software developers (and, in fact, many architects) are very focused on the business of building new software. However, successful software spends most of its life in production, not development. This mismatch between focus and lifecycle often manifests itself as a lack of operational facilities, which can result in a system that is difficult to monitor and control. Software developers can monitor or control the system by using primitive tools (operating system commands or simple scripts) because of their detailed knowledge of its internal workings. Operational staff often don't have this knowledge and need more sophisticated tools to automate the required operational procedures. Without such tools, the system is unlikely to be managed well.

RISK REDUCTION

- Understand the needs of your administration stakeholders as early as possible and involve them in the development of the Operational view.
- Ensure that administrators' needs are addressed with standard, system-wide facilities.

Lack of Integration into the Production Environment

Most information systems are deployed into some sort of existing production environment, even if it is a simple or informal one. Unfortunately, it's common to find that a new system doesn't work with the environment. This can be quite a problem for operational staff who need to learn new interfaces or tools or even totally new ways to manage the system.

RISK REDUCTION

- Make sure that you understand the existing environment and its integration needs early in your system design.
- Involve experts who understand the target production environment as early as possible, and get their advice on how it works and the type and level of integration needed.

Inadequate Backup Models

Backup and restore processes can fail quite spectacularly, and you don't want to find out about problems in your model when you are desperately trying to recover important data.

RISK REDUCTION

- Do not be tempted to skimp on this area or omit it from consideration entirely.
- Incorporate backup and restore as a central part of your architecture rather than trying to add it afterward.
- Make sure that your backup scheme includes all the information you need for data recovery.
- Estimate how long backup and recovery will take, and perform some practical testing under realistic conditions.
- Make sure that your model describes how data will be restored as well as backed up.
- Consider how to maintain data consistency across multiple data stores when you have to recover one of them to an earlier state.
- Consider a "belt-and-braces" approach to backup. For example, many end-user systems—especially older, mainframe-based ones—write copies of updates received from clients to an audit and recovery area as well as to the main database. This means that if the database is damaged, it is possible to replay these transactions into the database in order to get it in synch again.

CHECKLIST

- Do you know what it takes to install your system?
- Do you have a plan for backing out a failed installation?
- Can you upgrade an existing version of the system (if required)?

- Do you know how information will be moved from the existing environment into the new system?
- Do you have a clear migration strategy to move workload to the new system? Can you reverse the migration if you need to? How will you deal with data synchronization (if required)?
- Do you know how the system will be backed up? Are you confident that the approach identified will allow reliable system restoration in an acceptable time period?
- Are the administrators confident that they can monitor and control the system in production?
- Do the administrators have a clear understanding of the procedures they need to perform for the system?
- How will performance metrics be captured for the system's elements?
- Can you manage the configuration of all of the system's elements?
- Do you know how support will be provided for the system? Is the support provided suitable for the stakeholders it is being provided for?
- Have you cross-referenced the requirements of the administration model back to the Development view to ensure that they will be implemented consistently?
- Is the data migration architecture compatible with the amount of time available to perform the data migration? Are there catch-up mechanisms in place where the source data is volatile during the data migration?

FURTHER READING

Little or no existing literature deals with the operational aspects of a system from the perspective of a software architect. Although there are many books on installing and managing specific pieces of technology, we have found very few books that examine the principles that underpin reliable production systems operation.

We are aware of some books that at least partially address this area [GOOD99; VOSS89; WYZA99]. Also, Dyson and Longshaw [DYSO04] includes a number of patterns useful in the Operational view.