28 Application Architectures

Objectives

The objective of this chapter is to introduce architectural models for specific classes of application software system. When you have read this chapter, you will:

- be aware of two fundamental architectural organisations of business systems, namely batch and transaction processing;
- understand the abstract architecture of information and resource management systems;
- understand how command-driven systems, such as editors, can be structured as event processing systems;
- know the structure and organisation of language processing systems.

Contents

As I explained in Chapter 6, you can look at system architectures from a range of perspectives. So far, the discussions of system architectures in Chapters 6 and 18 have concentrated on architectural perspectives and issues such as control, distribution and system structuring. In this chapter, however, I take an alternative approach and look at architectures from an application perspective.

Application systems are intended to meet some business or organisational need. All businesses have much in common—they need to hire people, issue invoices, keep accounts and so forth—and this is especially true of businesses operating in the same sector. Therefore, as well as general business functions, all phone companies need systems to connect calls, manage their network, issue bills to customers etc. Consequently, the application systems that these businesses use also have much in common.

Usually, systems of the same type have similar architectures and the differences between these systems are in the detailed functionality that is provided. This can be illustrated by the growth of Enterprise Resource Planning (ERP) systems such as the SAP/R3 system (Appelrath and Ritter, 2000) and vertical software packages for particular applications. In these systems, which I discuss briefly in Chapter 16, a generic system is configured and adapted to create a specific business application. For example, a system for supply chain management can be adapted for different types of suppliers, goods and contractual arrangements.

In the discussion of application architectures here, I present generic structural models of several types of application. I discuss the basic organisation of these application types and, where appropriate, break down the high-level architecture to show sub-systems that are normally included in applications.

As a software designer, you can use these generic application architectures in a number of ways:

1. *As a starting point for the architectural design process* If you are unfamiliar with this type of application, you can base your initial designs on the generic architectures. Of course, these will have to be specialised for specific systems but they are a good starting point for your design.

1. *As a design checklist* If you have developed a system architectural design, you can check this against the generic application architecture to see whether you have missed any important design components.

2. *As a way of organising the work of the development team* The application architectures identify stable structural features of the system architectures and, in many cases, it is possible to develop these in parallel. You can assign work to group members to implement different sub-systems within the architecture.

3. *As a means of assessing components for reuse* If you have components you might be able to reuse, you can compare these with the generic structures to see whether reuse is likely in the application that you are developing.

4. *As a vocabulary for talking about types of applications* If you are discussing a specific application or trying to compare applications of the same types, then you can use the concepts identified in the generic architecture to talk about the applications.

There are many types of application system and, on the surface, they may seem to be very different. However, when you examine the architectural organisation of applications, many of these superficially dissimilar applications have much in common.

I illustrate this here by describing the architectures of four broad types of application:

1. *Data processing applications* Data processing applications are applications that are data-driven. They process data in batches without explicit user interventions during the processing. The specific actions taken by the application depend on the data that it is processing. Batch processing systems are commonly used in business applications where similar operations are carried out on a large amount of data. They handle a wide range of administrative functions such as payroll, billing, accounting, and publicity.

2. *Transaction processing applications* Transaction processing applications are database-centred applications that process user requests for information and that update the information in a database. These are the most common type of interactive business systems. They are organised in such a way that user actions can't interfere with each other and the integrity of the database is maintained. This class of system includes interactive banking systems, e-commerce systems, information systems and booking systems.

3. *Event processing systems* This is a very large class of application where the actions of the system depend on interpreting events in the system's environment. These events might be the input of a command by a system user or a change in variables that are monitored by the system. Many PC-based applications, including games, editing systems such as word processors, spreadsheets, image editors and presentation systems are event processing systems. Real-time systems, discussed in Chapter 20, also fall into this category.

4. *Language processing systems* Language processing systems are systems where the user's intentions are expressed in a formal language (such as Java). The language processing system processes this language into some internal format and then interprets this internal representation. The best-known language processing systems are compilers, which translate high-level language programs to machine code. However, language processing systems are also used to interpret command languages for databases and information systems and markup languages such as XML (Harold and Means, 2002), which is extensively used to describe structured data items.

I have chosen these particular types of system because they represent the majority of systems in use today. Business systems are generally either data processing or transaction processing systems, and most personal computer software is built around an event processing architecture. Real-time systems are also event processing systems although I do not cover these architectures in this chapter but in Chapter 20. All software development relies on language processing systems such as compilers.

Batch processing systems and transaction processing systems are both database centric systems. Because of the central importance of data, it is common for applications of different types to share the same database. For example, a business data processing system that prints bank statements uses the same customer account database as a transaction processing system that provides web-based access to account information.

Of course, as I discussed in Chapter 6, complex applications rarely follow a single, simple architectural model. Rather, their architecture is more often a hybrid, with different parts of the application structured in different ways. When designing these systems, you therefore have to consider the architectures of individual sub-systems as well as the how these are integrated within an overall system architecture.

28.1 Data processing systems

Businesses rely on data processing systems to support many aspects of their business such as paying salaries, calculating and printing invoices, maintaining accounts and issuing renewals for insurance policies. As the name implies, these systems focus on data and the databases that they rely on are usually orders of magnitude larger than the systems themselves. Data processing systems are batch processing systems where data is input and output in batches from a file or database rather than input from and output to a user terminal. These systems select data from the input records and, depending on the value of fields in the records, take some actions specified in the program. They may then write back the result of the computation to the database and format the input and computed output for printing.

The architecture of batch processing systems has three major components, as illustrated in Figure 28.1. An input component collects inputs from one or more sources; a processing component makes computations using these inputs; and an output component generates outputs to be written back to the database and printed. For example, a telephone billing system takes customer records and telephone meter readings (inputs) from an exchange switch, computes the costs for each customer (process) and then prints bills (outputs) for each customer.

The input, processing and output components may themselves be further decomposed into an input-process-output structure. For example:

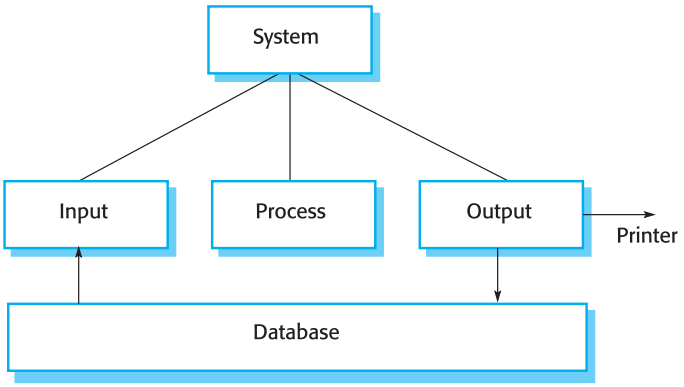

Figure 28.1 An input-process-output model of a data processing system

1. An input component may read some data (input) from a file or database, check the validity of that data and correct some errors (process), then queue the valid data for processing (output).
2. A processing component may take a transaction from a queue (input), perform some computations on the data and create a new data record recording the results of the computation (process), then queue this new record for printing (output). Sometimes the processing is done within the system database and sometimes it is a separate program.
3. An output component may read records from a queue (input), format these according to the output form (process), then send them to a printer or write new records back to the database (output).

The nature of data processing systems where records or transactions are processed serially with no need to maintain state across transactions means that these systems are naturally function-oriented rather than object-oriented. Functions are components that do not maintain internal state information from one invocation to another. Data-flow diagrams, introduced in Chapter 5, are a good way to describe the architecture of business data processing systems.

Data-flow diagrams are a way of representing function-oriented systems where each round-edged rectangle in the data flow represents a function that implements some data transformation, and each arrow represents a data item that is processed by the function. Files or data stores are represented as rectangles. The advantage of data-flow diagrams is that they show end-to-end processing. That is, you can see all of the functions that act on data as it moves through the stages of the system. The fundamental data-flow structure consists of an input function that passes data to a processing function and then to an output function.

Figure 28.2 illustrates how data-flow diagrams can be used to show a more detailed view of the architecture of a data processing system. This figure shows the

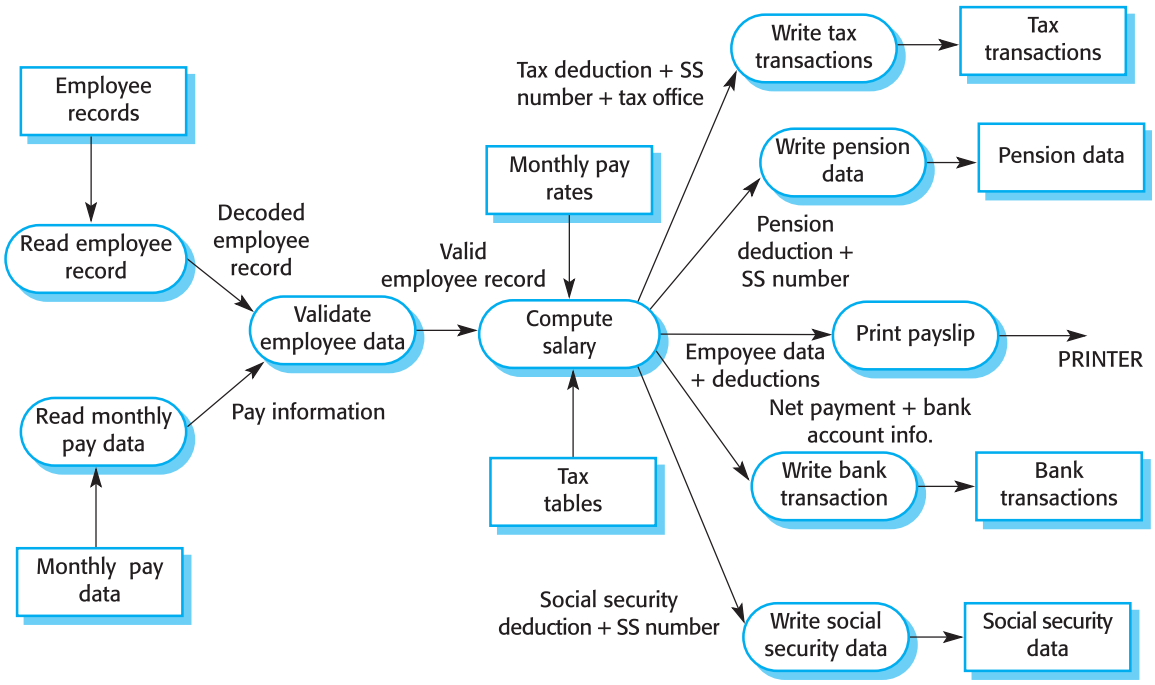

Figure 28.2 Data-flow diagram of a payroll system

design of a salary payment system. In this system, information about employees in the organisation is read into the system, monthly salary and deductions are computed, and payments are made. You can see how this system follows the basic input-process-output structure:

1. The functions on the left of the diagram 'Read employee record', 'Read monthly pay data' and 'Validate employee data' input the data for each employee and check that data.
2. The Compute salary function works out the total gross salary for each employee and the various deductions that are made from that salary. The net monthly salary is then computed.
3. The output functions write a series of files that hold details of the deductions made and the salary to be paid. These files are processed by other programs once details for all employees have been computed. A payslip for the employee, recording the net pay and the deductions made, is printed by the system.

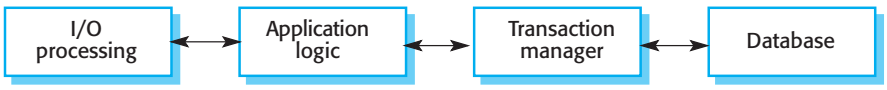

Figure 28.3 The structure of transaction processing applications

The architectural model of data processing programs is relatively simple. However, in those systems the complexity of the application is often reflected in the data being processed. Designing the system architecture therefore involves thinking about the data architecture (Bracket, 1994) as well as the program architecture. The design of data architectures is outside the scope of this book.

28.2 Transaction processing systems

Transaction processing systems are designed to process user requests for information from a database or requests to update the database (Lewis, *et al.*, 2003). Technically, a database transaction is sequence of operations that is treated as a single unit (an atomic unit). All of the operations in a transaction have to be completed before the database changes are made permanent. This means that failure of operations within the transaction do not lead to inconsistencies in the database.

An example of a transaction is a customer request to withdraw money from a bank account using an ATM. This involves getting details of the customer's account, checking the balance, modifying the balance by the amount withdrawn and sending commands to the ATM to deliver the cash. Until all of these steps have been completed, the transaction is incomplete and the customer accounts database is not changed.

From a user perspective, a transaction is any coherent sequence of operations that satisfies a goal, such as 'find the times of flights from London to Paris'. If the user transaction does not require the database to be changed then it may not be necessary to package this as a technical database transaction.

Transaction processing systems are usually interactive systems where users make asynchronous requests for service. Figure 28.3 illustrates the high-level architectural structure of these applications. First, a user makes a request to the system through an I/O processing component. The request is processed by some application-specific logic. A transaction is created and passed to a transaction manager, which is usually embedded in the database management system. After the transaction manager has ensured that the transaction is properly completed it signals to the application that processing has finished.

The input-process-output structure that we can see in data processing applications also applies to many transaction processing systems. Some of these systems are interactive versions of batch processing systems. For example, at one time banks input all customer transactions off-line then ran these transactions in a batch

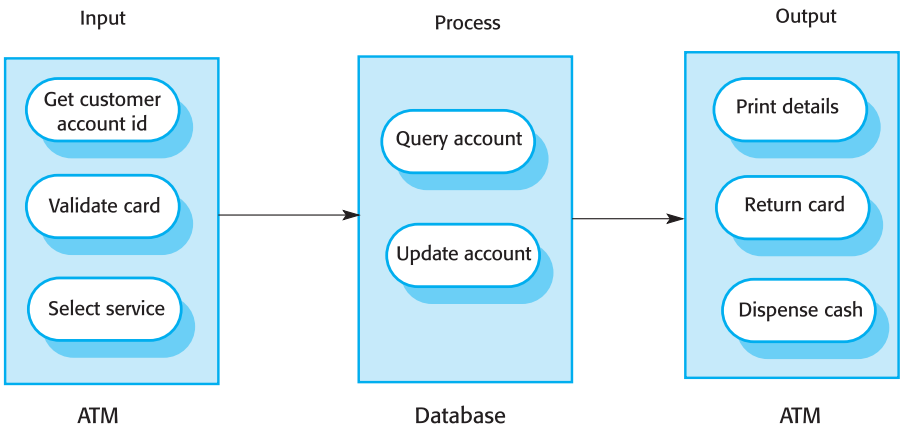

Figure 28.4 The software architecture of an ATM

against their accounts database every evening. This approach has mostly been replaced by interactive, transaction-based systems that update accounts in real time.

An example of a transaction processing system is a banking system that allows customers to query their accounts and withdraw cash from an ATM. The system is composed of two cooperating software sub-systems—the ATM software and the account processing software in the bank's database server. The input and output sub-systems are implemented as software in the ATM, whereas the processing sub-system is in the bank's database server. Figure 28.4 the architecture of this system. I have added some detail to the basic input-process-output diagram to show components that may be involved in the input, processing and output activities. I have deliberately not suggested how these internal components interact, as the sequence of operation may differ from one machine to another.

In systems such as a bank customer accounting systems, there may be different ways to interact with the system. Many customers will interact through ATMs, but bank staff will use counter terminals to access the system. There may be several types of ATMs and counter terminals used, and some customers and staff may access the account data through web browsers.

To simplify the management of different terminal communication protocols, large-scale transaction processing systems may include middleware that communicates with all types of terminal, organises and serialises the data from terminals, and sends that data for processing. This middleware is sometimes called a 'teleprocessing monitor' or a 'transaction management' system. IBM's CICS (Horswill and Miller, 2000) is a very widely used example of such a system.

Figure 28.5 shows another view of the architecture of a customer accounting system that handles personal account transactions from ATMs and counter terminals in a bank. The teleprocessing monitor handles the input and serialises transactions, which it converts to database queries. The query processing takes place in the database

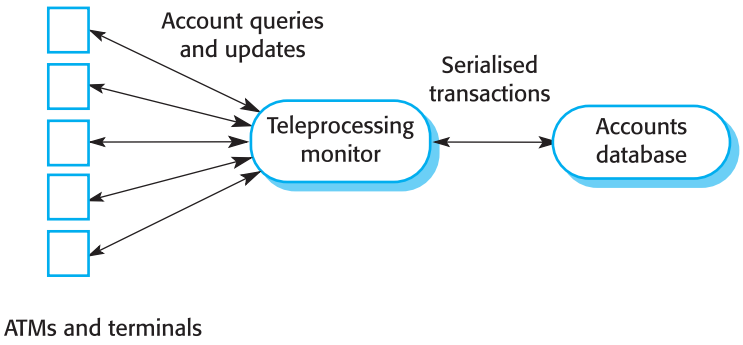

Figure 28.5 Middleware for transaction management

management system. Results are passed back to the teleprocessing monitor, which keeps track of terminals making the request. This system then organises the data into a form that can be handled by the terminal software and returns the results of the transaction to it.

28.2.1 Information and resource management systems

All systems that involve interaction with a shared database can be considered to be transaction-based information systems. An information system allows controlled access to a large base of information, such as a library catalogue, a flight timetable or the records of patients in a hospital. The development of the WWW meant that a huge number of information systems moved from being specialist organisational systems to universally accessible general-purpose systems.

Figure 28.6 is a very general model of an information system. The system is modelled using a layered or abstract machine approach (discussed in Chapter 6), where the top layer supports the user interface and the bottom layer the system database. The user communications layer handles all input and output from the user interface, and the information retrieval layer includes application-specific logic for accessing and updating the database. As we shall see later, the layers in this model can map directly onto servers in an Internet-based system.

As an example of an instantiation of this layered model, Figure 28.7 presents the architecture of a library system called LIBSYS. This system allows users to access documents in remote libraries and download these for printing, taking copyright considerations into account. I have added detail to each layer in the model by identifying the components that support user communications and information retrieval and access. You should also notice that the database is a distributed database. Users actually connect, through the system, to the databases of the libraries that provide documents.

The user communication layer in Figure 28.7 includes three major components:

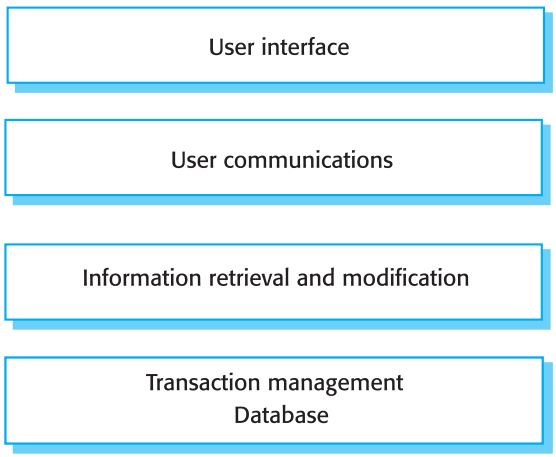

Figure 28.6 A layered model of an information system

1. The LIBSYS login component identifies and authenticates users. All information systems that restrict access to a known set of users need to have user authentication as a fundamental part of their user communication systems. User authentication can be personal but, in e-commerce systems, may also require credit card details to be provided.
2. The form and query manager component manages the forms that may be presented to the user and provides query facilities allowing the user to request information from the system. Again, all information systems must include a component that provides these facilities.
3. The print manager component is specific to LIBSYS. It controls the printing of documents that, for copyright reasons, may be restricted. For example, some documents may only be printed once on printers of the registered library.

The information retrieval and modification layer in the LIBSYS system includes application-specific components that implement the system's functionality. These components are:

1. *Distributed search* This component searches for documents in response to user queries across all of the libraries that have registered with the system. The list of known libraries is maintained in the library index.
2. *Document retrieval* This component retrieves the document or documents that are required by the user to the server where the LIBSYS system is running.
3. *Rights manager* This component handles all aspects of digital rights management and copyright. It keeps track of who has requested documents and,

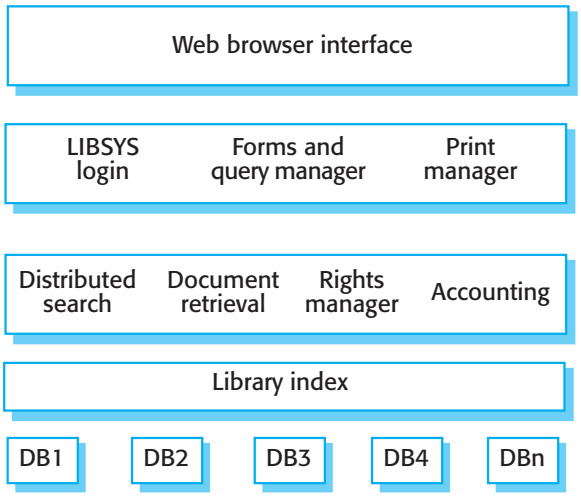

Figure 28.7 The architecture of the LIBSYS system

for example, ensures that multiple requests for the same document cannot be made by the same person.

4. *Accounting* This component logs all requests and, if necessary, handles any charges that are made by the libraries in the system. It also produces management reports on the use of the system.

We can see the same, four-layer generic structure in another type of information system, namely systems that are designed to support resource allocation. Resource allocation systems manage a fixed amount of some given resource, such as tickets for a concert or a football game, that must be allocated to users who request that resource from the supplier. Ticketing systems are an obvious example of a resource allocation system, but a large number of apparently dissimilar programs are also actually resource allocation systems. Some examples of this class of system are:

1. Timetabling systems that allocate classes to timetable slots. The resource being allocated here is a time period and there are usually a large number of constraints associated with each demand for the resource.
2. Library systems that manage the lending and withdrawal of books or other items. In this case, the resources being allocated are the items that may be borrowed. In this type of system, the resources are not simply allocated but must sometimes be deallocated from the user of the resource.
3. Air traffic management systems where the resource that is being allocated is a segment of airspace so that separation is maintained between the planes that are

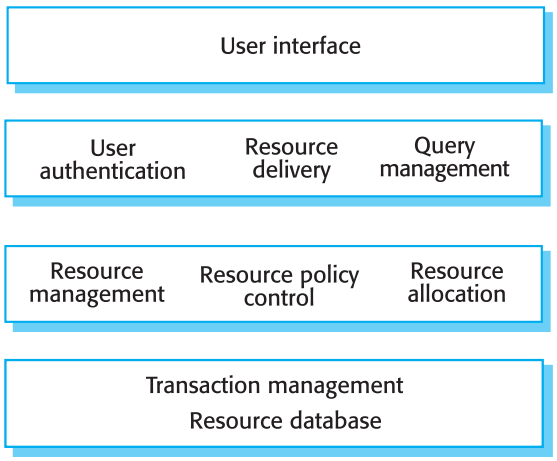

Figure 28.8 A layered model of a resource allocation system

being managed by the system. Again, this involves dynamic allocation and reallocation of resource, but the resource is a virtual rather than a physical resource.

Resource allocation systems are a very widely used class of application. If we look at their architecture in detail, we can see how it is aligned with the information system model shown in Figure 28.6. The components of a resource allocation system (shown in Figure 28.8) include:

1. A resource database that holds details of the resources being allocated. Resources may be added or removed from this database. For example, in a library system, the resource database includes details of all items that may be borrowed by users of the library. Normally, this is implemented using a database management system that includes a transaction processing system. The database management system also includes resource-locking facilities so that the same resource cannot be allocated to users who make simultaneous requests.
2. A rule set that describes the rules of resource allocation. For example, a library system normally limits who may be allocated a resource (registered library users), the length of time that a book or other item may be borrowed, the maximum number of books that may be borrowed and so on. This is encapsulated in the resource policy control component.
3. A resource management component that allows the provider of the resources to add, edit or delete resources from the system.

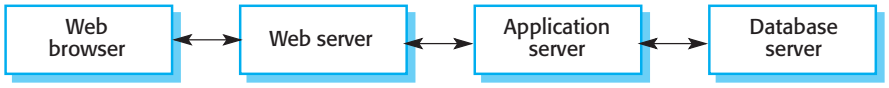

Figure 28.9 A multi-tier Internet transaction processing system

4. A resource allocation component that updates the resource database when resources are assigned and that associates these resources with details of the resource requestor.

5. A user authentication module that allows the system to check that resources are being allocated to an accredited user. In a library system this might be a machine-readable library card; in a ticket allocation system it could be a credit card that verifies the user is able to pay for the resource.

6. A query management module that allows users to discover what resources are available. In a library system, this would typically be based around queries for particular items; in a ticketing system, it could involve a graphical display showing what tickets are available for particular dates.

7. A resource delivery component that prepares the resources for delivery to the requestor. In a ticketing system, this might involve preparing an e-mail confirmation and sending a request to a ticket printer to print the tickets and the details of where these should be posted

8. A user interface component (often a web browser) that is outside the system and allows the requester of the resource to issue queries and requests for the resource to be allocated.

This layered architecture can be realised in several ways. Information systems software can be organised so that each layer is a large-scale component running on a separate server. Each layer defines its external interfaces and all communication takes place through these interfaces. Alternatively, if the entire information system executes on a single computer, then the middle layers are usually implemented as a single program that communicates with the database through its API. A third alternative is to implement finer-grain components as separate web services (discussed in Chapter 19) and compose these dynamically according to the user's requests.

Implementations of information and resource management systems based on Internet protocols are now the norm; the user interface in these systems is implemented using a web browser. These systems are usually implemented as multi-tier client server/architectures, as discussed in Chapter 18. The system organisation is shown in Figure 28.9. The web server is responsible for all user communications; the application server is responsible for implementing application-specific logic as well as information storage and retrieval requests; the database server moves information to and from the database. Using multiple servers allows high throughput and makes it possible to handle hundreds of transactions per minute.

E-commerce systems are Internet-based resource management systems that are designed to accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer. There is a wide range of these systems now in use ranging from systems that allow services such as car-hire to be arranged to systems that support the order of tangible goods such as books or groceries. In an e-commerce system, the application-specific layer includes additional functionality supporting a 'shopping cart' in which users can place a number of items in separate transactions, then pay for them all together in a single transaction.

28.3 Event processing systems

Event processing systems respond to events in the system's environment or user interface. As I discussed in Chapter 6**,** the key characteristic of event processing systems is that the timing of events is unpredictable and the system must be able to cope with these events when they occur.

We all use such event-based systems like this on our own computers—word processors, presentation systems and games are all driven by events from the user interface. The system detects and interprets events. User interface events represent implicit commands to the system, which takes some action to obey that command. For example, if you are using a word processor and you double-click on a word, the double-click event means 'select that word'.

Real-time systems, which take action in 'real time' in response to some external stimulus, are also event-based processing systems. However, for real-time systems, events are not usually user interface events but events associated with servers or actuators in the system. Because of the need for real-time response to unpredictable events, these real-time systems are normally organised as a set of cooperating processes. I cover generic architectures for real-time systems in Chapter 20.

In this section, I focus on describing the generic architecture of editing systems. Editing systems are programs that run on PCs or workstations that allow users to edit documents such as text documents, diagrams or images. Some editors focus on editing a single type of document, such as images from a digital camera or scanner. Others, including most word processors, are multi-editors and include support for editing different types including text and diagrams. You can even think of a spreadsheet as an editing system where you edit boxes on the sheet. Of course, spreadsheets have additional functionality to carry out computations.

Editing systems have a number of characteristics that distinguish them from other types of system and that influence their architectural design:

1. Editing systems are mostly single-user systems. They therefore don't have to deal with the problems of multiple concurrent access to data and have simpler data management than transaction-based systems. Even where data are shared,

transaction management is not usually used because transactions take a long time and alternative methods of maintaining data integrity are used.

2. They have to provide rapid feedback on user actions such as 'select' and 'delete'. This means they have to operate on representations of data that is held in computer memory rather than on disk. Because the data is in volatile memory, it can be lost if there is a system fault, so editing systems should make some provision for error recovery.

3. Editing sessions are normally much longer than sessions involving ordering goods, or making some other transaction. This again means that there is a greater risk of loss if problems arise. Therefore, many editing systems include recovery facilities that automatically save work in progress and recover this for the user in the event of a system failure.

A generic architecture for an editing system is shown **in** Error! Reference source not found. as a set of interacting objects. The objects in the system are active rather than passive and can operate concurrently and autonomously. Essentially, screen events are processed and interpreted as commands. This updates a data structure, which is then redisplayed on the screen.

The responsibilities of the architectural components shown in Figure 28.10 are:

1. *Screen* This object monitors the screen memory segment and detects events that occur. These events are then passed to the event processing object along with their screen coordinates.

2. *Event* This object is triggered by an event arriving from Screen. It uses knowledge of what is displayed to interpret this event and to translate this into the appropriate editing command. This command is then passed to the object responsible for command interpretation. For very common events, such as mouse clicks or key presses, the event object can communicate directly with the data structure. This allows faster updates of that structure.

3. *Command* This object processes a command from the event object and calls the appropriate method in the Editor data object to execute the command.

4. *Editor data* When the appropriate command method in Editor data object is called, it updates the data structure and calls the Update method in Display to display the modified data.

5. *Ancillary data* As well as the data structure itself, editors manage other data such as styles and preferences. In this simple architectural model, I have bundled this together under Ancillary data. Some editor commands, such as a command to initiate a spelling check, are implemented by a method in this object.

6. *File system* This object handles all opening and saving of files. These can be either editor data or ancillary data files. To avoid data loss, many editors have

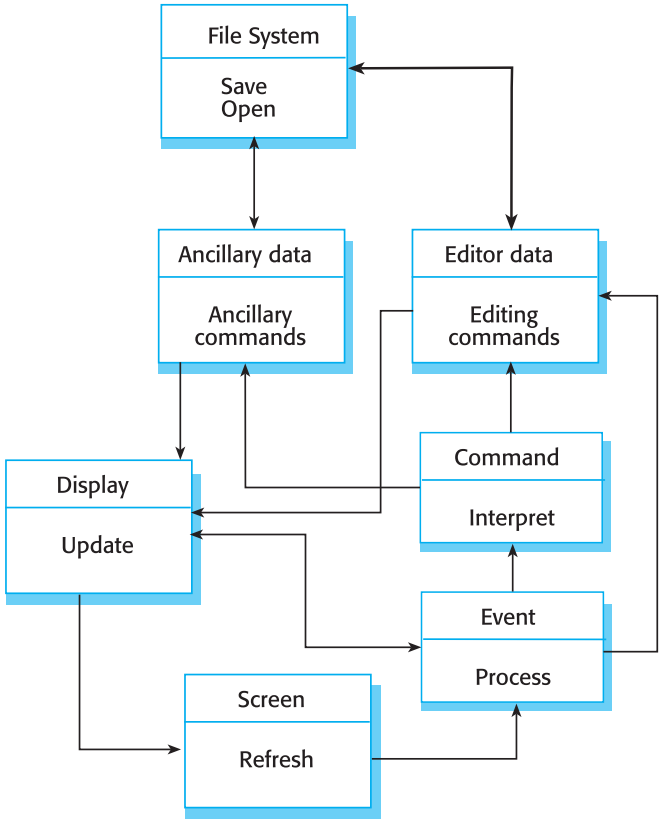

Figure 28.10 The abstract architecture of an editing system

auto-save facilities that save the data structure automatically. This can then be retrieved in the event of system failure.

7. *Display* This object keeps track of the organisation of the screen display. It calls the Refresh method in Screen when the display has been changed.

Because of the need for a rapid response to user commands, editing systems do not have a central controller that calls the components to take action. Rather, the critical components in the system execute concurrently and can communicate directly (e.g., the event processor can communicate directly with the editor data structure) so that faster performance can be achieved.

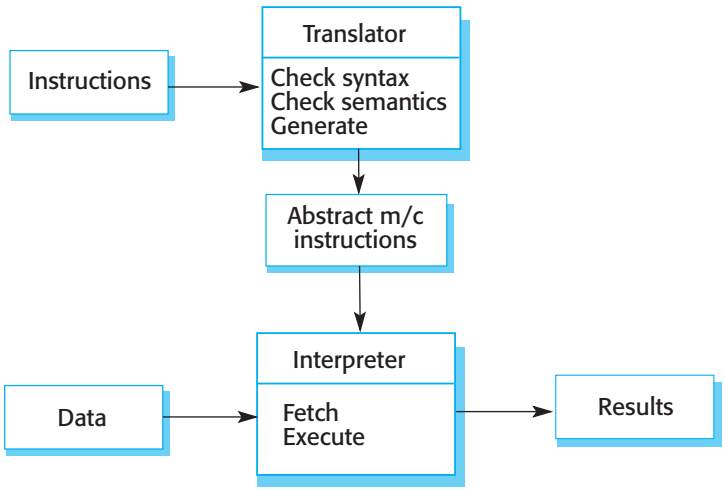

Figure 28.11 The abstract architecture of a language processing system

28.4 Language processing systems

Language processing systems accept a natural or artificial language as an input and generate some other representation of that language as an output. In software engineering, the most widely used language processing systems are compilers that translate an artificial high-level programming language into machine code, but other language-processing systems translate an XML data description into commands to query a database and natural language processing systems that attempt to translate one natural language to another.

At the most abstract level, the architecture of a language processing system is illustrated in Figure 28.11. The instructions describe what has to be done. These are translated into some internal format by a translator. The instructions correspond to the machine instructions for an abstract machine. These instructions are then interpreted by another component that fetches the instructions for execution and executes them using, if necessary, data from the environment. The output of the process is the result of interpreting the instructions on the input data. Of course, for many compilers, the interpreter is a hardware unit that processes machine instructions and the abstract machine is a real processor. However, for many languages, such as Java, the interpreter is a software component.

Language processing systems are used in situations where the easiest way to solve a problem is to specify that solution as an algorithm or as a description of the system data. For example, meta-CASE tools are program generators that are used to

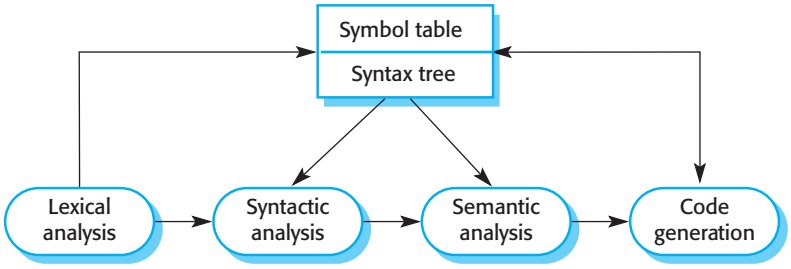

Figure 28.12 A data-flow model of a compiler

create specific CASE tools to support software engineering methods. Meta-CASE tools include a description of the method components, its rules and so on, written in a special-purpose language that is parsed and analysed to configure the generated CASE tool.

Translators in a language processing system have a generic architecture (Figure 28.12) that includes the following components:

1. A lexical analyser that takes input language tokens and converts them to an internal form.
2. A symbol table, which holds information about the names of entities (variables, class names, object names, etc.) used in the text that is being translated.
3. A syntax analyser, which checks the syntax of the language being translated. It uses a defined grammar of the language and builds a syntax tree.
4. A syntax tree, which is an internal structure representing the program being compiled.
5. A semantic analyser that uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
6. A code generator that 'walks' the syntax tree and generates abstract machine code.

Other components might also be included which transform the syntax tree to improve efficiency and remove redundancy from the generated machine code. In other types of language processing system, such as a natural language translator, the generated code is actually the input text translated into another language.

The components that make up a language processing system can be organised according to different architectural models. As Garlan and Shaw point out (Garlan and Shaw, 1993), compilers can be implemented using a composite model. A data-flow architecture may be used with the symbol table acting as a repository for shared data. The phases of lexical, syntactic and semantic analysis are organised sequentially, as shown in Figure 28.12.

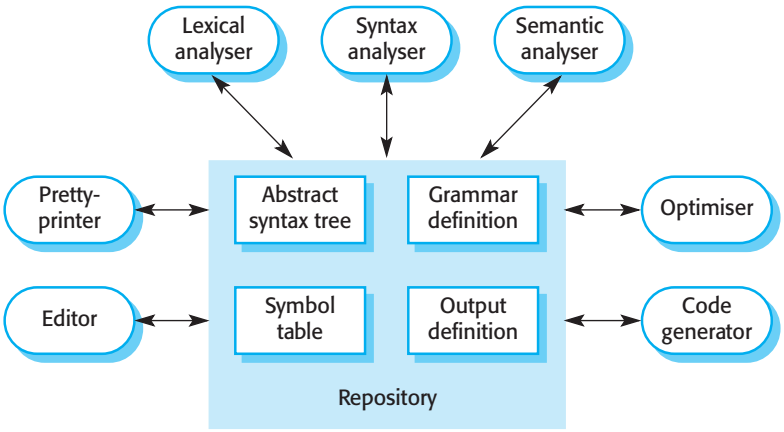

Figure 28.13 The repository model of a compiler

This data-flow model of compilation is still widely used. It is effective in batch environments where programs are compiled and executed without user interaction. It is less effective when the compiler is to be integrated with other language processing tools such as a structured editing system, an interactive debugger or a program prettyprinter. The generic system components can then be organised in a repository-based model, as shown in Figure 28.13.

This figure illustrates how a language processing system can be part of an integrated set of programming support tools. In this example, the symbol table and syntax tree act as a central information repository. Tools or tool fragments communicate through it. Other information that is sometimes embedded in tools, such as the grammar definition and the definition of the output format for the program, have been taken out of the tools and put into the repository. Therefore, a syntax-directed editor can check that the syntax of a program is correct as it is being typed and a prettyprinter can create listings of the program in a format that is easy to read.

KEY POINTS

- Generic models of application systems architectures help us understand the operation of applications, compare applications of the

same type, validate application system designs and assess large-scale components for reuse.

- Many applications either fall into one of four classes of generic application or are combinations of these generic applications. The four types of generic application covered here are data processing systems, transaction processing systems, event processing systems and language processing systems.
- Data processing systems operate in batch mode and generally have an input-process-output structure. Records are input into the system, the information is processed and outputs are generated.
- Transaction processing systems are interactive systems that allow information in a database to be remotely accessed and modified by a number of users. Information systems and resource management systems are examples of transaction processing systems.
- Event processing systems include editing systems and real-time systems. In an editing system, user interface events are interpreted and an in-store data structure is modified. Word processors and presentation systems are examples of editing systems.
- Language processing systems are used to translate texts from one language into another and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.

FURTHER READING

The topic of application architectures has been largely neglected; authors of books and articles on software architecture tend to focus on abstract principles or product line architectures.

Design and Use of Software Architectures. This book takes a product-line approach to software architectures and therefore discusses architecture from an application perspective. (J. Bosch, 2000, Addison-Wesley)

Databases and Transaction Processing: An Application-oriented Approach. This is not really a book on software architecture, but it discusses the principles of transaction processing and data-centric applications. (P.M. Lewis, A. J. Bernstein and M. Kifer, 2003, Addison-Wesley)

EXERCISES

28.1 Explain how the generic applications architectures described here can be used to help the designer make decisions about software reuse.

28.2 Using the four basic application types introduced in this chapter, classify the following systems and explain your classification:

A point-of-sale system in a supermarket
A system that sends out reminders that magazine subscriptions are due to be paid
A photo album system that provides some facilities for restoring old photographs
A system that reads web pages to visually disabled users
An interactive game in which characters move around, cross obstacles and collect treasure
An inventory control system that keeps track of what items are in stock and automatically generates orders for new stock then the level falls below a certain value.

28.3 Based on an input-process-output model, expand the Compute salary function in Figure 28.2 and draw an activity diagram that shows the computations carried out in that function. You need the following information to do this:

- The employee record identifies the grade of an employee. This grade is then used to look up the table of pay rates.
- Employees below a particular grade may be paid overtime at the same rate as their normal hourly pay rate. The extra hours for which they are to be paid are indicated in their employee record.
- The amount of tax deducted depends on the employee's tax code (indicated in the record) and their annual salary. Monthly deductions for each code and a standard salary are indicated in the tax tables. These are scaled up or down depending on the relationship between the actual salary and the standard salary used.

28.4 Explain why transaction management is necessary in systems where user inputs can result in database changes.

28.5 Using the basic model of an information system as presented in Figure 28.6, show the components of an information system that allows users to view information about flights arriving and departing from a particular airport.

28.6 Using the layered architecture shown in Figure 28.8, show the components of a resource management system that could be used to handle hotel room bookings.

28.7 In an editing system, all user interface events can be translated into implicit or explicit commands. Explain why, in Figure 28.10, the Event object therefore communicates directly with the editor data structure as well as the Command object.

28.8 Modify Figure 28.10 to show the generic architecture of a spreadsheet system. Base your design on the features of any spreadsheet system that you have used.

28.9 What is the function of the syntax tree component in a language processing system?

28.10 Using the generic model of a language processing system presented here, design the architecture of a system that accepts natural language commands and translates these into database queries in a language such as SQL.

REFERENCES

Appelrath, H.-J. and Ritter, J. (2000). SAP R/3 Implementation: Methods and Tools (SAP Excellence). Berlin: Springer-Verlag.

Bracket, M. H. (1994). Data Sharing using a Common Data Architecture. New York: John Wiley & Sons.

Garlan, D. and Shaw, M. (1993). 'An Introduction to Software Architecture'. Advances in Software Engineering and Knowledge Engineering, 1 1-39.

Harold, E. R. and Means, W. S. (2002). XML in a Nutshell. Sebastopol. Ca.: O'Reilly.

Horswill, J. and Miller, S. A. (2000). Designing and Programming CICS Applications. Sebastopol, Ca.: O'Reilly.

Lewis, P. M., Bernstein, A. J. and Kifer, M. (2003). Databases and Transaction Processing: An Application-oriented Approach. Boston: Addison-Wesley.