14

System Design

14.1a. An electronic chess companion combines features of interactive interface and batch transformation architectures. Entering and displaying moves is dominated by interactions between the chess companion and the human user. Once a human has entered a move, the activity of the chess companion is a batch transformation in which the positions of the pieces on the board are used to compute the next move.

b. An airplane flight simulator for a video game system combines features of interactive interface and dynamic simulation architectures. Interactive features include interpreting the joystick input and displaying cockpit information and the view of the terrain. Simulation involves computing the motion of the airplane. Another acceptable answer is interactive interface and continuous transformation, since computing the motion of the airplane is done by a continuous transformation of the joystick input.

c. A floppy disk controller chip is predominately a real-time system. With the possible exception of computing a cyclic redundancy code, there are practically no computations. Strict timing constraints must be satisfied to avoid losing data. The most important part of the design is the state diagram.

d. A sonar system is mainly a continuous transformation with some real time and some interactive interface aspects. Converting a continuous stream of pulses and echoes into range information is a continuous transformation. Because the system must process the data as fast as it comes in, there is a real-time flavor. Display of the range information involves interactive interface issues.

14.2a. Procedural control is adequate for the chess companion since the user interface is simple.

b. Event-driven control would be best for the flight simulator. Then it could accept push button events and a timing interrupt could pace the display update so that the apparent flight velocity does not change with the complexity of the calculations. Procedural control can be used, but the quality of the simulation would suffer. We assume that the ter-

rain is static; otherwise we would consider concurrent control to handle the motion of other objects.

c. A floppy disk controller requires hardware control, not software control. Hardware controls are typically designed using state diagrams, which are relatively easy to convert into hardware. One way to do this is to use micro-coding. The state diagrams are converted into a procedure which in turn is implemented as a sequence of micro-instructions. Each output of the controller is typically controlled by one of the bit fields in an instruction. Instructions for branching and looping based on inputs and controller registers are usually available.

d. The real time aspect of the sonar system makes concurrent control desirable.

14.3 It is totally out of the question to store the data samples without buffering. For example, during the 10 milliseconds that the read/write head takes to move from one track to the next, 160 bytes would be lost. It is clear that a buffer is needed. We will discuss some of the constraints and control issues. Then we will present a couple of approaches.

A system architect would probably question the decision to limit the buffer size to 64000 bytes. Memory is cheap, and 160000 bytes would be enough to store all of the samples, greatly simplifying the design. However, within the context of this problem, 64000 bytes is a constraint.

Careful system design can reduce the size of the buffer. It is a good idea to store the data in sequential tracks to minimize the time spent in moving from one track to the next. Beyond that, there are special track formats to reduce the time lost moving between tracks. For example, if only a portion of a track is used or if the tracks are carefully arranged, the beginning of one track could be at the read/write head just as the head has completed an adjacent track.

Another dimension to consider is whether or not the buffer can be emptied while a track is being written. If it can, there is some hope of making the buffer only as large as required to store data that would be lost during the head motion between tracks. Otherwise the buffer may have to hold several tracks of data. First, estimate the rate that data can be written to the disk. The average time to find the beginning of a track, 83 milliseconds, is approximately equal to the time for the disk to turn one half of a revolution, so it takes about 166 milliseconds to make a complete turn. Therefore, the disk is revolving 6 times in a second. The number of bytes per track is approximately equal to the total storage, 243000 bytes, divided by the number of tracks, 77, or about 3155 bytes per track. (51 tracks are needed, leaving 26 tracks on the disk for other functions.) The rate at which data could be written to the disk is equal to the bytes per track times the spinning rate, or about 18935 bytes per second. This is greater than the rate of data input so there is some hope of sizing the buffer for a single track.

There are two control aspects to consider: initiating data storage and servicing the analog to digital converters and the disk drive.

We must assume that the initiation of data storage is to be controlled externally, because there is no reason for us to believe that our product will be able to control the events that are generating the data.

Another aspect of control is servicing the converters and the disk drive. Two common approaches are polling and interrupt driven. Polling involves continuously checking the status of both devices and transferring data as needed. This approach simplifies the hardware design at the expense of tying up the CPU during the capture of data. The interrupt driven approach relies on a hardware interrupt controller to notify the CPU when external devices need servicing, freeing up the CPU to do other tasks in the meantime. Our choice will depend on whether or not there is anything to do while data is being collected. It is also entirely possible that the interrupt driven approach has been selected to satisfy the requirements of other functions of the product, so we would want to explore further before making a decision.

We now analyze a couple of approaches:

- Sequential tracks, standard formatting with a buffer that holds several tracks

The simplest solution is to not use any special disk formatting, for which a worst case analysis is indicated. For each 3155 byte track that is written, it will take 10 milliseconds to position the head, up to 166 milliseconds for the beginning of the track to come around, and 166 milliseconds to write the track. During this time, 5488 bytes come in, so the buffer is filling faster that it can be emptied.

Without special formatting, it takes 342 milliseconds to complete a track. During the 10 seconds that it takes for all data to come in, a little over 29 tracks can be written. To be on the safe side, plan on writing 91495 bytes to the disk in 10 seconds. This leaves over 68000 bytes for the buffer, so this approach will not work unless we increase the amount of memory in the system.

- Sequential tracks, special disk formatting with a buffer for one track

If the disk is formatted carefully, the time to find the beginning of a track after completing an adjacent track can be reduced. This time will not be zero, because we will want some margin for error. The problem statement does not give any information about variations in disk speed, but 5 milliseconds (representing a 3 percent variation in disk speed) should give us sufficient margin for error. By careful formatting, the total time to write a track could be cut to 181 milliseconds. During this time, only 2896 bytes is input, which is less than a track, so the strategy could work. Because the system has no control over when the data starts coming in, it must be prepared to buffer one complete track of data. This approach will work with a buffer of 3155 bytes.

14.4 This system combines features of an interactive interface with a batch transformation. An interactive interface is needed to edit drawings. Converting a drawing to an N/C tape is a batch transformation. Determining an efficient sequence of drilling operations is a form of the traveling salesman problem. The optimum solution would take a long time to compute, and is not really needed for this application. There are several engineering solutions that produce a reasonably efficient sequence of operations without using a lot of computing resources. An excellent algorithm is given by John D. Litke in “An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes”, *Communications of the ACM*, Volume 27, Number 12, December 1984, 1227–1236.

It is convenient to break the system into three major subsystems: editor, planner, and puncher. The editor could be subdivided into user interface, graphics display and file interface. Everything runs on a PC, so there is no question of assigning subsystems to hardware devices. The editor is used to prepare drawings. The planner determines a sequence of operations to accomplish the drilling. The puncher, which incorporates a driver for operating a punch, prepares an N/C tape.

There are no concurrency issues. A sequential approach is adequate. Experience has shown that a PC is adequate for two dimensional drafting.

Control must be considered for the editor and the puncher. Either procedural control or event driven control is suitable for the puncher. The editor should be event driven. Global resources are adequately handled by the operating system of the PC.

14.5 The interface is simplified by being line oriented, leaving the bulk of the work to be done on the batch transformations required to execute the commands.

There are three major subsystems: user interface, file interface and expression handling. There is no inherent concurrency. We assume that the system operates on a sufficiently powerful computer. Files are adequate for storing previous work. One design task is to devise a language for representing expressions. A procedural approach is adequate for control.

14.6 Figure A14.1 shows one possible partitioning.

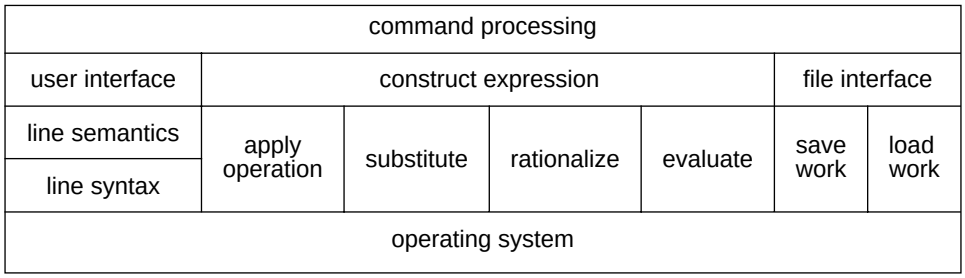

Figure A14.1 Block diagram for an interactive polynomial symbolic manipulation system

14.7 A single program provides faster detection and correction of errors and eliminates the need to implement an interface between two programs. With a single program, any errors that the system detects in the process of converting the class diagram to a database schema can be quickly communicated to the user for correction. Also, the editing and the conversion portions of the program can share the same data, eliminating the need for an interface such as a file to transfer the class diagram from one program to another.

Splitting the functionality into two programs reduces memory requirements and decouples program development. The total memory requirement of a single program would be approximately equal to the sum of the requirements of two separate programs.

Since both programs are likely to use a great deal of memory, performance problems could arise if they are combined. Using two separate programs also simplifies program development. The two programs can be developed independently, so that changes made in one are less likely to impact the other. Also, two programs are easier to debug than one monolithic program. If the interface between the two programs is well defined, problems in the overall system can be quickly identified within one program or the other.

Another advantage of splitting the system into two programs is greater flexibility. The editor can be used with other back ends such as generating language code declarations. The relational database schema generator can be adapted to other graphical front ends.

14.8a. A single geometrical model completely describes a class diagram, but would be too low level a representation. It is tedious to construct class diagrams by drawing lines, boxes, text, and so forth. Deriving the logical model from the physical representation is possible, but is time consuming and ambiguous.

b. Two separate models. This is the best solution, because it decouples two separate aspects of a class diagram, making both aspects less cluttered and easier to understand during system design. The geometrical model is useful for preparing printouts. The logical model is useful for semantic checking and interacting with the backend programs which need to know what the diagram means but do not care about the precise manner in which it is drawn.

14.9 The fastest way to implement the system is to use a commercially available desktop publishing system to edit the class diagrams. This eliminates the need to design and implement interactive graphics software, a step that is time consuming and error prone. As a bonus, there will be probably be good user documentation for the desktop system and vendor support. The impact of future enhancements made by the vendor is uncertain. On the one hand, the added functionality may make the overall system more attractive. On the other hand, future releases of the commercial system may change the markup language.

Implementing your own editor will result in a more robust system because the system can be designed to allow the user to draw only valid class diagrams. With the desktop editor, it is possible to draw pictures which cannot be interpreted as class diagrams. Also, because your own editor can be customized to the semantics of class diagrams, it will be easier and faster to use than the general-purpose desktop publishing system.

14.10 Here is an evaluation of each solution.

a. Do not worry about it at all. Reset all data every time the system is turned on. This is the cheapest, simplest approach. It is relatively easy to program, since all that is needed is an initialization routine on power up to allow the user to enter parameters. However, this approach cannot be taken for systems which must provide continuous service or which must not lose data during power loss.

b. **Never turn the power off if it can be helped. Use a special power supply, including backup generators, if necessary**. This approach is used for critical systems which must provide continuous service no matter what. This kind of power supply is called an UPS (uninterruptable power supply) and is relatively expensive. A bonus of this approach is that the application program is simplified, since it is assumed to be always running.

c. **Keep critical information on a magnetic disk drive. Periodically make full and/or incremental copies on magnetic tape**. This approach is moderately expensive and bulky. In the event of a power failure, the system stops running. An operating system is required to cope with the disk and tape drive. An operator is required to manage the tapes, which would preclude applications where unattended operation is required.

d. **Use a battery to maintain power to the system memory when the rest of the system is off. It might even be possible to continue to provide limited functionality**. This solution is relatively cheap, compact, and requires no operator. The main drawback of this approach is having to remember to change the battery periodically. In many applications, the user may not be aware that there is a battery that needs to be changed.

e. **Use a special memory component**. This approach is relatively cheap and is automatic. However, the system cannot run when power is off. Some restrictions may apply such as a limit on the number of times data can be saved or on the amount of data that can be saved. A program may be required to save important parameters as power is failing.

f. **Critical parameters are entered by the user through switches**. This approach is well suited to certain types of data, such as options, that are set only once, and is relatively cheap. Usually only a limited amount of data can be entered this way. Of course, the program cannot save any data through this scheme.

14.11a.Four function pocket calculator. Do not worry about permanent data storage at all. All of the other options are too expensive to consider. This type of calculator sells for a few dollars and is typically used to balance checkbooks. Memory requirements are on the order of 10 bytes.

b. **Electronic typewriter**. Memory requirements are on the order of 10,000 to 100,000 bytes per document. The system does not need to function when power is off, but the work in progress should not be lost. Commercially available electronic typewriters typically use floppy disk drives, backup batteries, or special memory components. Each of these methods will preserve data for more than a year. The advantage of floppy disk drives is that they can store many documents. Backup batteries or special memory components cost less than disk drives, and are used in some of the less expensive models even though they have limited storage capacity. Some electronic typewriters use memory modules that can be removed from the typewriter. Memory modules are more expensive than floppy disks, but are more immune to the hazards that smoke, dirt, bending and liquid spills present to floppy disk drives, so are better suited to users who are not familiar with floppy disk drives.

c. System clock for a personal computer. Only a few bytes are required, but the clock must continue to run with the main power off. Battery backup is an inexpensive solution. Clock circuits can be designed that will run for 5 years from a battery.

d. Airline reservation system. The amount of data to be stored is very large. Use a combination of uninterruptable power supplies, disk drives, and magnetic tapes. This application requires that the system always is available. Expense is a secondary consideration.

e. Digital control and thermal protection unit for a motor. On the order of 10 to 100 bytes are needed. This application is sensitive to price. An uninterruptable power supply is too expensive to consider. Tape and disk drives are too fragile for the harsh environment of the application. Use a combination of switches, special memory components, and battery backup. Switches are a good way to enter parameters, since an interface is required anyway. Special memory components can store computed data. A battery can be used to continue operation with power removed but presents a maintenance problem in this application. We would question the last requirement, seeking alternatives such as assuming that the motor is hot when it is first turned on or using a sensor to measure the temperature of the motor.

14.12a. A description of the diagram, ignoring tabs, spaces, and line feeds, is:

```
(DIAGRAM
   (CLASS
      (NAME “Polygon”))
   (CLASS
      (NAME “Point”)
      (ATTRIBUTE “x”)
      (ATTRIBUTE “y”))
   (ASSOCIATION
      (END (NAME “Polygon”) ONE)
      (END (NAME “Point”) MANY)))
```

b. Data in storage and data in motion are similar in that they can convey the same information. The same format can be used to store data or to transmit it from one location to another. In fact, many operating systems provide services that work equally well on files or on streams of data. The answer to the first part of this problem could be interpreted as either a format for a file or for a data transmission. Both data in storage and data in motion have semantics and syntax that can be described using the same tools.

c. We give both production rules and a diagram for the language to describe polygons. Each production rule is a nonterminal, followed by ::=, followed by a (possibly empty) string of nonterminals and terminals. Nonterminals are lower case identifiers. The productions rules for the language are:

```
diagram ::= ( DIAGRAM list_of_polygons_opt );
list_of_polygons_opt ::=
                       | list_of_polygons;
list_of_polygons ::= polygon
```

```
                    | list_of_polygons polygon;
polygon ::= ( POLYGON list_of_points );
list_of_points ::= point
                 | list_of_points point;
point ::= ( POINT x y );
```

We have chosen to allow an empty list of polygons but a polygon must contain at least one point. Other definitions are possible. The nonterminals *x* and *y* are integers. The corresponding BNF is shown in Figure A14.2.

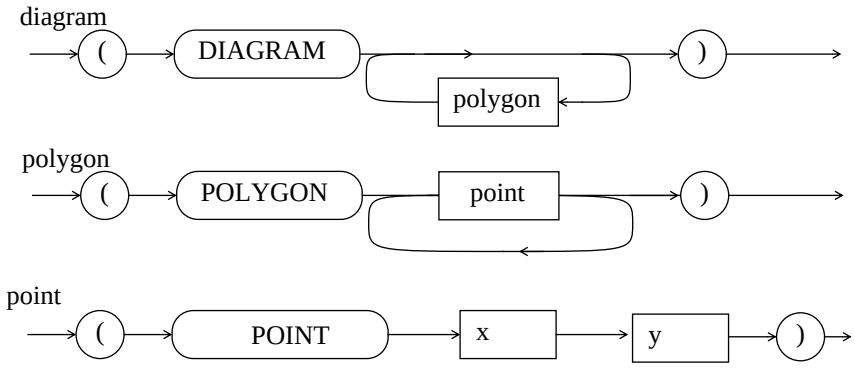

Figure A14.2 BNF diagram for a language to describe polygons

An example of a square in this language is:

```
( DIAGRAM
   ( POLYGON
      ( POINT 0 0 )
      ( POINT 10 0 )
      ( POINT 10 10 )
      ( POINT 0 10 )))
```

An example of a triangle in this language is:

```
( DIAGRAM
   ( POLYGON
      ( POINT 10 0 )
      ( POINT 10 10 )
      ( POINT 0 10 )))
```

In both cases the points are listed in the order of their connectivity.

14.13 The hardware approach is fastest, but incurs the cost of the hardware. The software approach is cheapest and most flexible, but may not be fast enough. Use the software approach whenever it is fast enough. General purpose systems favor the software approach, because of its flexibility. Special purpose systems can usually integrate the added circuitry with other hardware.

Actually, there is another approach, firmware, that may be used in hardware architectures. Typically, in this approach a hardware controller calculates the CRC under the direction of a microcoded program that is stored in a permanent memory that is not visible externally. We will count this approach as hardware.

a. Floppy disk controller. Use a hardware approach. Flexibility is not needed, since a floppy disk controller is a special purpose system. Speed is needed, because of the high data rate.

b. Transmission over telephone lines. Use a software approach. This system is a general purpose one, capable of running on a variety of computers. Data rates are relatively modest.

c. Memory board in the space shuttle. Use hardware to check memory. This is an example of a specific application, where the function can probably be integrated with the circuitry in the memory chips. The data rate is very high.

d. Magnetic tape drive. Use hardware or software, depending on the data rate and quality of the tape drive, and the application. If the drive is tightly integrated into a larger, specialized system, and the data rate is not too high, it may be cheaper to compute the CRC in software. On the other hand, if the tape drive is a general purpose one, or if the data rate is high, a hardware approach is better.

e. Validation of an account number. Use a software approach. The data rate is very low. The system handling the account number is probably running on a general purpose computer.)

14.14 Figure A14.3 extends the scheduler class models so that permissions can be assigned by group. An *AccessEntity* can belong to multiple groups. A *Group* can consist of multiple access entities. This kind of model leads to a directed graph and lets us recursively specify the composition of a group.

A schedule is owned by a user. Multiple entities can be given permission to access a schedule. There can be various kinds of permissions, such as read, insert, delete, and update. An individual user can access a schedule if he or she has direct permission or belongs to a group that has permission.

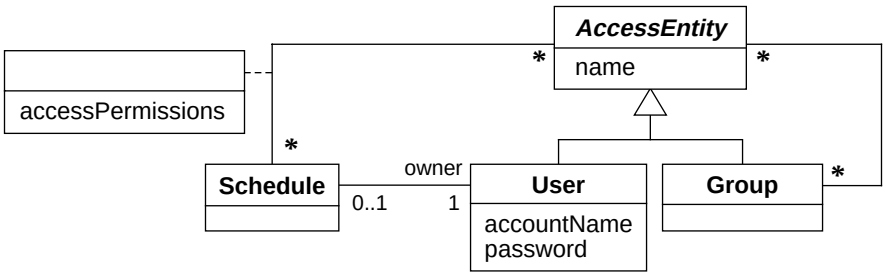

Figure A14.3 Class model for scheduler software extended for group permissions