LECTURE 13: A SHORT CASE STUDY

Software Engineering

Mike Wooldridge

2 A Small Case Study

- All modern operating systems contain *file handling subsystems*.
- Typically, files are *owned* by users of the system.
- Each file occupies a series of *disk blocks*.
- There is an upper limit to the number of files that might be owned by any one user.
- Operations which might be performed on a file system include:
  - create (add) a file to the system;
  - remove a file from the system;
- In a more sophisticated system, we might expect the following facilities (*à la* UNIX):
  - access rights (rwx for ugo);
  - hierarchical file store;
  - disk partitions;
  - ...
- We shall do a specification for a simple file system.

1 Z Documents

- A Z specification document for a system contains the following:
  - a set of schemas, which define:
    - the state space of the system;
    - the operations, or events, which can occur to the system and possibly cause state changes;
  - explanatory text, to help the reader to understand the system.

2.1 The Types

- We require three types for our specification:
  1. *USERS*
     the set of all possible system users;
  2. *FILES*
     the set of all possible file names;
  3. *BLOCKS*
     the set of all possible disk block numbers.
- To be able to use these types we must *parachute them* into our specification. We do this by including the following horizontal schema:

$$[USERS, FILES, BLOCKS]$$

- Once declared in this way, we can use them as required.

2.2 System Invariants

- What are the invariants on our file system?
  1. there is an upper limit to the number of users;
  2. files must be owned by someone;
  3. the blocks used to store files are not free for subsequent use;
  4. the blocks not used to store files are free;
  5. only system users are allowed to own files;
  6. no file can be owned by more than one user;
  7. no two files can share the same block;
  8. ...

2.3 State Space

- Here is the state-space schema:

FileStore

$$
\begin{array}{l}
owns : USERS \pfun \mathbb{P}\, FILES \\
occupies : FILES \pfun \mathbb{P}\, BLOCKS \\
system\_users : \mathbb{P}\, USERS \\
free\_blocks : \mathbb{P}\, BLOCKS \\
no\_users : \mathbb{N} \qquad \text{(* max no users *)} \\
\hline
\#system\_users \leq no\_users \\
\\
\forall f : \operatorname{dom} occupies \bullet \exists u : \operatorname{dom} owns \bullet \\
\quad f \in owns(u) \\
\\
\forall f : \operatorname{dom} occupies \bullet \forall b : BLOCKS \bullet \\
\quad b \in occupies(f) \Rightarrow b \notin free\_blocks \\
\\
\operatorname{dom} owns = system\_users \\
\\
\forall f_1, f_2 : \operatorname{ran} owns \bullet \\
\quad f_1 \neq f_2 \Rightarrow f_1 \cap f_2 = \emptyset
\end{array}
$$

2.4 The Remove Operation

- Inputs:
  - file name;
  - user name.
- Pre-conditions:
  - the user is known to the system;
  - the file is owned by the user.
- Post-conditions:
  - number of users does not change;
  - ditto system users;
  - blocks formerly occupied by file are now free;
  - file is no longer owned by anyone.

Remove

$$
\begin{array}{l}
\Delta FileSystem \\
uname? : USERS \\
fname? : FILES \\
\hline
uname? \in system\_users \\
\\
fname? \in owns(uname?) \\
\\
system\_users' = system\_users \\
\\
no\_users'\_no\_users \\
\\
free\_blocks' = \\
\quad free\_blocks \cup occupies(fname?) \\
\\
occupies' = occupies \setminus \\
\quad \{fname? \mapsto occupies(fname?)\} \\
\\
owns' = \\
\quad (owns \setminus \{uname? \mapsto owns(uname?)\}) \\
\quad \cup \{uname? \mapsto \\
\qquad owns(uname?) \setminus \{fname?\}\}
\end{array}
$$

2.5 The Add (Create) Operation

- Inputs:
  - file name;
  - user name.
- Pre-conditions:
  - user is known to system;
  - file is not already owned.
- Post-condition:
  - the system users, no. of users, and free blocks remain invariant;
  - the user owns the file;
  - the file occupies no blocks (i.e, it is empty).

Add

$$\Delta FileSystem$$
$$uname? : USERS$$
$$fname? : FILES$$

$$fname? \notin owns(uname?)$$

$$uname? \in system\_users$$

$$system\_users' = system\_users$$

$$free\_blocks' = free\_blocks$$

$$no\_users' = no\_users$$

$$\forall u : \mathrm{dom}\, owns \bullet (u \neq uname?) \Rightarrow owns'(u) = owns(u)$$

$$owns'(uname?) = owns(uname?) \cup \{fname?\}$$

$$occupies' = occupies \cup \{fname? \mapsto \emptyset\}$$

3 Closing Remarks

- There are many ways we could extend the basic system; for example, let's think about access rights, in the UNIX sense.
- In UNIX, each file has access rights associated with three types of user:
  - user (the file owner);
  - group (the user category, e.g., staff or student);
  - other (everyone else).
- For each type of user, there are three possible access rights:
  - read (is able to read file);
  - write (is able to alter file);
  - execute (is able to execute file).
- For example, you might have all your files set so that you can read, write and execute them, your group can read and execute them, and others cannot read, write or execute them.

- We can model access rights by altering the *FileStore* schema. First, we define two new types:

$$UTYPES == \{u, g, o\}$$
$$RIGHTS == \{r, w, x\}$$

We then alter the *FileStore* schema thus ...

- New variable:

$$access\_rights : (FILES \times UTYPES) \pfun \mathbb{P}\, RIGHTS$$

- New invariant:

$$\mathrm{dom}\, access\_rights = (\mathrm{dom}\, occupies \times UTYPES)$$

- The new variable *access_rights* associates files and user types with access rights; for example, if

  $$access\_rights(myfile, g) = \{r, w\}$$

  then people in the same group as the owner of file *myfile* have read (r) and write (w) access rights to *myfile*.
- The new invariant means that the function *access_rights* is defined for all combinations of files known to the system, and user types.
- In other words, if *myfile* is a file known to the system, then *access_rights* is defined for:

  $access\_rights(myfile, u)$
  $access\_rights(myfile, g)$
  $access\_rights(myfile, o)$

- EXERCISES.

  Extend the simple specification given here to the full UNIX system.

  (See the article *Specification of the UNIX filing System* by Morgan & Sufrin, in *IEEE Transactions on Software Engineering*, Vol. SE-10 No. 2, 1984.)