SI Models & Inscape Examples

Dewayne E Perry
ENS 623A
Office Hours: T/Th 11:00-12:00
perry @ ece.utexas.edu
www.ece.utexas.edu/~perry/education/382v-s06/

File Operation Specs

Preconditions:	ValidFilePtr (FP)	R1
	FileOpen (FP)	R2
	LegalRecordNr (R)	R3
	RecordExists (R)	R4
	RecordReadable (R)	R5
	RecordConsistent (R)	R6
ReadRecord (FP, R, L, Bufptr)		
Postconditions:	ValidFilePtr (FP)	R7
	FileOpen (FP)	R8
	LegalRecordNr (R)	R9
	RecordExists (R)	R10
	Was (RecordReadable (R))	R11
	Was (RecordConsistent (R))	R12
	Allocated (Bufptr)	R13
	0 <= L <= Allocated (Bufptr)	R14
	RecordIn (Bufptr)	R15
Obligations:	Deallocated (Bufptr)	R16

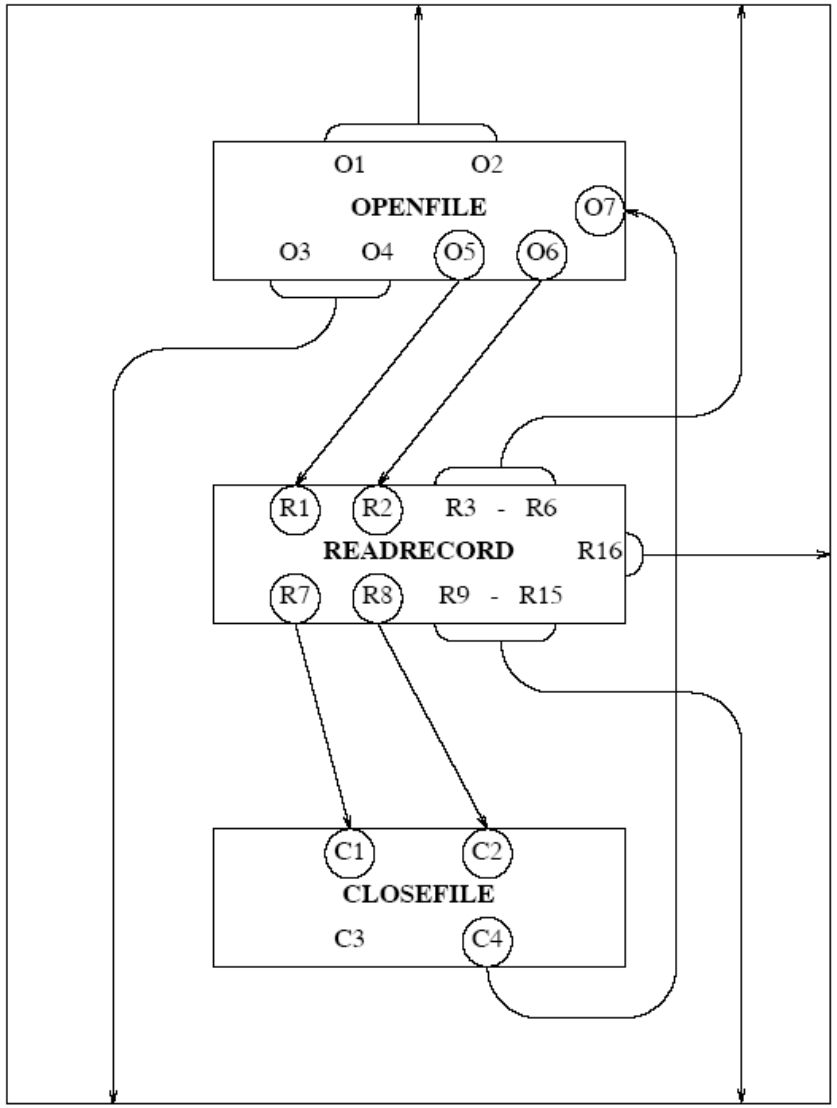
File Operation Specs

Preconditions:	LegalFileName (F)	O1
	FileExists (F)	O2
OpenFile (F, FP)		
Postconditions:	LegalFileName (F)	O3
	FileExists (F)	O4
	ValidFilePtr (FP)	O5
	FileOpen (FP)	O6
Obligations:	FileClosed(FP)	O7

File Operation Specs

Preconditions:	ValidFilePtr (FP)	C1
	FileOpen (FP)	C2
CloseFile (FP)		
Postconditions:	Not ValidFilePtr(FP)	C3
	FileClosed (FP)	C4
Obligations:	<none>	

Semantic Interconnections



Resulting Interface Spec

Preconditions:	LegalFileName (F)	O1
	FileExists (F)	O2
	LegalRecordNr (R)	R3
	RecordExists (R)	R4
	RecordReadable (R)	R5
	RecordConsistent (R)	R6
ObtainRecord (FP, R, L, Bufptr)		
Postconditions:	LegalFileName (F)	O3
	FileExists (F)	O4
	LegalRecordNr (R)	R9
	RecordExists (R)	R10
	Was (RecordReadable (R))	R11
	Was (RecordConsistent (R))	R12
	Allocated(Bufptr)	R13
	0 <= L <= Allocated (Bufptr)	R14
	RecordIn (Bufptr)	R15
Obligations:	Deallocated (Bufptr)	R16

File Mgmt – Predicate Specs

LegalFileName(filename F)
Definition: *NonNullString(F) and*
each (i in 1..length(F)) { Alphabetic(F[i]) }
Informally: A legal file name is a non-empty string of alphabetic characters only.

FileExists(filename F) . . .
ValidFilePtr(fileptr FP) . . .

FileOpen(fileptr FP)
Definition: *primitive*
Informally: The file is opened for reading and writing.

FileClosed(fileptr FP)
Definition: *not FileOpen(FP)*
Informally: The file is closed for I/O.

LegalRecordNr(int R) . . .
RecordExists(int R) . . .
RecordReadable(int R) . . .
RecordConsistent(int R) . . .
RecordWriteable(int R) . . .
RecordIn(buffer B) . . .
BufferSizeSufficient(buffer B; int R) . . .

File Mgmt – Data Specs

Type: `filename`
Representation: `string`
Properties: *each (filename F) { LegalFileName(F) }*
Synopsis: A filename is a non-empty string that is limited to alphabetic characters.

Type: `fileptr` . . .

File Mgmt – Operation Specs

`int CreateFile(<in> filename FN;<out> fileptr FP)`

Synopsis: `CreateFile` creates a file named by FN, automatically opens it, and returns a handle to be used in all subsequent file operations until the file is closed.

Preconditions:

LegalFileName(FN)	<validated>
not FileExists(FN)	<validated>

Results:

<Successful result: `CreateFile == 0`>

Synopsis: The file named by FN has been created and opened, and the output parameter FP is the handle for subsequent file operations.

Postconditions:

LegalFileName(FN)	*ValidFilePtr(FP)*
FileExists(FN)	*FileOpen(FP)*

Obligations: *FileClosed(FP)*

<Exception IllegalFilename: `CreateFile == 1`>

Synopsis: The file was not created because the file name in FN was invalid.

Failed: *LegalFileName(FN)*

Postconditions: *not LegalFileName(FN)*

Obligations: *<none>*

Recovery: Ensure that FN has an appropriate string.

<Exception FileAlreadyExists: `CreateFile == 2`>

. . .

File Mgmt – Operation Specs

```
int OpenFile(<in> filename FN; <out> fileptr FP)
...

void CloseFile(<inout> fileptr FP)
```

Synopsis: `CloseFile` closes the file and trashes the file pointer FP.

Preconditions:

ValidFilePtr(FP) <assumed>

FileOpen(FP) <assumed>

Results:

<Successful result: assumed>

Synopsis: Assumes the file is open and FP is a valid file pointer; the file is closed and FP is no longer valid.

Postconditions:

FileClosed(FP)

not ValidFilePtr(FP')

Obligations: <*none*>

```
int ReadRecord(<in> fileptr FP; <in> int R;
               <out> int L; <out> buffer B)
...
```

File Mgmt – Operation Specs

```
int WriteRecord(<in> fileptr FP; <in> int R;
                <in> int L; <inout> buffer B)
```

Synopsis: ...

Preconditions:

ValidFilePtr(FP)	<assumed>
FileOpen(FP)	<assumed>
LegalRecordNr(R)	<validated>
RecordIn(B)	<assumed>
Allocated(B)	<validated>
BufferSizeSufficient(B, L)	<validated>
RecordWriteable(R)	<dependent>

Results:

<Successful result: `WriteRecord == 0`>

Synopsis: The record R has been written in the file denoted by FP.

Postconditions:

LegalRecordNr(R)	*BufferSizeSufficient(B,L)*
Deallocated(B)	*RecordExists(R)*

Obligations: <*none*>

File Mgmt – Operation Specs

<Exception IllegalRecordNr: `WriteRecord == 1`>
Synopsis: An invalid record number.
Failed: *LegalRecordNumber*
Postconditions: *not LegalRecordNr*
Obligations: <*none*>
Recovery: . . .

<Exception UnallocatedBuffer: `WriteRecord == 2`>
. . .
Postconditions: *not Allocated(B)*
. . .

<Exception BufferTooSmall: `WriteRecord == 3`>
. . .

<Exception WriteError: `WriteRecord == 4`>
Synopsis: The record has not been written due to an I/O error.
Failed: *RecordWriteable(R)*
Postconditions:

LegalRecordNr(R)	*BufferSizeSufficient(B, L)*
Allocated(B)	*not RecordWriteable(R)*

Obligations: <*none*>
Recovery: `RecoverFileSystem`

File Mgmt – Operation Specs

`boolean FileExists(<in> filename FN) ...`

Synopsis: `FileExists` determines whether the file named by FN exists.

Preconditions:

LegalFileName(FN) <assumed>

Results:

<Successful result: `FileExists == TRUE`>

Synopsis: The file exists.

Postconditions: *FileExists(FN)*

Obligations: <*none*>

<Successful result: `FileExists == FALSE`>

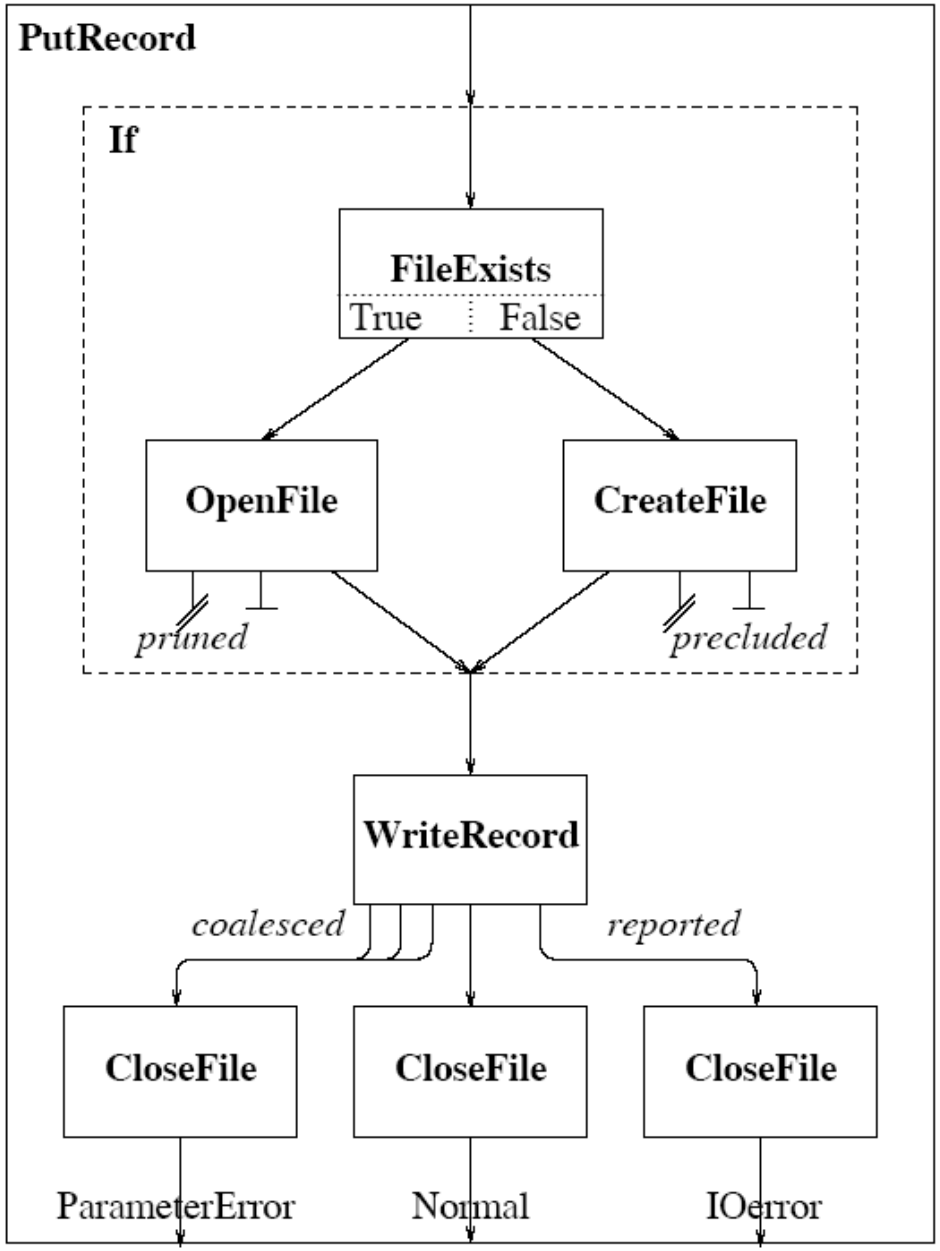
Synopsis: The file does not exist.

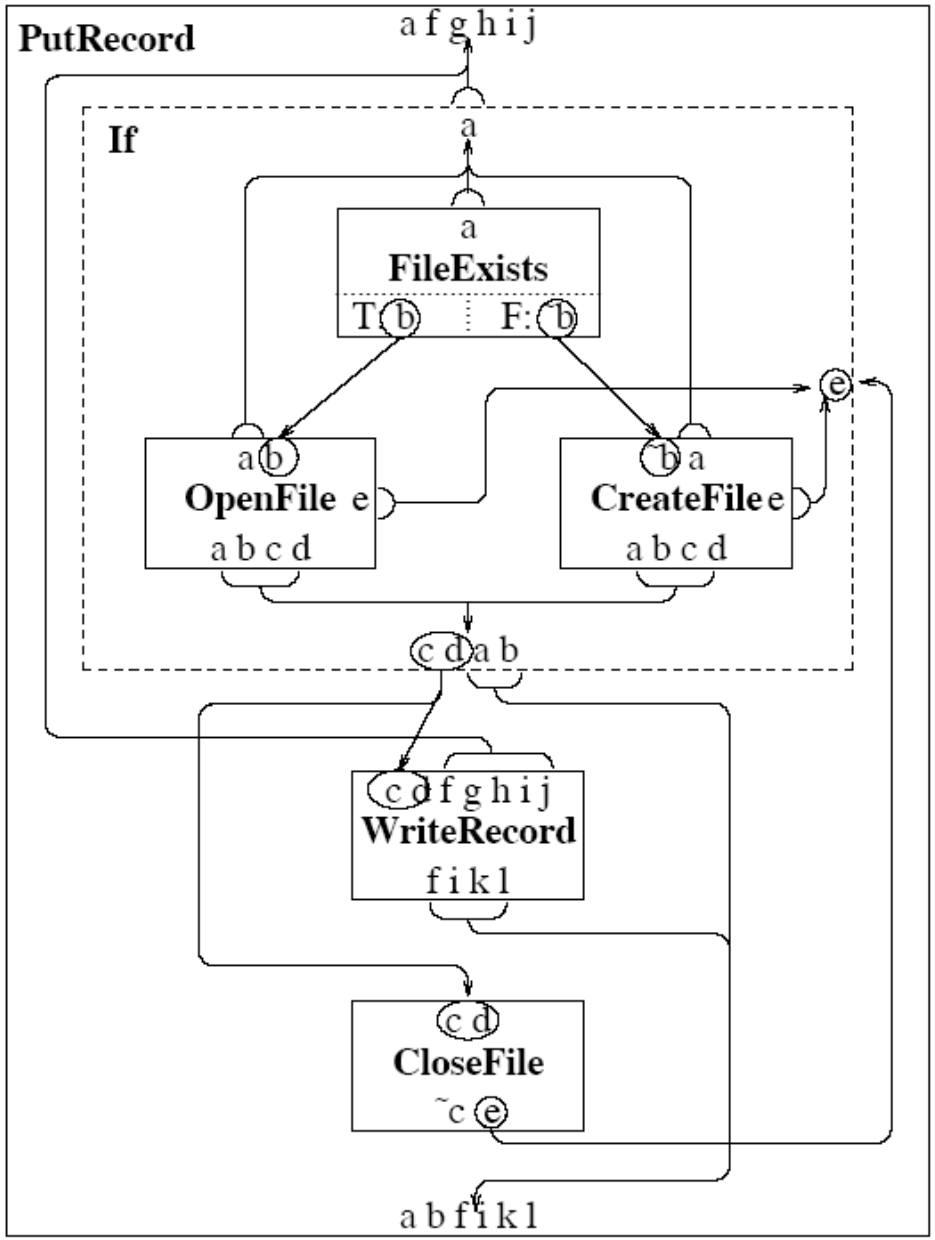
Postconditions: *not FileExists(FN)*

Obligations: <*none*>

Implementation of PutRecord

```
PutRecord(<in> filename FN; <in> int R;
          <in> int L; <inout> buffer B);
   {
   fileptr FP;
   if FileExists(FN)
      OpenFile(FN, FP);
         <exception IllegalFileName pruned>
         <exception NonExistentFile precluded>
   else
      CreateFile(FN, FP);
         <exception IllegalFileName pruned>
         <exception FileAlreadyExists precluded>
   WriteRecord(FP, R, L, B);
      exception IllegalRecordNr      <coalesced>
      exception UnallocatedBuffer    <coalesced>
      exception BufferTooSmall       <coalesced>
         CloseFile(FP);
         return ParameterError;      <propagated>
      exception WriteError
         CloseFile(FP);
         return IOerror;             <propagated>
   CloseFile(FP);
   }
```


PutRecord
If
FileExists
True
False
OpenFile
CreateFile
primed
precluded
WriteRecord
coalesced
reported
CloseFile
CloseFile
CloseFile
ParameterError
Normal
IOerror


PutRecord
a f g h i j
If
a
a
FileExists
T: b
F: ~b
e
a b
OpenFile e
a b c d
~b a
CreateFile e
a b c d
c d a b
c d f g h i j
WriteRecord
f i k l
c d
CloseFile
~c e
a b f i k l

Propagated Interface for PutRecord

Propagated Preconditions:

LegalFileName(FN) *Allocated(B)*
LegalRecordNr(R) *BufferSizeSufficient(B, L)*
RecordIn(B)

Propagated Results:

<normal exit>

Propagated Postconditions:

LegalFileName(FN) *BufferSizeSufficient(B, L)*
FileExists(FN) *Deallocated(B)*
LegalRecordNr(R) *RecordExists(R)*

Propagated Obligations: *<none>*

<exception ParameterError>

Propagated Postconditions:

FileExists(FN)
not IllegalRecordNr(R) or not Allocated(B)
or not BufferSizeSufficient(B, L)

Propagated Obligations: *<none>*

<exception IOerror>

Propagated Postconditions:

LegalFileName(FN) *Deallocated(B)*
FileExists(FN) *BufferSizeSufficient(B, L)*
LegalRecordNr(R) *not RecordWriteable(R)*

Propagated Obligations: *<none>*