Honest Deputy

James thinks the answer to the compiler as confused deputy problem is relatively straghtforward. Given a spec for the compiler, something like:

```
forall s : Object, calls compiler.compile(inName,outName)  ->
  Next[ FileContents( outName ) == Compile( FileContents( intName )) ]
```

and for a billable service

```
forall s : Object, s calls billable(_) ->
   Next( FileContents ( BILLING ) == Prev(FileContents(BILLING)) ++ ThisBill )
```

or the Hoare logic verions, being a bit more picky:

```
     ExistingFile(inName) && ValidFileName(outName)
 { compiler.compile(inName,outName) }
     FileContents( outName ) == Compile( FileContents( intName ))

   forall c = FileContents(BILLING)
 { compiler.compile(inName,outName) }
   FileContents (BILLING) == c ++ ThisBill
```

the point is that composing those specs together must lead to an unstatisfiable specifiaction, because if you call the compiler with outName=BILLING, then the spec requlres both billing data and compiled file contents to be in the BILLING file. To make it satisfiable, you need to add in a precondition e.g. that inFile != Billing...

We could also bound the authority of the deputy - this will manage the risk, but doesn't stop the classical confusion.

```
  forall d : Deputy; forall f : File;
     d accesses f ->
       (f = BILLING) ||
         (Was( exists o : Object. o calls d(_) && o accesses f))
```