I've been thinking about coinop things and money in general. Boy have I! I decided to look at the entire way we handle money. Here are the protocols that we currently support on the host:

```
TELEPORT:
        request: <teleport> <PAY>

          r_msg: <teleport> <PAY> <FAIL>
                --or--
          r_msg: <teleport> <PAY> <SUCCESS>
          n_msg: <teleport> <PAY$>


FORTUNE MACHINE:
        request: <fortune machine> <PAY>

          r_msg: <fortune machine> <PAY> <FAIL>
                --or--
          r_msg: <fortune machine> <PAY> <SUCCESS> <text>
          n_msg: <fortune machine> <SPEAKFORTUNE$> <text>


JUKEBOX:
        request: <jukebox> <PAY>

          r_msg: <jukebox> <PAY> <FAIL>
                --or--
          r_msg: <jukebox> <PAY> <SUCCESS>


VENDO:
        request: <vendo> <PAY>

          r_msg: <vendo> <PAY> <FAIL>
                --or--
          r_msg: <vendo> <PAY> <SUCCESS> <obvec>
          n_msg: <vendo> <SELL$> <whonoid> <obvec>


ATM:
        request: <atm> <DEPOSIT> <tokenid>

          r_msg: none
          n_msg: <region> <GOAWAY_$> <tokenid>


        request: <atm> <WITHDRAW> <amount>

          r_msg: <atm> <WITHDRAW> <actual amount>
                --or--
          r_msg: <atm> <WITHDRAW> <actual amount>
          n_msg: <region> <HEREIS_$> <obvec>


TOKENS:
        request: <token> <PAY> <whotarget> <amount>

          r_msg: <token> <PAY> <FAIL>
                --or--
          r_msg: <token> <PAY> <SUCCESS> <amount>
          n_msg: <whotarget> <PAID$> <amount>
                --or--
          r_msg: <token> <PAY> <SUCCESS> <amount> <obvec>
          n_msg: <whotarget> <PAID$> <amount> <obvect>
                --or--
          r_msg: <token> <PAY> <SUCCESS> <amount> <obvec>
          n_msg: <region> <HEREIS_$> <obvec>
```

```
CREDIT CARD:
        request: <card> <PAYTO> <whotarget> <amount>

          r_msg: <card> <FAIL>
                --or--
          r_msg: <card> <SUCCESS>
          p_msg: <whotarget> <PAYTO$> <amount>
```

If you look you'll see that there's a pattern, but it isn't followed consistently. I'd like to adopt the following model for coin operated devices then:

```
        request: <obj> <PAY>

          r_msg: <obj> <PAY> <FAIL>                     ; for failure
                --or--
          r_msg: <obj> <PAY> <SUCCESS> <amount> <params>; for success
          n_msg: <obj> <PAY$> <whonoid> <amount> <params>
```

The host passes back the amount paid. This lets us change the cost of coin operated devices from the host. In the asynchronous case, the host also tells who did the paying. We do *not* send a GOAWAY message deleting depleted tokens. Instead, we keep each token's denomination correctly and delete the token when it reaches 0. I realized that we MUST keep each token's denomination correctly, not just the ones you are holding, since someone could put a token down on the ground and then you could pick it up, thus transferring the token from somebody else to you without ever requiring the host to inform you of the correct denomination...

<params> refers to any additional parameters that the host behavior might wish to pass back, such as the <text> provided by the fortune machine or the <obvec> provided by the vendo. Tacking these on the end allows a single uniform routine to handle every coinop machine's monetary transaction part and then lets a special behavior pick off the rest for class-dependent processing.

Also, ATM <WITHDRAW> needs an additional asynchronous behavior to handle the case where money is withdrawn but a new token object does not need to be created (this goes along with keeping each token's denomination correct).

Oddly enough, the protocol for token object itself seems to be correct already. Credit card is also OK, since it only deals with the bank account. The interface for using the credit card to pay for things is a little messy though and needs further thought.

And one other thing: we will need to go through and make sure all the message numbers are assigned consistently. Fortunately, we don't need to have the same asynchronous message number across objects, since each asynchronous behavior is looked up independently.

Whew! Does this all make sense?