```
// OfferServer.cpp - OFFER SERVER DEFINITION
//
// MODULE INDEX
// NAME                          CONTENTS
// OfferServer_c::Initiate       Initiate the offer server
// OfferServer_c::Tick           Process a clock tick
// OfferServer_c::ReplyReceived  Process a reply received from an atm
// OfferServer_c::AnswerReceived Process an answer received from a bcs
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 23-09-12 JS  Original
//
//-----------------------------------------------------------------------

#include <ctime>          // C-style system time functions
#include <cstdlib>        // C-style standard library
#include <cstring>        // C-style string manipulation functions
#include <iostream>       // C++ I/O stream declarations
#include <string>         // C++ string declarations
#include <map>            // C++ map declarations
#include <list>           // C++ list declarations
using namespace std;          // Expand the standard namespace
#include "OfferComms.h"       // Offer communications declarations
#include "OfferServer.h"      // Offer server declarations

//-----------------------------------------------------------------------

// DEFINITIONS

#define RETRY_PERIOD     30   // Retry period in seconds

//-----------------------------------------------------------------------

// INITIATE THE OFFER SERVER

void
OfferServer_c::Initiate (
    OfferComms_c     *xOfferComms,     // Offer communications class instance
    const Offer_t    *offerArr,   // Offer array
    size_t       offerCnt)    // Number of offers
{
    size_t       i;       // General purpose index
    OfferData_t offerData;   // Offer data structure

    // Save the pointer to the communications class

    offerComms = xOfferComms;

    // Load the offer map

    offerMap.clear ();
    for (i = 0; i < offerCnt; i++) {
        offerData.bcsId = offerArr[i].bcsId;
        offerData.offerState = OFFER_STATE_OPEN;
        offerMap [ OfferKey_t (offerArr[i].offerCode,
            offerArr[i].customerId)] = offerData;
```

```
    }

    // Reset the retry map

    retryMap.clear ();
}

//---------------------------------------------------------------------

// PROCESS A CLOCK TICK

void
OfferServer_c::Tick ()
{
    time_t      currentTime;     // Current time
    RetryIter_t retryIter;   // Retry iterator
    OfferIter_t offerIter;   // Offer iterator

    // Look up the current time

    currentTime = time(0);

    // Process outstanding retries

    for (
        retryIter = retryMap.begin();
        retryIter != retryMap.end();
        retryIter ++
    ) {
        if (retryIter->second.retryTime <= currentTime) {

            // Look up the offer iterator

            offerIter = offerMap.find (retryIter->second.offerKey);
            if (offerIter == offerMap.end()) {
                cerr << "Error: bad offerKey\n";
                abort ();
            }

            // Resend the reply

            offerComms->SendToBCS (offerIter->second.bcsId,
                offerIter->second.atmId,
                offerIter->first.offerCode,
                offerIter->first.customerId,
                offerIter->second.offerAccepted);

            // Update the retry time

            retryIter->second.retryTime = currentTime+RETRY_PERIOD;
        }
    }
}

//---------------------------------------------------------------------

// PROCESS A REPLY RECEIVED FROM AN ATM
```

```
void
OfferServer_c::ReplyReceived (
    long        atmId,       // ATM identifier
    long        offerCode,   // Offer code
    long        customerId,  // Customer identifier
    bool        offerAccepted)   // Offer accepted flag
{
    OfferIter_t offerIter;   // Offer iterator
    RetryIter_t retryIter;   // Retry iterator
    RetryData_t retryData;   // Retry data structure

    // Look up the offer in the offer map

    offerIter = offerMap.find (OfferKey_t (offerCode, customerId));

    // If the offer does not exist, reject the reply

    if (offerIter == offerMap.end()) {
        offerComms->SendToATM (atmId, offerCode, customerId,
            0, "Invalid offer code or customer identifier");
        return;
    }

    // Process new replies

    if (offerIter->second.offerState == OFFER_STATE_OPEN) {

        // Load the reply information into the offer map

        offerIter->second.atmId = atmId;
        offerIter->second.offerAccepted = offerAccepted;

        // If the BCS is active, queue the reply to the BCS

        retryIter = retryMap.find (offerIter->second.bcsId);
        if (retryIter != retryMap.end()) {

            // Update the offer status

            offerIter->second.offerState = OFFER_STATE_QUEUED;

            // Append the reply to the reply queue

            retryIter->second.replyQueue.push_back (
                offerIter->first);
        }

        // If the BCS is active, queue the reply to the BCS

        else {
            // Update the offer status

            offerIter->second.offerState = OFFER_STATE_SENT;

            // Load the retry table entry
```

```
            retryData.retryTime = time(0) + RETRY_PERIOD;
            retryData.offerKey = offerIter->first;
            retryData.replyQueue.clear ();
            retryMap[offerIter->second.bcsId] = retryData;

            // Relay the reply

            offerComms->SendToBCS (offerIter->second.bcsId,
                atmId, offerCode, customerId, offerAccepted);
        }
    }

    // Process repeated replies

    else {
        // Check that the reply is from the same ATM

        if (atmId != offerIter->second.atmId) {
            offerComms->SendToATM (atmId, offerCode, customerId,
                0, "Reply already processed");
            return;
        }

        // Check that the reply is the same

        if (offerAccepted != offerIter->second.offerAccepted) {
            offerComms->SendToATM (atmId, offerCode, customerId,
                0, "Reply different");
            return;
        }

        // If the reply has been processed, retransmit the
        // previous answer

        if (offerIter->second.offerState == OFFER_STATE_ANSWERED) {
            offerComms->SendToATM (atmId, offerCode, customerId,
                offerIter->second.replyAccepted,
                offerIter->second.reason.c_str());
        }

        // If the original reply is still being processed, quietly
        // ignore the repeated reply
    }
}

//---------------------------------------------------------------------------

// PROCESS AN ANSWER RECEIVED FROM A BCS

void
OfferServer_c::AnswerReceived (
    long        bcsId,       // BCS identifier
    long        offerCode,   // Offer code
    long        customerId,  // Customer identifier
    bool        replyAccepted,   // Reply accepted flag
    const char  *reason)     // Reason reply was rejected
{
```

```
    OfferIter_t offerIter;   // Offer iterator
    RetryIter_t retryIter;   // Retry iterator

    // Look up the offer in the offer map

    offerIter = offerMap.find (OfferKey_t (offerCode, customerId));

    // If the offer does not exist, log the anomaly

    if (offerIter == offerMap.end()) {
        cerr << "Warning: invalid offer code or customer id\n";
        return;
    }

    // Check that the reply is being sent by the correct
    // business customer's server

    if (bcsId != offerIter->second.bcsId) {
        cerr << "Warning: wrong BCS id\n";
        return;
    }

    // Process the answer depending on the offer state

    switch (offerIter->second.offerState) {

    // Open offers

    case OFFER_STATE_OPEN:
        cerr << "Warning: offer is still open\n";
        return;

    // Queued replies

    case OFFER_STATE_QUEUED:
        cerr << "Warning: reply is still queued\n";
        return;

    // Sent replies

    case OFFER_STATE_SENT:

        // Save the answer

        offerIter->second.replyAccepted = replyAccepted;
        if ( ! replyAccepted)
            offerIter->second.reason = reason;
        else
            offerIter->second.reason = "";

        // Update the offer state

        offerIter->second.offerState = OFFER_STATE_ANSWERED;

        // Relay the answer to the ATM

        offerComms->SendToATM (offerIter->second.atmId,
```

```
            offerCode, customerId, replyAccepted, reason);

        // If another reply is queued, relay the queued reply

        retryIter = retryMap.find (bcsId);
        if (retryIter == retryMap.end()) {
            cerr << "Error: missing retry map entry\n";
            abort ();
        }
        if (retryIter->second.replyQueue.size() != 0) {

            // Update the retry map

            retryIter->second.offerKey =
                retryIter->second.replyQueue.front();
            retryIter->second.retryTime =
                time(0) + RETRY_PERIOD;
            retryIter->second.replyQueue.pop_front();

            // Look up the offer iterator

            offerIter = offerMap.find (retryIter->second.offerKey);
            if (offerIter == offerMap.end()) {
                cerr << "Error: bad offerKey\n";
                abort ();
            }

            // Update the offer state

            offerIter->second.offerState = OFFER_STATE_SENT;

            // Send the reply

            offerComms->SendToBCS (offerIter->second.bcsId,
                offerIter->second.atmId,
                offerIter->first.offerCode,
                offerIter->first.customerId,
                offerIter->second.offerAccepted);
        }

        // If no replies are queued, drop the retry map entry

        else {
            retryMap.erase (retryIter);
        }
        break;

    // Answered replies

    case OFFER_STATE_ANSWERED:

        // Validate the answer

        if (replyAccepted != offerIter->second.replyAccepted) {
            cerr << "Warning: answer changed\n";
            return;
        }
```

```
        if ( ! replyAccepted && offerIter->second.reason != reason) {
            cerr << "Warning: reason changed\n";
            return;
        }

        // Quietly ignore the repeated answer

        break;

    // Other states are anomalous

    default:
        cerr << "Error: bad state\n";
        abort ();
        // NOTREACHED
    }
}
```