FiTS: A Financial Transaction System

Gordon J. Pace, Christian Colombo

June 2013

Abstract

FiTS is a cut-down version of a financial transaction system, aimed at providing basic functionality to enable students to experiment for testing and verifying its behaviour. The system has been kept reasonably simple to ensure that the focus is on the verification techniques and not on understanding the underlying system. In this document, we present basic documentation to facilitate the use of the system.

1 Overview of FiTS

FiTS (A Financial Transaction System) is a bare-bones system mocking the behaviour of a financial transaction system. It emulates a virtual banking system in which users may open accounts from which they may perform money transfers. The following terms are defined so as to aid understanding of the rest of the documentation:

Administrator: The administrators of FiTS have more rights tha normal clients to be able to perform certain actions such as approving the opening of an account, enabling a new user and registering new users.

User: In FiTS, clients or users have access to a number of actions to access their (money) accounts which they have registered on FiTS — actions which they may invoke through an online interface. Information about each registered user is stored in a database, including information such as the client's name and country of origin, and user classification information:

- **User type:** Clients fall under three types or categories: Gold, Silver or Normal. The user type determines charges which he or she will incur when transfering funds using FiTS.
- **User status:** Each user is either white- or blacklisted. The latter may be stopped from making certain money transfers.
- **User mode:** A user may be in one of a number of modes: (i) *activated* users are allowed to log in and make transfers; (ii) users start off as *disactivated* (before being activated by the administrator) and may also end up in this mode if they are discovered to have engaged in

fraudulent behaviour; (iii) users may temporarily freeze their user account — *frozen* users do not allow money transfers to be made, thus ensuring better security for the user if they do not plan to use FiTS for a while. At any point, users may reactivate a frozen account.

Account: Each user account may be associated with a number of money accounts belonging to him or her. Users may request the creation of a new account or close down one of their accounts at any point in time.

Session: An existing user may open a login session on FiTS to make transfers or manage their money accounts. A user may have multiple sessions open at the same time.

2 The Modules

The FiTS code provided primarily consists of the backend code which handles the user and accounts information database behind the system. From this backend, only certain methods can be accessed by the front-ends of the system which the administrator and the user have access to. Although the front-end typically ensures that the behaviour of the users is limited (e.g. by not giving a blacklisted user the option to make a transfer), the backend was built with the intension of double checking that such violations do not occur.

Below are descriptions of the main FiTS modules and the method calls which may be on interest to understand how the system works:

- `Interface:` The FiTS Interface

  This module contains the only methods over which the user interface for the administrators (or automated backend tools) and the clients may interact with the backend. The methods start with either `ADMIN` or `USER` to indicate who can access them[1].

  The methods which the administrators may execute are:

  - `void ADMIN_initialise()`
    Initialise the transaction system.
  - `void ADMIN_reconcile()`
    Reconcile the accounts to ensure that the overall balances are correct.
  - `Integer ADMIN_createUser(String name, String country)`
    Create a new user with the given name (`name`) and country of residence (`country`), returning the new user's id.
  - `void ADMIN_disableUser(Integer uid)`
    `void ADMIN_activateUser(Integer uid)`
    Activate or disable a user with the given user id (`uid`).

[1] Ideally, these methods are separated in two separate classes, but given the aims of FiTS, they are kept in a single module for simplicity.

- `void ADMIN_blacklistUser(Integer uid)`
  `void ADMIN_greylistUser(Integer uid)`
  `void ADMIN_whitelistUser(Integer uid)`
  Black/grey or whitelist the user with the given id (`uid`).
- `void ADMIN_makeGoldUser(Integer uid)`
  `void ADMIN_makeSilverUser(Integer uid)`
  `void ADMIN_makeNormalUser(Integer uid)`
  Make the user with the given id (`uid`) a Gold, Silver or Normal user.
- `void ADMIN_approveOpenAccount(Integer uid, String account_number)`
  Approve the opening of account with the given number (`account_number`) owned by the user with the given id (`uid`).

The methods which the users may execute are the following. All the methods include the user id of the user executing the command (`uid`).

- `Integer USER_login(Integer uid)`
  `void USER_logout(Integer uid, Integer sid)`
  Login the user into the system, returning the new session's id and logout the user of the session with the given id (`sid`).
- `boolean USER_freezeUserAccount(Integer uid)`
  `boolean USER_unfreezeUserAccount(Integer uid, Integer sid)`
  Freeze and unfreeze the user with id `uid` in session with id `sid`, returning whether the action was successful.
- `String USER_requestOpenAccount(Integer uid, Integer sid)`
  Request the opening of a new money account owned by the user with id `uid` in session with id `sid`, returning the account number that will be assigned if approved.
- `void USER_closeAccount(Integer uid, Integer sid, String account_number)`
  Close an existing money account owned by the user `uid`, requested from session `sid`.
- `void USER_depositFromExternal`
  `(Integer uid, Integer sid, String to_account_number, double amount)`
  Deposit money (`amount`) from an external source (e.g. from a credit card) to an account owned by the user `uid` (with account number `to_account_number`), and from session with id `sid`.
- `boolean USER_payToExternal`
  `(Integer uid, Integer sid, String from_account_number, double amount)`
  Send money (`amount`) to an external money account (e.g. pay a bill) from the account with the given account number (`from_account_number`) — charges are applied and also taken from the user's account. The method returns whether the transfer was successful.

- `boolean USER_transferToOtherAccount (Integer from_uid, String from_account_number, Integer to_uid, String to_account_number, double amount)`
  Transfer money (`amount`) from the user's account (`from_account_number`) to another user's (`to_uid`) account (`to_account_number`) — charges are applied and also taken from the sender's account. The method returns whether the transfer was successful.
- `boolean USER_transferOwnAccounts (Integer uid, String from_account_number, String to_account_number, double amount)`
  Transfer money (`amount`) across accounts owned by the same user (from account with number `from_account_number` to `from_account_number`) — charges do not apply in this case. The method returns whether the transfer was successful.

- `TransactionSystem`: The Transaction System Model

  The underlying model of FiTS is given in this module. The main methods of interest in this module are:

  - `void initialise()`
    Initialises the transaction system. For the sake of verification, the system is started in a predetermined initial state with a number of users and accounts.
  - `Integer addUser(String name, String country)`
    Creates a new user to the database with the given name (`name`) and country of residence (`country`), returning the new user's id.

  Other methods are defined and documented in the module, but these are the only ones relevant to the verification process.

  Also note the difference between the method `addUser` (which adds the user to the database) and the method `ADMIN_createUser` in the module `TransactionSystem` (which is the administrator's request to create the user). Similar distinctions will be seen in other methods.

- `UserInfo`: The Users' Information Database

  The `UserInfo` object stores the state of a particular user. Verification-relevant methods are the following:

  - `void makeGoldUser()`
    `void makeSilverUser()`
    `void makeNormalUser()`
    Sets the user's type to Gold, Silver or Normal accordingly.
  - `void blacklist()`
    `void greylist()`
    `void whitelist()`
    Sets the user's status to blacklisted or whitelisted respectively.

- `void makeActive()`
  `void makeFrozen()`
  `void makeDisabled()`
  Sets the user's mode to active, frozen or disabled respectively.
- `Integer openSession()`
  Open a new session for the user, returning the session id.
- `void closeSession(Integer sid)`
  Close the session with the given session id (`sid`) owned by the current user.
- `String createAccount()`
  Create a money account owned by the user, returning the account number.
- `void deleteAccount(String account_number)`
  Delete the money account with the given account number (`account_number`).

- `UserAccount`: Money Accounts

  The `UserAccount` object stores the state of a particular user's money account. Verification-relevant methods are the following:

  - `void withdraw(double amount)`
    `void deposit(double amount)`
    Respectively withdraw and deposit the requested sum (`amount`) from and to the account.
  - `Integer getOwner()`
    Returns the user id of the owner of the account.

- `UserSession`: Login Sessions The `UserSession` object stores the state of a particular user's login session and a log with the actions carried out in that session. Verification-relevant methods are the following:

  - `void initialiseSession()`
    `void closedownSession()`
    Methods to initialise and close a session.
  - `void log(String l)`
    Add the given line to the log of the session.
  - `String getLog()`
    Returns the log of the session as a string.

3 The Scenarios and their Execution

The `Scenarios` module provides test cases for driving FiTS. The module comes with two test cases for each property (see Section 4), one violating the property and one not. The two main methods provided are:

`static void runViolatingScenarioForProperty(Integer n)`
`static void runNonViolatingScenarioForProperty(Integer n)`

These methods allow the running of individual tests to see the results. Information about which test is being run is displayed in the console.

In most cases, one would be interested in running all test cases, using the method: `static void runAllScenarios()`. A main class and method have been provided to do exactly this.

4 Correctness Properties of FiTS

If correctly implemented, FiTS is meant to satisfy the following properties through the implementation of the front end (which functionality is provided to which users) and the back end (ensuring that undesired behaviour is not allowed through).

1. Only users based in Argentina can be Gold users.
2. The transaction system must be initialised before any user logs in.
3. No account may end up with a negative balance after being accessed.
4. An account approved by the administrator may not have the same account number as any other already existing account in the system.
5. Once a user is disabled, he or she may not withdraw from an account until being made activate again.
6. Once greylisted, a user must perform at least three incoming transfers before being whitelisted.
7. No user may not request more than 10 new accounts in a single session.
8. The administrator must reconcile accounts every 1000 attempted external money transfers or an aggregate total of one million dollars in attempted external transfers (attempted transfers include transfers requested which never took place due to lack of funds).
9. A user may not have more than 3 active sessions at once.
10. Logging can only be made to an active session (i.e. between a login and a logout).

In fact, as built, FiTS does not satisfy these properties, so as to enable us to validate our verification tools on it.

5 Summary

FiTS is intended to provide a sufficiently realistic and complex transactions managment system to illustrate how verification of complex properties can be carried out. On the other hand, the system has been kept reasonably small so as to ensure that not too much effort is required to be able to understand its inner workings, thus enabling verification instrumentation.