Christian Colombo
(joint course design with Gordon J. Pace)
University of Malta

March 2018

RUNTIME VERIFICATION
FROM THEORY TO PRACTICE AND BACK

Manual Monitoring

FiTS: A Financial Transaction System

- A bare-bones system mocking the behaviour of a financial transaction system.
- Emulates a virtual banking system in which users may open login session and money accounts from which they may perform transfers.

FiTS: A Financial Transaction System

- **Administrator:** Can perform certain actions such as approving the opening of an account, enabling a new user and registering new users.
- **Users:** The normal clients who may open bank accounts and make transfers across them.
  - **Status:** White-/Grey-/Blacklisted
  - **Type:** Gold/Silver/Normal users
  - **Mode:** Active/Disabled/Frozen

FiTS: A Financial Transaction System

- **Session:** A login session associated with a user during which there is user interaction with the system.
- **Account:** Users may have multiple accounts in which money may be stored and transferred to and from other accounts and external sources.

FiTS: A Financial Transaction System

- Main modules:
  - Interface
  - TransactionSystem
  - UserInfo
  - UserAccount
  - UserSession

Properties

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 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).

Some Notes: Assertions

- Assertions in Java cause an exception which will need to be caught and complicate the logic. For this exercise, we will use a simplified inbuilt assertion manager in the static class `Verification`:
  - `Verification.assertion(condition, string);`
- If the condition is not satisfied, the string will be printed to the standard output.

Some Notes: Verifier State (1)

- Some properties may require the use of some memory, for instance, to remember whether something happened or not.
- The easiest approach is to add such a state as static fields to the `Verification` class.

Some Notes: Verifier State (2)

- To check that “A happens before B”, add a Boolean variable:

```
public class Verification {
        ...
        private static boolean A_happened;
        public static void initialise() {
                A_happened = false;
        }
        public static void eventA() { A_happened = true; }
        public static void pastA() { return A_happened; }
}
```

Some Notes: Verifier State (2)

- And refer to the state in the code:

```
... A(...) {
  Verification.eventA();
  ...
}

... B(...) {
  Verification.assert(Verification.pastA(), “ooops”);
  ...
}
```

Some Notes: Verifier State (2)

- Initialisation is used to enable running multiple scenarios in a single run:

```
public static void runAllScenarios() {
  for (Integer n=1; n<=SCENARIO_COUNT; n++) {
    restartTransactionSystem();
    Verification.initialiseVerification();
    scenario(n);
  }
}
```

Some Notes: Verifier State (3)

- When verifying a property which should be true for every user (or session or account) we can either:
  - Create a `Verification` object encapsulated in each `UserInfo` object (respectively `UserSession` or `UserAccount`); or
  - Create a global Verification object which has all the information within it (e.g. using `HashMap`).

Exercises

Add assertions to the code to check a number of properties of FiTS.

Exercises

Add assertions to the code to chec
of properti

Run the scenarios you were given with the code to check that they run as expected.

A Solution of Property 5

Once a user is disabled, he or she may not withdraw from an account until being made activate again.

A Solution of Property 5

Verification.java

```
public class Verification {
  public static Set<Integer> disabledButNotEnabled;
  ...
  static public void initialiseVerification()
  {
      disabledButNotEnabled = new Set<Integer>();
  }
}
```

A Solution of Property 5

Interface.java

```
public void ADMIN_activateUser(Integer uid)
{
  Verification.disabledButNotEnabled.remove(uid);
  ...
}

public void ADMIN_disableUser(Integer uid)
{
  Verification.disabledButNotEnabled.add(uid);
  ...
}
```

A Solution of Property 5

UserInfo.java

```
public void withdrawFrom(String account_number, double amount)
{
  Verification.assertion(
    !Verification.disabledButNotEnabled.contains(uid),
    “Property 5 violated”
  );
  ...
}
```

Some Observations

- Adding the properties into the system code makes it difficult to separate: where does the property end and the system start.
- Some properties are not simply assertions, and require additional logic – the code implementing this logic is also mixed with the system.
- Changes to the properties result in direct changes in the project code.
- If we want to change the mode of verification (e.g. produce logs to check offline), it will require reengineering the whole effort.