COMP-667 Software Fault Tolerance

Assignment 3: Transactions

(20% of Final Grade)

Problem Statement

Write an implementation of a transaction library providing the atomicity and isolation properties of transactions. Your library should use the following implementation strategies / provide the following features:

- The library must support semantic concurrency control, i.e. it should be possible for the user to define conflict information (forward commutativity) for all operations of a transactional object
- Isolation is provided using deferred update (meaning that for each transactional object that is accessed by a transaction, a separate "local" copy is made)
- Atomicity (in-memory recovery) is provided using deferred update (meaning that only when a transction commits, the operations executed on the local copy of the transaction object are applied to the main copy)
- Serializability is ensured by using optimistic concurrency control with forward validation using broadcast commit (i.e. a validating transaction aborts all active transactions that have executed conflicting operations)

The library does not need to support:

- Nested transactions
- Pre-emptive transaction abortion

The idea is that the implementation should be in the form of a reusable "library" (e.g. object / package / template / generic / framework / aspect).

To show that your library works and to allow us to test your library, write a small "banking" sample application that makes use of your library. The idea is to provide a transactional *BankAccount* class that defines the operations *withdraw*, *deposit* and *getBalance*. Your main application should use transactions to:

- Transfer a given *amount* of money from account *source* to account *dest*

- Withdraw a given *amount* of money from account *source* if and only if account *cond* has at least *minimum* amount of money in it
- Query the account balances of a set of accounts

In addition to the functionality mentioned above, there should be a way for us to "artificially" specify the commit order of the transactions so we can test your solution for correctness.

Grading

The grade will be based on:

- **Correctness**:
  - Does your infrastructure correctly implement atomicity and isolation?
    - Is the semantic based concurrecy control implemented correctly?
    - Is optimistic forward validation implemented correctly?
    - Are the state changes made by committed transactions correctly applied to the transactional objects?
- **Ease of use**:
  - Does your infrastructure have a simple, yet elegant interface?
  - Can the programmer use your infrastructure without having to do an excessive amount of programming on his own?
  - Are the design decisions reasonable (and documented)?

- **Safety**:
  - Does your infrastructure prevent a programmer from making mistakes when using it?
  - Are programming conventions verified (statically / at run-time)?
  - Is your infrastructure thread-safe?
- **Sample Application**:
  - Is there a clear separation of code between the application and the library?
  - Does the sample application provide the required functionality?
  - Does the sample application allow the graders to specify the transaction commit order?

Hand-In

Please hand in your solution before the end of Wednesday March 21st! Send an email to Joerg.Kienzle@mcgill.ca and Wisam.AlAbed@mail.mcgill.ca with the title "COMP-667 Assignment 3 of <your name>" containing:

- Source code
- Instructions that explain how to compile and run the code (if you are using other languages than C, C++, Ada, Java, AspectJ, be prepared to give me a small demo)
- Text explaining your design decisions
  - Justify your interface
    - Explain what makes your library easy to use
    - Explain what makes your library safe to use
  - Justify your implementation decisions
    - Explain why you use the programming language features that you use
- Text explaining how we can modify the bank application to run specific test transactions. We will want to create our own account objects, and then start transactions (transfer, conditional withdraw and balance queries) on them. Please explain clearly how to specify the transaction commit order.