miniGL

Alejandro P. Revilla, jPOS.org

Abstract

Many jPOS based applications have to deal with accounts and have to accurately and efficiently keep track of its transactions and balances.

I've found myself writing ad-hoc transaction logging software using custom database schemas and balance calculation strategies over and over, and I think this may have happened to many of you as well.

While writting custom logic is certainly a feasible option and may seem like the simplest approach, code that looks okay in a test environment may degrade once you load it with hundreds of millions of transactions. Code that passes all tests in a development environment may encounter nasty problems under heavy load, caused by race conditions, poor performance, etc.

And then you need a myriad of supporting applications for extracts, analysis, reporting, data-entry, UI, etc. that have to be customized to your unique data model.

miniGL attempts to address this problem by providing a minimalistic solution to the recurring task of safely and efficiently keeping track of accounts.

While doing so, miniGL explores the possibility of leveraging double-entry accounting methods in situations were one would tend to use just single-entry.

1. Project goal and status

The project's goal is to solve in a secure and elegant way the aforementioned problem.

In order to kick start things, I've developed a prototype that uses `Hibernate` for the O/R mapping. This prototype incorporates extensive feedback and code from Anthony Schexnaildre <aps@storedvalueconsulting.com> and Andy Orrock <orrock@olsdallas.com> as well as past experience on related projects with A/P Daniel Larrosa <dflc@cs.com.uy>, Dave Bergert <dbergert@tranvia.com> and many other jPOS developers and users that I have been regularly bugging about this for the last couple of years. Special thanks go to Mark Salter <marksalter@dsl.pipex.com> for proof-reading this announcement.

I don't claim that this is the ultimate solution, it's just something to start with. I'd be okay if we have to drop every single line of what we have today, if together, as a community, we manage to figure out a better way to do it. That's the whole idea of this early release.

In the following sections, I'll describe what we have now. While you read them, you may want to open miniGL's javadocs [http://jpos.org/minigl/javadoc/].

2. Account, Entries and Transactions

The whole idea of miniGL is to keep track of accounts. You need accurate balances at a given point in time, and you need them fast. You need to support a massive number of transactions, so you have to be very careful about when to lock and when to use a lock-less strategy, you have to fine tune all database queries and processing.

I've seen many developers using an `account table` with a `balance column`, then some kind of `entries table` where they keep each transaction.

When you insert an entry, you need to lock the account record and update its balance. This may be alright if you just need to access the current balance for a given account in a given set of entries, but you still need to scan all entries when you need to calculate the balance at a given point in time, such as the end of a given business day.

In miniGL, we use an hybrid approach, we store records that have impact on a given account in an entries table, but we don't store the balance in the account table, we just scan the entries to compute a given balance and use a disposable checkpoint table to accelerate balance calculation for heavy used accounts.

2.1. Account hierarchy

We have found it very useful to keep accounts within a user-specified hierarchy, a chart of accounts. Imagine you are to keep track of an ATM's bills stock, you can split the total balance in multiple accounts, one per bill cassette, i.e:

```
ATM Stock
   10 dollar bills cassette ................... 2000.00
   20 dollar bills cassette ................... 2000.00
   50 dollar bills cassette ................... 2000.00
  100 dollar bills cassette ................... 2000.00
ATM Stock Total ..................................... 8000.00
```

or you can have a stored value card with both settled and pending transactions, you could split it in two accounts:

```
Account XXX
   Current ................................... 1000.00
   Pending Pre-Auth........................... -100.00
Account XXX Total ..................................... 900.00
```

miniGL has an `Account` abstract base class extended by two concrete classes, `CompositeAccount` and `FinalAccount`. Using this simple `composite pattern` we can support an unlimited number of nesting levels.

This can be used to support any user-specified account structure, including standard groups for Assets, Liabilities, Equity, Earnings and Losses. It is up to the system architect to use such standard structure or just a set of ad-hoc groups.

When you call `getBalance(...)` in a `CompositeAccount` you get the sum of its children accounts.

2.2. Grouping Entries

Most financial transactions involve multiple accounts (cardholder, fees, etc.) so it makes sense to group them together within some kind of container.

In miniGL we use two classes, `GLTransaction` and `GLEntry`. `GLTransaction` is the container for multiple `GLEntry` objects. [1]

[1] We use the `GL` prefix because `Transaction` and `Entry` are heavily used names in most APIs.

3. The Journal

The Journal is a container of `GLTransactions`. It is up to the system designer to use just a single `Journal` or many of them.

When you `post` a `GLTransaction`, you post it to a given `Journal`. When you request the balance of a given account at a given point in time, you do so using a `Journal` too. The same account can have different balances in different journals.

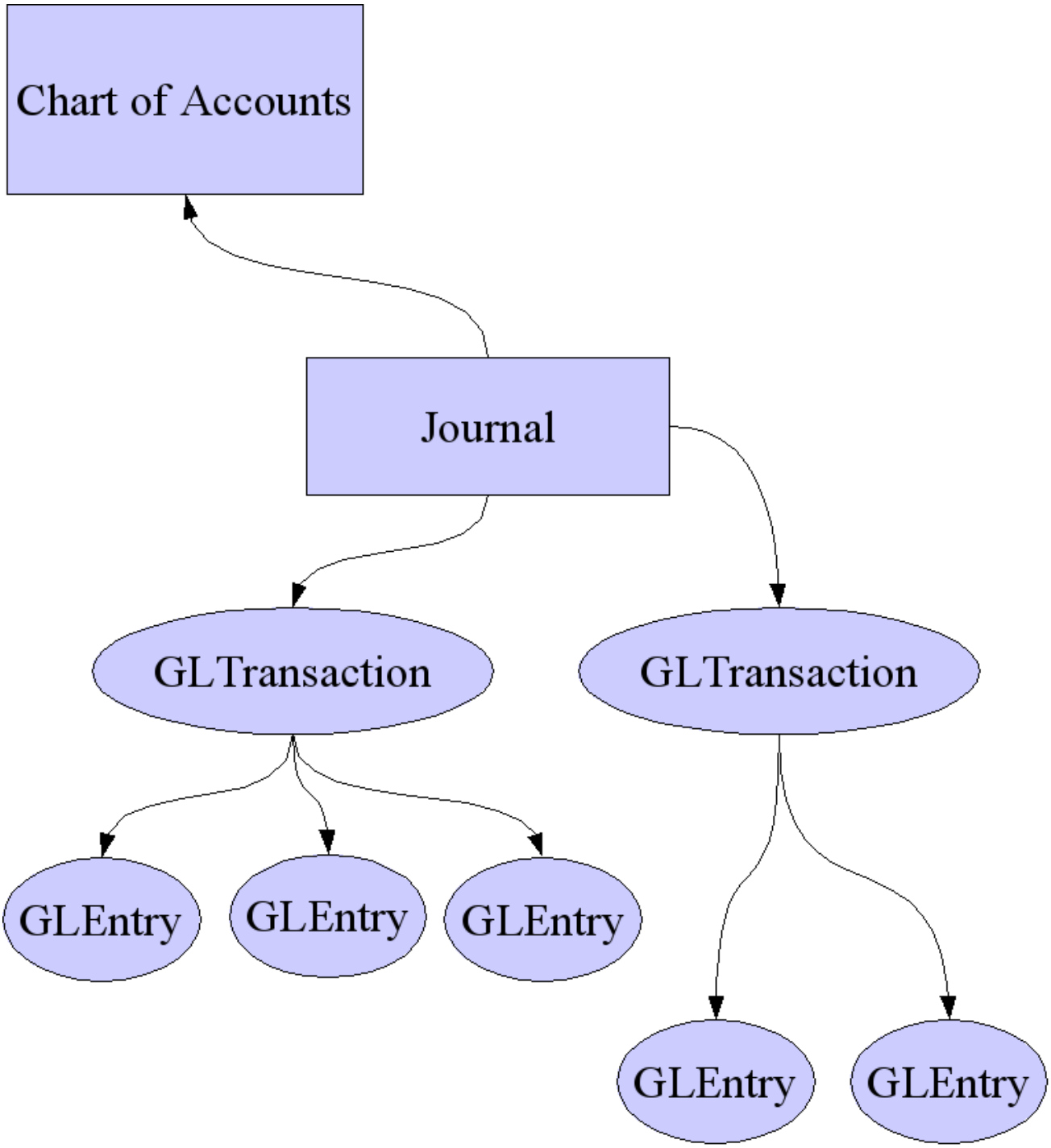

Figure 1. Journal and Transactions

4. Permissions and Rules

For our own protection, I thought it would be nice to include the ability to validate permissions, for example:

- User can post in a given Journal.
- User can post a transaction in a given Journal using a posting date older than the current date.
- User can create CheckPoints in a given Journal.
- etc...

We can also have rules to ensure that a given account won't surpass a maximum balance, and won't go below zero. We can define this at the final account level or at higher composite accounts level.

miniGL has a few rule implementations for this proof-of-concept; I hope that, working together, we can develop more and more out-of-the-box rules.

Table 1. Available Rules

Rule	Description
CanPost	Verify that user can post to a given journal. This rule asserts the following conditions: • User has Permission.POST to this journal. • Journal is not closed. • Transaction posting date is not null. • If Journal has a start date, check it against txn.postDate • If Journal has an end date, check it against txn.postDate • Check entries' accounts for expired accounts
DoubleEntry	Assert double-entry accounting rules.
FinalMaxBalance	Checks the maximum balance of a final account
FinalMinBalance	Checks the minimum balance of a final account
CompositeMaxBalance	Checks the maximum balance of a composite account
CompositeMinBalance	Checks the minimum balance of a composite account

5. XML export/import

While developing this early prototype I faced a "chicken and egg" problem, I had no UI for data-entry and my database schema was in a state of flux, so I thought it could be nice to have the ability to export to and import from XML. This would also be useful during regression testing.

I've used the following DTD [http://jpos.org/minigl.dtd]

```
<?xml version="1.0" encoding="UTF-8" ?>

<!ELEMENT minigl (
    create-schema?,user*,currency*,chart-of-accounts*,journal*,transaction*
)*>

<!ELEMENT create-schema EMPTY>
<!ELEMENT user (name+,fullname?,grant*) >
<!ELEMENT name (#PCDATA) >

<!ELEMENT fullname (#PCDATA) >
<!ELEMENT grant (#PCDATA) >
<!ATTLIST grant user CDATA #IMPLIED>

<!ELEMENT currency (symbol+,name*) >
<!ATTLIST currency id CDATA #REQUIRED>
<!ELEMENT symbol (#PCDATA) >

<!ELEMENT chart-of-accounts (description?,composite-account*,account*) >
<!ATTLIST chart-of-accounts code CDATA #REQUIRED>
<!ATTLIST chart-of-accounts created CDATA #IMPLIED>
<!ATTLIST chart-of-accounts currency CDATA #REQUIRED>
<!ELEMENT description (#PCDATA) >

<!ELEMENT composite-account (description?,(account|composite-account)*) >
<!ATTLIST composite-account code CDATA #REQUIRED>
<!ATTLIST composite-account type (debit|credit) #REQUIRED>

<!ELEMENT account (description?) >
<!ATTLIST account code CDATA #REQUIRED>
<!ATTLIST account type (debit|credit) #REQUIRED>
<!ATTLIST account currency CDATA #IMPLIED>

<!ELEMENT journal (name+,start?,end?,status+,chart+,grant*,rule*) >
<!ELEMENT start (#PCDATA) >
<!ELEMENT end (#PCDATA) >
<!ELEMENT status (#PCDATA) >
<!ELEMENT chart (#PCDATA) >
<!ELEMENT rule (#PCDATA|param)* >
<!ATTLIST rule clazz CDATA #REQUIRED>
<!ATTLIST rule account CDATA #IMPLIED>
<!ELEMENT param (#PCDATA) >

<!ELEMENT transaction (detail?,entry+) >
<!ATTLIST transaction date CDATA #REQUIRED>
<!ATTLIST transaction post-date CDATA #REQUIRED>
<!ATTLIST transaction journal CDATA #REQUIRED>
<!ELEMENT detail (#PCDATA) >

<!ELEMENT entry (detail?,amount,foreign-amount?) >
<!ATTLIST entry account CDATA #REQUIRED>
<!ATTLIST entry type (debit|credit) #REQUIRED>
<!ELEMENT amount (#PCDATA) >
<!ELEMENT foreign-amount (#PCDATA) >
<!ATTLIST foreign-amount currency CDATA #REQUIRED>
```

Here is an example of an exported file, used as the starting point for our unit tests. Most elements are self explanatory.

```
```

```
<?xml version="1.0" ?>
<!DOCTYPE minigl SYSTEM 'http://jpos.org/minigl.dtd'>
<minigl>
 <create-schema />

 <!-- add user 'bob' with 'read', 'write' and 'grant' permissions -->
 <user>
  <name>bob</name>
  <fullname>Bob Book Keeper</fullname>
  <grant>read</grant>
  <grant>write</grant>
  <grant>grant</grant>
 </user>

 <!-- add user 'eve' with no permissions  -->
 <user>
  <name>eve</name>
  <fullname>Eve Noperm</fullname>
 </user>

 <!--
   add some currencies. Although we're using ISO-4217,
   nothing prevents us from creating ad-hoc currencies such as
   'miles', 'points', 'bonuses' etc.
  -->

 <currency id="840">
  <symbol>USD</symbol>
  <name>US Dollars</name>
 </currency>

 <currency id="032">
  <symbol>ARG</symbol>
  <name>Pesos argentinos</name>
 </currency>

 <currency id="858">
  <symbol>$U</symbol>
  <name>Pesos uruguayos</name>
 </currency>

 <!--
   A sample chart with a simple structure used by our unit tests

   1 Assets
      11 Cash
         111 Cash Dollars
         112 Cash Pesos
         113 Bank Account

   2 Liabilities
      21 Loan
      23 Cards

   3 Equity
      31 Bob's equity
      32 Alice's equity

   4 Earnings
   5 Losses

 -->

 <chart-of-accounts code="TestChart" created="20050101" currency="840">
  <description>Test Chart</description>
  <composite-account code="1" type="debit" >
   <description>Assets</description>
   <composite-account code="11" type="debit" >
    <description>Cash</description>
    <account code="111" type="debit" >
     <description>Cash Dollars</description>
```

```
    </account>
    <account code="112" type="debit" currency="858">
     <description>Cash Pesos</description>
    </account>
    <account code="113" type="debit">
     <description>Bank Account</description>
    </account>
   </composite-account>

 </composite-account>
 <composite-account code="2" type="credit" >
  <description>Liabilities</description>
  <account code="21" type="credit">
   <description>Loan</description>
  </account>
  <composite-account code="23" type="credit">
   <description>Cards</description>
  </composite-account>
 </composite-account>
 <composite-account code="3" type="credit" >
  <description>Equity</description>
  <account code="31" type="credit" >
   <description>Bob's equity</description>
  </account>
  <account code="32" type="credit" >
   <description>Alice's equity</description>
  </account>
 </composite-account>
 <composite-account code="4" type="credit" >
  <description>Earnings</description>
 </composite-account>
 <composite-account code="5" type="debit" >
  <description>Losses</description>
 </composite-account>
</chart-of-accounts>

<!--
  a TestJournal with some user permissions and user-specified
  rules, used by our unit tests
--->
<journal>
 <name>TestJournal</name>
 <start>20010101</start>
 <end>20201231</end>
 <status>open</status>
 <chart>TestChart</chart>
 <grant user="bob">post</grant>
 <grant user="bob">checkpoint</grant>
 <rule clazz="org.jpos.gl.rule.CanPost">
  Verifies permissions, start/end dates, status, and chart
 </rule>
 <rule clazz="org.jpos.gl.rule.DoubleEntry">
  Verifies that credits equals debits
 </rule>
 <rule clazz="org.jpos.gl.rule.CompositeMaxBalance" account="1">
   Check assets doesn't go above one million
   <param>1000000.00</param>
 </rule>
 <rule clazz="org.jpos.gl.rule.FinalMinBalance" account="11">
   Check that the account has always a balance greater than 10.00
   <param>10.00</param>
 </rule>
 <rule clazz="org.jpos.gl.rule.CompositeMinBalance" account="11">
   Check that the account has always a balance greater than 45000.00
   <param>45000.00</param>
 </rule>
 <rule clazz="org.jpos.gl.rule.FinalMaxBalance" account="31">
   Put a limit to Bob's equity
   <param>100000.00</param>
 </rule>
</journal>
```

```
<!--
                          Debit | Credit
  ----------------------------+---------
  111 Cash Dollars     10000.00|
  31  Bob's equity              | 10000.00

  Bob's initial deposit
-->
<transaction post-date="20050101" date="20050101130000" journal="TestJournal">
 <detail>Bob's initial deposit</detail>

 <entry account="111" type="debit">
  <detail>Bob Check #001</detail>
  <amount>10000.00</amount>
 </entry>
 <entry account="31" type="credit">
  <detail>Bob Initial cash (USD)</detail>
  <amount>10000.00</amount>
 </entry>
</transaction>

<!--
                          Debit | Credit
  ----------------------------+---------
  111 Cash Dollars      5000.00|
  112 Cash Pesos        5000.00|          ($12500 pesos)
  32  Alice's equity           | 10000.00

  Alice's initial deposit
-->
<transaction post-date="20050101" date="20050101130001" journal="TestJournal">
 <detail>Alice's initial deposit</detail>

 <entry account="111" type="debit">
  <detail>Alice Check #001</detail>
  <amount>5000.00</amount>
 </entry>
 <entry account="112" type="debit">
  <detail>Alice Check #002</detail>
  <amount>5000.00</amount>
  <foreign-amount currency="858">12500.00</foreign-amount>
 </entry>
 <entry account="32" type="credit">
  <detail>Alice Initial deposit converted to Pesos</detail>
  <amount>10000.00</amount>
 </entry>
</transaction>

<!--
                          Debit | Credit
  ----------------------------+---------
  111 Cash Dollars     10000.00|
  31  Bob's equity              | 10000.00

  Bob's second deposit
-->
<transaction post-date="20050102" date="20050102130000" journal="TestJournal">
 <detail>Bob's second deposit</detail>

 <entry account="111" type="debit">
  <detail>Bob Check #002</detail>
  <amount>10000.00</amount>
 </entry>
 <entry account="31" type="credit">
  <detail>Bob More cash (USD)</detail>
  <amount>10000.00</amount>
 </entry>
</transaction>

<!--
```

```
                          Debit | Credit
   ---------------------------+---------
   113 Bank Account     20000.00|
   21  Loan                    | 20000.00

   Loan

 -->
 <transaction post-date="20050102" date="20050102130000" journal="TestJournal">
  <detail>Loan</detail>

  <entry account="113" type="debit">
   <detail>Deposit reference #1234</detail>
   <amount>20000.00</amount>
  </entry>
  <entry account="21" type="credit">
   <detail>Loan reference #4321</detail>
   <amount>20000.00</amount>
  </entry>
 </transaction>
</minigl>
```

The above example is used by our unit tests in order to verify some initial balances. After that some accounts are created under the "cards" composite account and multiple transactions are fired against random cards.

6. Invitation

If this problem sounds familiar, we invite you to dive in and help us build a better solution, far better than what's presented here.

We expect to define a miniGL API and then try multiple implementations, starting with but not limited to Hibernate.

We opened a **miniGL** subproject under the **jPOS.org** umbrella. We have created a mailing list `minigl-dev@jpos.org` (you are welcome to subscribe by sending an e-mail to `minigl-dev-subscribe@jpos.org`) and we've opened a subversion repository [svn://cvs.jpos.org/lab/minigl/trunk] where you can get the latest version of miniGL.

You can also find us online most of the time at

```
   irc.freenode.net channel #jpos
```

--Alejandro