Automated Java component assembly with Poplar

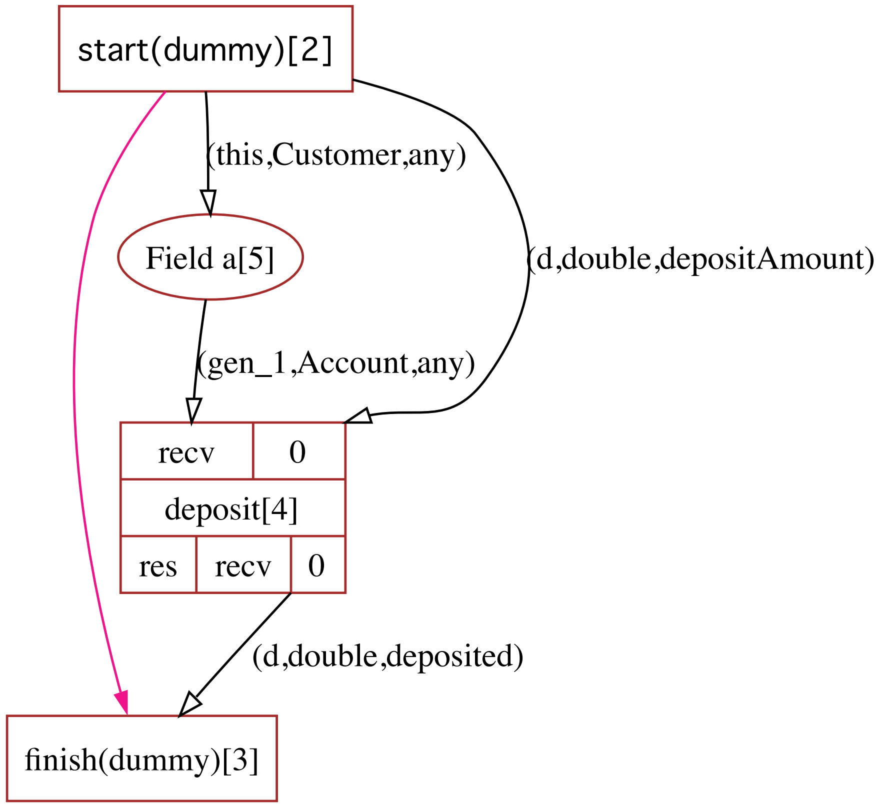
NII/University of Tokyo, Honiden lab., Johan Nyström-Persson
johan@nii.ac.jp http://www.poplar-lang.org

Poplar is a Java extension that was designed in order to help programmers handle API changes that occur naturally as a result of class and component upgrades. Poplar generates, checks and regenerates integration code automatically.

```
class Customer {

      Account a: (none);
      void doTransactions() {
            double d: (depositAmount) = 5.0;
            int id: (customerId) = 3145;

            //Integration query
            #transform(d, deposited);
      }
}
```

```
class Account {
      resource balance;

      double aBalance = 0d;

      void deposit(double amount)
            amount: depositAmount, +deposited. {
            aBalance += amount;
      }

      void withdraw(double amount)
            amount: withdrawAmount, +withdrawn.
      {
            aBalance -= amount;
      }
}
```

In Poplar, Java methods are annotated with extra information that indicates the purposes of fields, variables, and methods, and how they may interact with each other.
In response to an integration query, which can request *transformation* or *production,* Poplar is able to generate a *plan* for the integration and extract generated code from it. The following plan corresponds to the Java code `this.a.deposit(d);` .



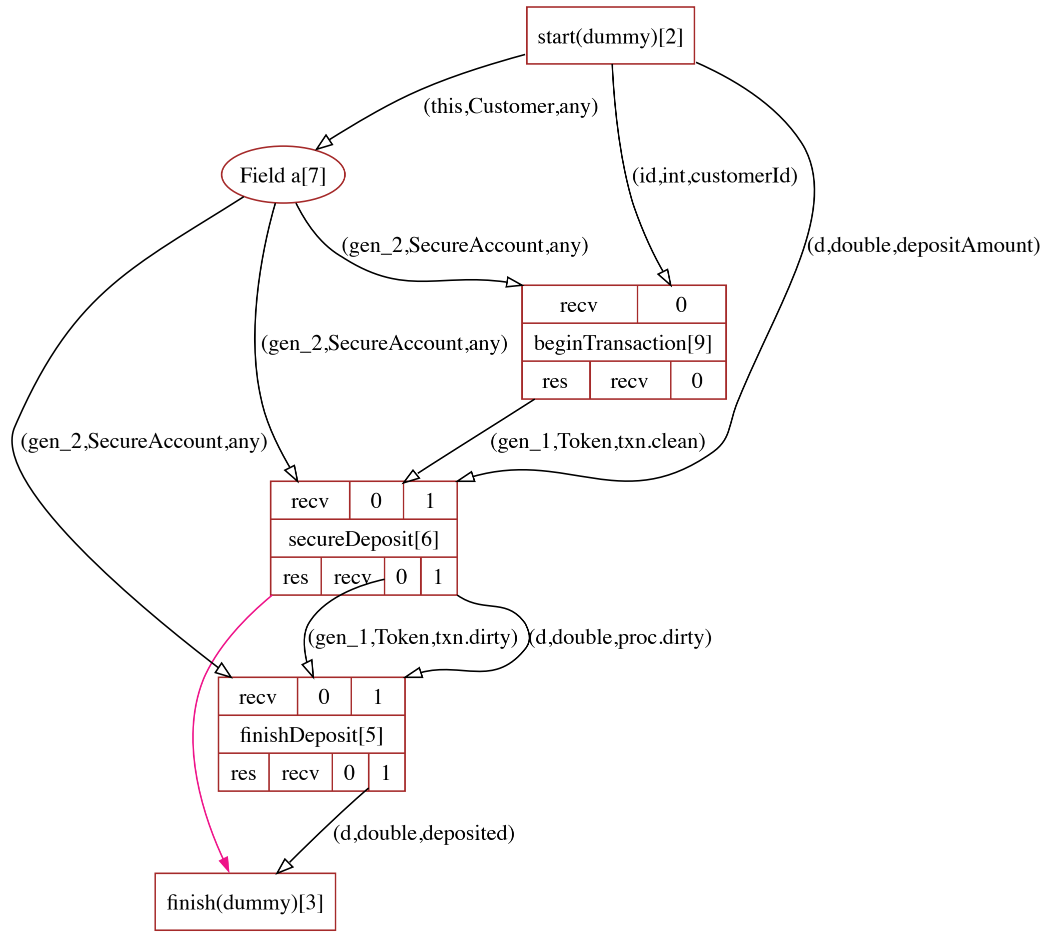
The following is a more advanced Account class, which requires a transaction to be opened and identified by a Token, in order to achieve the same functionality as simple Account.
Protocols such as `txn@clean->dirty` indicate a transition between two mutually exclusive states. Effects can reside in *abstract resources* using an annotation such as `[* transactions]`. If the method stopAllTransactions is invoked, then the effects associated with this resource are logically destroyed.

```
package oh2011;

class SecureAccount {
      resource balance, transactions;
      double aBalance;

      static class Token { ... }

      Token beginTransaction(int id)
            id: customerId;
            result: +txn@clean [*transactions].
      {
            return new Token(); //etc.
      }

      void secureDeposit(Token token, double amount) [!balance]
            token: txn@clean->dirty [*transactions];
            amount: depositAmount, +proc@dirty.
      {
            aBalance += amount; //etc.
      }


      void finishDeposit(Token token, double amount)
            token: txn@dirty->committed [*transactions];
            amount: proc@dirty->deposited, +deposited.
            {
                  commit(token); //etc.
            }

      void stopAllTransactions() [!transactions] { ... }
}
```

The figure below shows the plan that integrates the same client with the secure account (except that the type of the `Account a` field was changed to `SecureAccount a`).
This corresponds to the following Java code:

```
Token t = a.beginTransaction
(id);
a.secureDeposit(t, d);
a.finishDeposit(t, d);
```

A first release of the Poplar compiler, with limited features, is planned for late June 2011. Please check http://www.poplar-lang.org for news and downloads at that time.


start(dummy)[2]
(this,Customer,any)
Field a[7]
(id,int,customerId)
(d,double,depositAmount)
(gen_2,SecureAccount,any)
recv | 0
beginTransaction[9]
res | recv | 0
(gen_2,SecureAccount,any)
(gen_2,SecureAccount,any)
(gen_1,Token,txn.clean)
recv | 0 | 1
secureDeposit[6]
res | recv | 0 | 1
(gen_1,Token,txn.dirty)
(d,double,proc.dirty)
recv | 0 | 1
finishDeposit[5]
res | recv | 0 | 1
(d,double,deposited)
finish(dummy)[3]