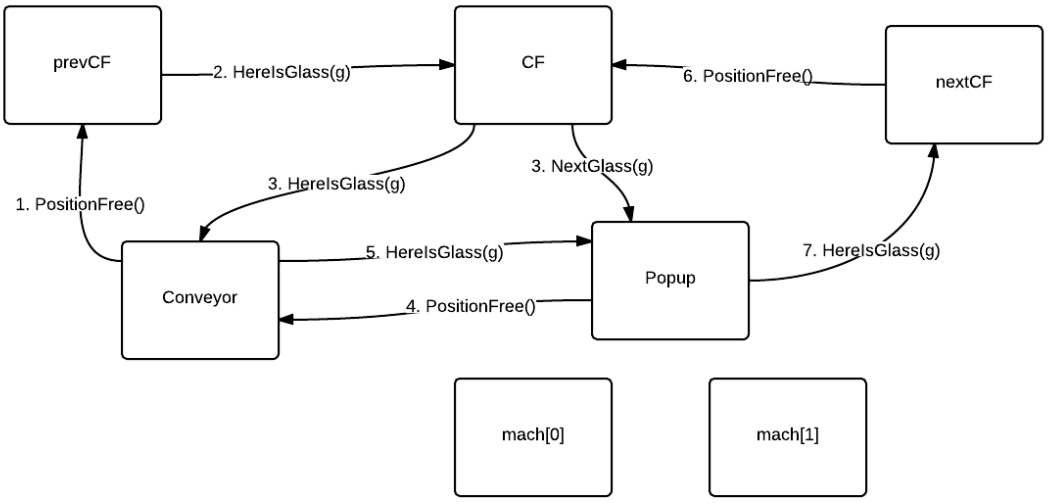
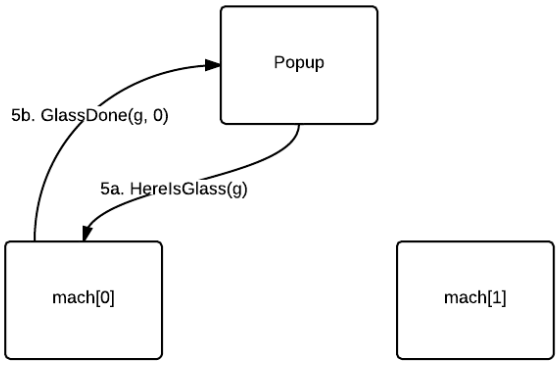
Evan Brown
CSCI 201
Mon-Wed

Glass Factory V.1 Design

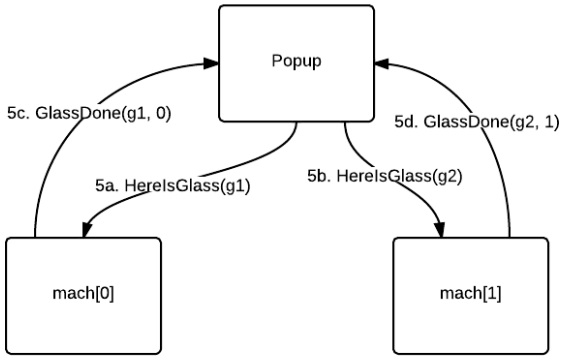
No Machine Processing Diagram



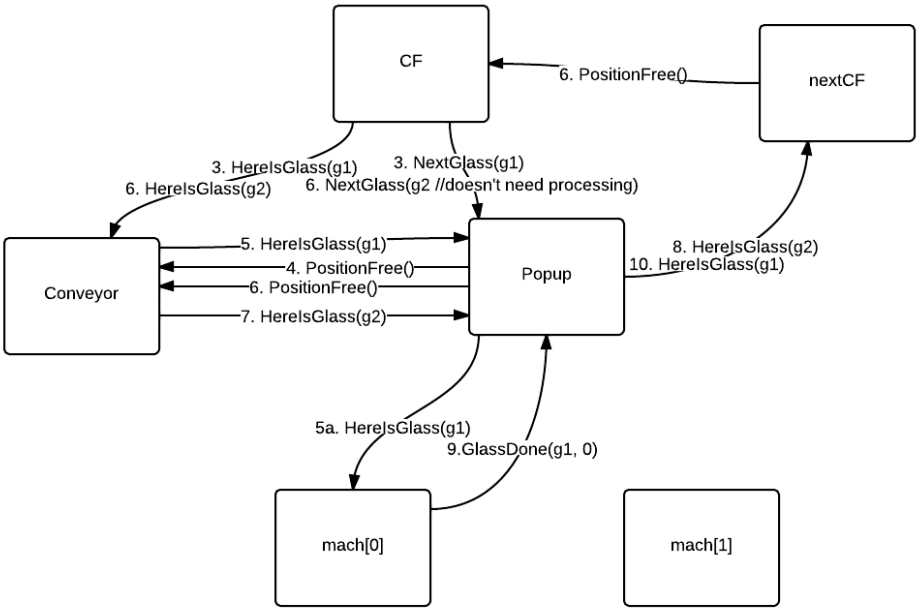
Machine Processing
(Same as above up to 5)



Multiple Machine Processing
(Same as no processing up to 5)



Processing and Pass-through
(Pass-through can happen when one or both machines
are occupied)



Data Classes:

```
class Glass {
        int id;
        Map<MachineType, Boolean> recipe;

        Boolean getNeedsProcessing(MachineType);
}
```

ConveyorFamilyImplementation (PseudoAgent)

Data:

```
Conveyor c;
Popup p;
```

Messages:

```
HereIsGlass(Glass g) {
        c.HereIsGlass(g);
}
PositionFree() {
        p.PositionFree();
}
GlassDone(Glass g, int index) {
        p.GlassDone(g, index);
}
```

Not a real agent → no scheduler or actions

Conveyor Agent

Data:

```
ConveyorFamily prevCF;
Popup p;
Transducer t;
int id; //place in GUI

Glass g;

enum GlassState {pending, arrived, moving, atEnd, done, none};
GlassState gs;

boolean posFree;
```

Messages:

```
HereIsGlass(Glass g) {
        this.g = g;
```

```
        if (gs == none) // it could have arrived already, maybe in a buggy scenario (for resiliency)
                gs = pending;
}
PositionFree() {
        posFree = true;
}
// Transducer Event
eventFired(TChannel channel, TEvent event, Object[] args) {
        if (front sensor pressed)
                gs = arrived;
        else if (front sensor released)
                gs = moving;
        else if (back sensor pressed)
                gs = atEnd;
        else if (back sensor released)
                gs = done;
}
```

Scheduler:

```
if (gs == pending) {}
else if (gs == arrived) {
        startConv();
}
else if (gs == moving) {}
else if (gs == atEnd) {
        if (posFree)
                sendGlass(); // send to popup
        else
                tellPopupReadyAndWait();
}
else if (gs == done) {
        readyForMore(); // tell popup glass has arrived
}
else if (gs == none) {}
```

Actions:

```
startConv() {
        t.fireEvent(do start conveyor);
}
stopConv() {
        t.fireEvent(do stop conveyor);
}
tellPopupReadyAndWait() {
        stopConv();
        p.NextGlass(g);
}
```

```
sendGlass() {
        p.HereIsGlass(g); // tell popup sending glass
        startConv();
}
readyForMore() {
        gs = none;
        posFree = false; // popup now occupied
        stopConv();
        prevCF.PositionFree(); // this conveyor now free
}
```

Popup Agent

Data:

```
ConveyorFamily nextCF;
Conveyor c;
WorkStation mach[2];
Transducer t;
int id; // place in animation

private class MyGlass {
        Glass g;

        enum GlassState {pending, needsProcessing, atMachine, doneProcessing, waiting};
        GlassState gs;
        int i; // machine index
}
List<MyGlass> glasses;

MachineType mt;
TChannel mtc; // machine type channel
enum MachineState {free, processing, done};
MachineState ms[2];

Semaphore animSem[5]; // animation delay semaphores: load finished, move up, move down,
                      // release finished, machine load finished

boolean mFree[2], up, posFree; // machine free, up or down, nextCF position free
```

Messages:

```
NextGlass(Glass g) {
        glasses.add(new MyGlass(g));
}
HereIsGlass(Glass g) {
        for (mg : glasses) // find matching glass in glasses
                if (mg.g.equals(g)) {
```

```
                        if (mg.g.getNeedsProcessing(mt))
                                mg.gs = needsProcessing;
                        else
                                mg.gs = waiting;
                }
}
PositionFree() {
        posFree = true;
}
GlassDone(Glass g, int index) {
        for (mg : glasses)
                if (mg.g.equals(g))
                        mg.gs = doneProcessing;
}

// Transducer Event
eventFired(TChannel channel, TEvent event, Object[] args) {
        if (it's an event for this popup) {
                if (load finished)
                        animSem[0].release();
                else if (moved up)
                        animSem[1].release();
                else if (moved down)
                        animSem[2].release();
                else if (release finished)
                        animSem[3].release();
        }
        else if (machine load finished)
                animSem[4].release();
}
```

Scheduler:

```
if (there exists a MyGlass mg in glasses s.t. mg.gs == waiting) {
        if (posFree)
                sendGlass(mg);
        else
                return false; // popup is holding glass so can't do anything else
}
else if (there exists a MyGlass mg in glasses s.t. mg.gs == needsProcessing) {
        int i;
        if (machine 0 is free)
                i = 0;
        else // machine 1 must be free
                i = 1;
        moveUpAndToMachine(mg, i);
}
```

```
else if (there exists a MyGlass mg in glasses s.t. mg.gs == doneProcessing) {
        removeFromMachine(mg, i);
}
else if (there exists a MyGlass mg in glasses s.t. mg.gs == pending
            and (mg doesn't need processing or there is a machine open)){ // else can't take more
        readyForGlass(mg);
}
```

Actions:

```
moveUpAndToMachine(MyGlass mg, int i) {
        mg.i = i;
        mg.gs = atMachine;
        t.fireEvent(do move up);
        acquire move up sem;
        up = true;

        mach[i].HereIsGlass(mg.g);
        t.fireEvent(do load machine i);
        acquire machine load sem;
        mFree[i] = false;
}
removeFromMachine(MyGlass mg, int i) {
        if (!up) { // move up if necessary
                t.fireEvent(do move up);
                acquire move up sem;
                up = true;
        }
        t.fireEvent(do machine i release glass);
        acquire load sem;
        mFree[i] = true;

        t.fireEvent(do move down);
        acquire move down sem;
        up = false;

        mg.gs = waiting;
}
sendGlass(MyGlass mg) {
        nextCF.HereIsGlass(mg.g);
        t.fireEvent(do release glass);
        acquire glass release finished sem;
        glasses.remove(mg);
        posFree = false; // nextCF now occupied
}
readyForGlass(MyGlass mg) {
        if (up) { // move down if necessary
```

```
                t.fireEvent(do move down);
                acquire move down sem;
                up = false;
        }
        c.PositionFree();
        acquire load sem;
        mg.gs = (mg.g.getNeedsProcessing(mt)) ? needsProcessing : waiting;
}
```