Workflows

An introduction to ittyFlow and the concepts behind it

The "Workflow" Problem

- We have some long lived process (a "task")
- This process needs input from multiple external participants
- Execution will start and then block until a participant intervenes and advances it to the next step
- Repeat until done

Challenges

- Managing state
  - the process is suspended and resumed frequently
  - Information comes from external participants
- Managing control flow

Concepts

- Let's formalize our vocabulary
  - Task
  - Wait state
  - Workflow
  - Actors

Actor

- An entity outside of the system
- Pushes events into the system
- Examples:
  - User interacting on a web page
  - Thread polling event api table

Wait state

- Represents what a task is waiting for
- Enumeration

Workflow

- A directed graph
- Vertices represent wait states
- Edges represent events
- Changing states only occurs in response to an event

Task

- A long running process that needs occasionally needs to wait for one or more actors
- Has a wait state which represents current position in workflow
- Has addition domain specific state

Goals

- Ease definition of domain specific “tasks”
- Show the big picture
- Lightweight
- Provide insolation between Actor facing interfaces and workflow

Wait state objects

- Can be almost anything
- final static & immutable
- Can have state

  Suggestions: name, description, and ActionBean

- No behavior in wait states

Example WaitState

```
public enum AssayRunState {
    /**
     * Indicates that the run is waiting for compound
     * plate format groups to be selected
     */
    Waiting_For_Map_Groups("Waiting for Map Groups", "1",
                    ChooseGroupsActionBean.class),

    /**
     * Indicates that some of the run's compound plate
     * format groups need plates to be created
     */
    Waiting_For_Plates("Waiting For Plates", "1",
                    LoadPlatesActionBean.class),
  ...
```

Events

- Event = method call that returns a "wait state"
- Different events have different signatures
- Events can be parameterized
- First parameter is always task object
- All execution takes place in response to an event, and *may* result in a state transition

Example Event

```
public interface Transitions {

    public AssayRunState userSelectedPlateMaps(AssayRun assayRun,
            String[] plateMaps, boolean includeControlsAtBeginning);

    public AssayRunState userClickedCancel(AssayRun assayRun);

...

}
```

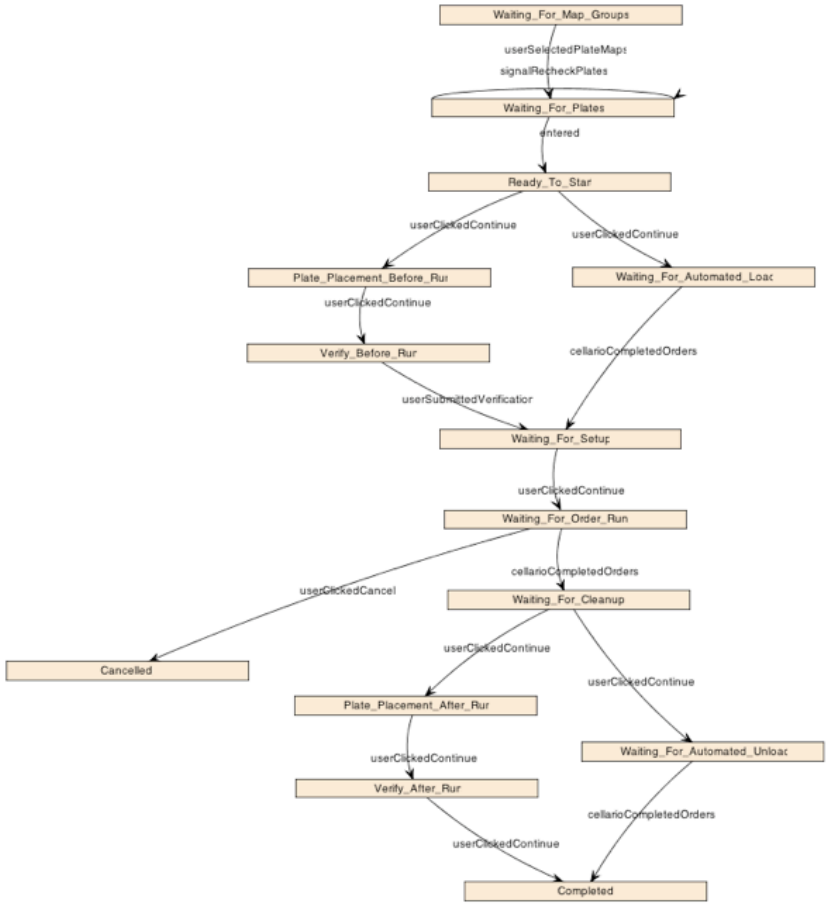
Workflow

- Uses generics to parameterize over <WaitState, Transitions>
- Instantiate and add states with associated event listeners

Example workflow

```
Workflow<AssayRunState, Transitions> workflow =
    new Workflow<AssayRunState, Transitions>(AssayRun.class, Transitions.class);

workflow.addListener(AssayRunState.Waiting_For_Map_Groups, new AbstractTransitions()
   {
      public AssayRunState userSelectedPlateMaps(AssayRun assayRun,
          String[] plateMaps, boolean includeControlsAtBeginning)
      {
         ... mutate assayRun as needed.  Make service calls, validation, etc ...
          return AssayRunState.Waiting_For_Plates;
      }
   });
```



Use data flow analysis to automatically derive topology for visualization...

Interaction with Actors

- call signal(task) and the event to fire
- Checks both start and resulting state validity against workflow
- Automatically generates entered() events

Interaction example

```
Workflow<AssayRunState, Transitions> wf =
  AssayRunWorkflowFactory.makeWorkflow();

AssayRun assayRun = new AssayRun();

wf.signal(assayRun).userSelectedPlateMaps(null,
  new String[] {"G001", "G002", "G003"} , false);
```