Overview

This application is a mock personal finance manager. Its goal is to explore the issues involved in integrating the CE Task Engine into a typical user-driven desktop application. As much as possible, the structure of the interaction is defined by the task model. Once the intelligent interaction is initiated, the user interface reacts to the state of the task engine, showing different panels to acquire user input or report on the state of the interaction, among other things.

Instructions for a Typical Interaction

- Select "File" → "New" from the main menu to ensure the application is initialized with a blank document
- Select "Account" → "Create…" from the main menu
- Enter the details for a mock checking account, for example:
    - Account Number: 1234
    - Account Name: My Checking Account
    - Institution: My Bank
- Click the new account in the main view's left panel to select it
- Select "Account" → "Enter Transactions…" from the main menu
- Enter some mock transactions; for example:
    - Payee: "My Phone Company", "Pizza Planet", "The Cable Company", …
    - Amount: *any number*
    - Date: *click the drop-down date picker and select any date*
- Select "Account" → "Reconcile" from the main menu; This initiates the task-modeled account reconciling process
    - Enter the start date (e.g., first of the month), closing date (e.g., end of the month), previous balance, total withdrawn amount, etc. values. These would normally come from a paper bank statement, but will have to be culled from the mock data entered above.
    - Click "Next" and follow the remaining prompts

What Worked

The original idea for this project was to apply the CE Task Engine to a broad range of features in a personal finance management application. As the project progressed, the focus shifted toward exploring the questions of integration more deeply. For example,

- What is involved in moving the application's data out of the task model script (as in the CE Task examples) to become Java objects?
- Is it possible to drive the user interface from the task model?
- Is it possible to generate user interface panels for a given task?
- And so on…

Going deeper into questions of integration turned out to be an interesting direction. With only relatively minor changes to the CE Task source code, many (to all) of the questions

and issues that surfaced were either solved or proven feasible. Much of the intelligent interaction in "Checkbook" is configured by ".properties" files and the task model.

What Didn't Work

Most of the stumbling blocks to driving the interface from the task model stemmed from the CE Task engine's command-line orientation. The best example of this is that there is no implementation of the Observer pattern for the `Guide` class. That is, there is no "listener" interface that client code could implement and register with the task engine in order to be notified of changes as the interaction progresses through a plan. The "Checkbook" application's copy of the task engine approximates this by extending the `Guide` class and overriding the `setFocus(…)` method. Checkbook's custom guide class maintains its own list of observers which it notifies when the focus changes. Checkbook's user interface manager registers such an observer to create UI panels corresponding to the new focused task.

When a sub-task is completed, focus returns to the parent task. If the parent task has a UI panel, the user would have already seen it. This should be indicated somehow on returning to this panel. Checkbook's UI manager accomplishes this by storing a history of task panels. This is easy enough to do, but perhaps there is an additional piece of state that `Task` objects could carry that indicates that the task has been started by is not achieved yet. (Alternatively, perhaps this piece of state really means to answer the question "are all inputs defined", since defining inputs is one of a task panel's primary functions. This could likely be accomplished by building on functions that exist already in the `Task` class.)

As demonstrated by "DiamondHelp", a running task model makes it possible to allow the user to ask "What's next?" Providing such a button on a task panel meant to define the tasks inputs turned out to be a little problematic. Issuing a "next" command to determine the next step in the interaction actually put the next task in focus, leaving the old task with its inputs half-defined, if at all. A "query-only" form of the "next" command would be useful for providing information without changing the state of the interaction. In the long run, the user interface panels could be made to handle this properly. If the user uses "next" to discover then pursue a sub-task, then, on completion of the sub-task, the UI could re-display the original task's panel with any previously given input values shown in the panel.

Future Directions

This project ran out of time before it ran out of energy or fruitful directions. The over-arching future direction would be tighter integration on all fronts between the application's UI and the task engine. Many of the issues discussed in "What Didn't Work" suggest more specific directions within this.