Beyond Code: An Introduction to Model-Driven Software Development (CISC 836, Fall 2021)

Assignment 2 (MDSD with RSARTE)

Due: Tue, Oct 19

Intro

1. **Purpose of the Assignment:**
    1. Give you more design and development experience (in the context of UML-RT) by asking you to develop a system from scratch (Part I)
    2. Deepen your understanding of modeling and model-driven development with UML-RT and RSARTE (Part I)
    3. Reinforce key aspectes of the execution semantics of UML-RT (message delivery, run-to-completion steps) (Part II)
2. **Preparation:**
    1. Note that this is an **individual assignment**. That is, each student must prepare and submit his/her own solution. All work submitted must be your own. Queen's Arts and Science regulations on Academic Integrity are in effect and will be enforced, https://www.queensu.ca/artsci/students-at-queens/academic-integrity
    2. The RSARTE installation that you used for Assignment 1 will also be used for this assignment.
    3. Make sure that you are aware of the design guidelines discussed in class.
    4. If you don't know it already, familiarize yourself with the 'Rock, Paper, Scissors' game.

Part I (40 points total)

1. **Problem Description:**
   In this part, your task is to use RSARTE to design and develop an implementation of the Rock, Paper, Scissors game. More precisely, the implementation should simulate game play for two simulated (i.e., non-human) players.

   Game play consists of a sequence of rounds. In each round, the referee should ask each player for their choice, collect the responses, and determine the outcome. Each round can have one of three possible outcomes: Player 1 wins, Player 2 wins, or there is a tie (because both players returned the same choice to the referee). Between two rounds, the referee should wait for a user-specified number of seconds before it starts the next round. When asked for her choice, a player should randomly return one of the three possible choices (Rock, Paper, or Scissors). The choice should be determined using a random number generator.

   The number of rounds played in a game should depend on a command line parameter `bestOf`. The game should end as soon as a winner can be determined and after `bestOf` rounds the latest. In other words, the game should end as soon as it has become impossible for one player to lose (and for the other player to win), assuming a maximum of `bestOf` rounds. For instance, if `bestOf` is 1, game play always consists of exactly 1 round. If `bestOf` is 2, game play always consists of exactly 2 rounds. However, if `bestOf` is 3, game play could end after 2 rounds (if one player has won both of these rounds), but must end after 3 rounds. If `bestOf` is 7, the game could end after 4, 5, 6, or 7 rounds. Note that regardless of the value of `bestOf`, a game can always have 3 different outcomes (Player 1 wins, Player 2 wins, or the game is tied).

   Your design should

   - **R1**: contain three, possibly replicated, components (one for the referee and two for each of the players) that communicate solely via an appropriately chosen set of messages,
   - **R2**: accept the value for `bestOf` at code invocation time as a command line parameter,
   - **R3**: accept the amount of time that the referee should wait between two rounds (in seconds),
   - **R4**: implement game play as described above,
   - **R5**: ensure that player choices are random across games, i.e., it should be possible that players make different choices in different invocations of the game,
   - **R6**: update the user about game progress during execution via suitable, informative log messages, and
   - **R7**: respect the design guidelines discussed in class (see above).

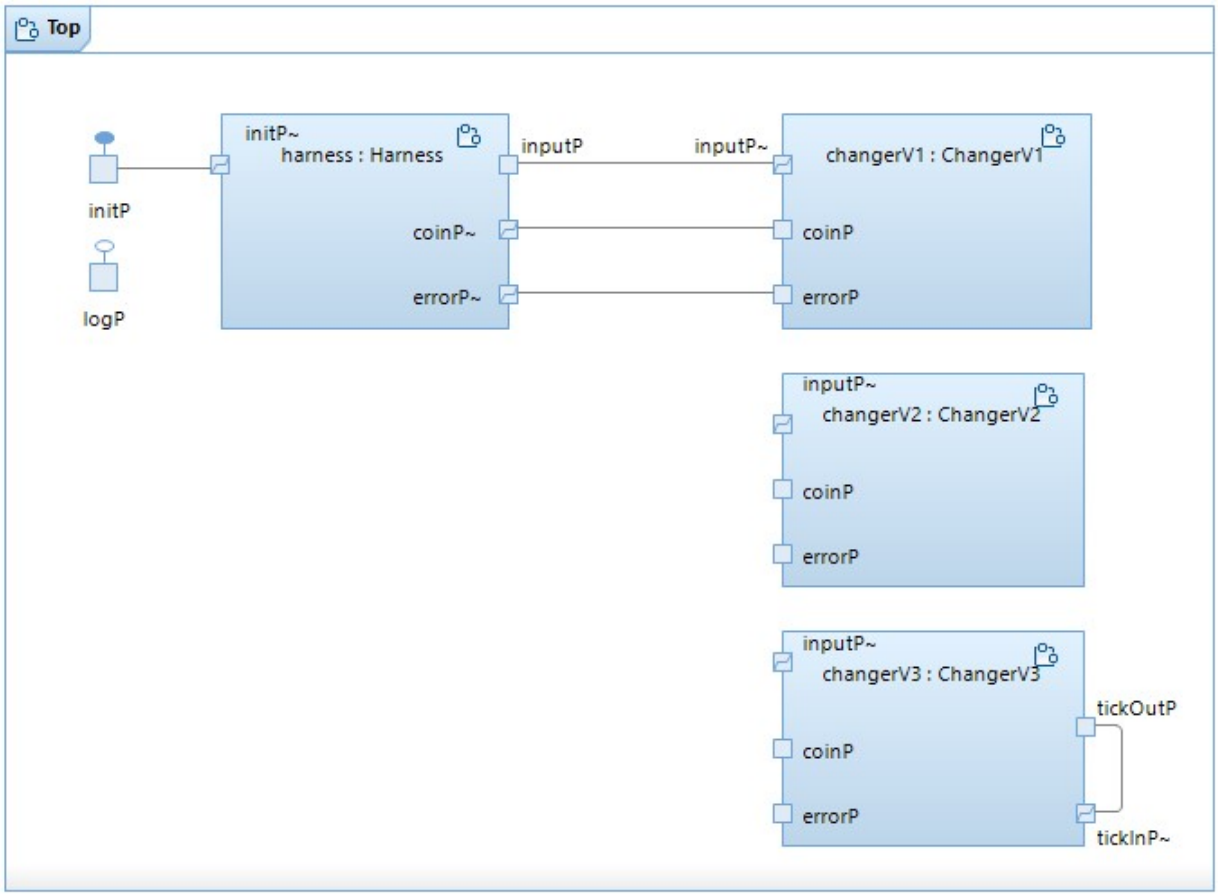
2. **Task description:**

- **Task 1** [34 points]: Use RSARTE to develop a system that satisfies the requirements above. Following the suggested MDE development process (slide 35 in this deck used at the beginning of the course), develop this system iteratively and incrementally. That is, break down development of the system with full functionality into a sequence of manageable steps such that
  - you add some small functionality to the system in each step, and
  - you can determine the correct completion of the step at the end of each step (most likely through an appropriate amount of testing).

  For more on iterative and incremental software development go here.
  One way to complete this part is without the use of replication (i.e., with two distinct player capsules). However, for full points you should define a single player capsule only and then replicate it (i.e., give it multiplicity '2'). A version without a replicated player capsule will get 30 points at most.
- **Task 2**: [4 points] Write down how you have broken down development, i.e., briefly describe each of the development steps in a text file '`Part1.txt`' that is part of your project (see Assignment 1 for instructions on how to create such a text file).
- **Task 3**: [2 points] In '`Part1.txt`' also briefly describe how your code is to be invoked (for testing purposes).
- **Task 4 (optional)**: [0 points] Watch the 2008 US Rock-Paper-Scissors Championship Match or chimpanzees play Rock, Paper, Scissors --- almost.

Part II (6 points total)

1. **Problem Description:**
   Consider a simple model for giving change as in, e.g., a traditional, coin-operated vending machine. The model consists of a harness instance and three versions of a changer instance ('`changerV1`', '`changerV2`', and '`changerV3`').



   Similiar to the hotel safe example, the harness feeds the relevant inputs to one of the changers and receives any output it produces. The ports of the harness should always be connected to the corresponding ports of exactly one of these three changer instances.

   Depending on which version of the changer is used, we get different versions of the entire system. When '`changerV1`' is used, we call the resulting system '`GiveChangeV1`'. Similarly for other versions of the changer.

   The code generated from model is invoked with

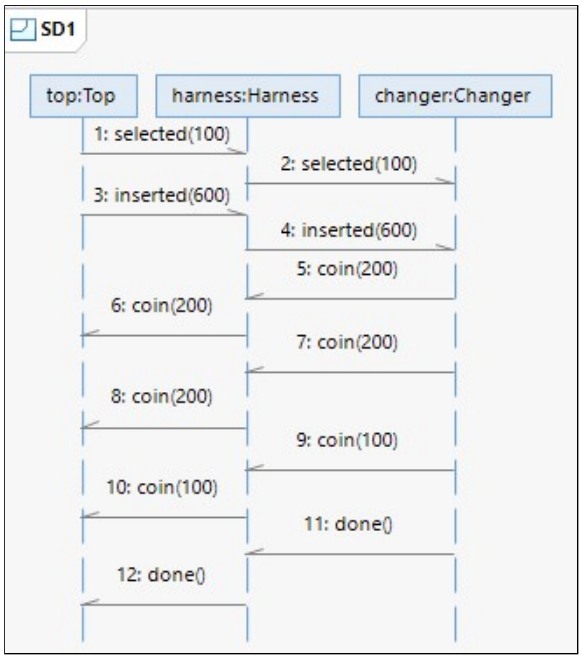
```
>>./executable -URTS_DEBUG=quit -UARGS <sel> <ins>
```

where `<sel>` represents the value of item (in cent) selected by the customer, and `<ins>` represents the amount of money (in cent) inserted. So, e.g., the invocation

```
>>./executable -URTS_DEBUG=quit -UARGS 900 2000
```

should cause the changer (no matter which version is used) to dispense five 200-cent coins (i.e., five 'toonies') and one 100-cent coin (i.e., one 'loonie'), terminate successfully and not report any errors.

2. **Task description:**
   - **Task 1** [3 points]: Consider the following sequence diagram '`SD1`'.



     Which of the three system versions have an execution that is consistent with this sequence diagram? I.e., list all the system versions that have at least one execution along which the messages '`selected`', '`inserted`', '`coin`', and '`done`' are ordered as shown in '`SD1`'. Create another text file in your project from Part I, name it '`Part2.txt`', and put your answer in it.
   - **Task 2** [3 points]: For each of the three system versions ('`GiveChangeV1`', '`GiveChangeV2`', and '`GiveChangeV3`') consider the execution caused by the invocation

```
>> ./executable -URTS_DEBUG=quit -UARGS 100 600
```

     For each system, how long is the execution caused by this invocation, i.e., how many run-to-completion steps does the execution consist of? Put your answer into the same text file created in the previous task ('`Part2.txt`').

What to hand in

As for Assignment 1, export the project containing your final system (i.e., the model with the most functionality) and the two text files '`Part1.txt`' and '`Part2.txt`' into a .zip file and upload it to OnQ (again, no need to include the generated code into the export). Use '`[firstName]_[lastName]_A2_CISC836_F21.zip`' as name where '[firstName]' and '[lastName]' are replaced by your first and last names, respectively.

Marking

For Task 1 in Part I, your model will be marked based on their correctness and completeness (with respect to the assignment instructions and the system descripion), but also using the design guidelines discussed in class. You can find a summary of these guidelines on slides 68 to 70 of the UML-RT slide deck used in class, http://research.cs.queensu.ca/~dingel/cisc836_F21/slides/UML_RT_F21_Parts1To3_4up.pdf. Your answer to Task 2 in Part I will be marked based on appropriateness of each of the identified steps and the entire sequence, but also on the clarity of the writing.

Last modified: Sun Oct 03 2021 14:09:19