Empirical Study on Reactive Programming - Tasks

Moritz Dinser, Ragnar Mogk

TECHNISCHE UNIVERSITÄT DARMSTADT

1 Structure of Repository

To better find your way through the repository we provide you with a small overview:

- Each task has an own directory and can be found in: *empirical-studies-2021/src/main/scala*. Carry out each task in the respective file.
- Tasks descriptions can be found in this document, but are also available in the respective files in form of comments.
- Feel free to use the REScala documentation at any time: `https://www.rescala-lang.com/empirical-studies-20` `rescala/default$.html`

2 Tasks

Task 1: Warm-Up

You can implement this task in: empirical-studies-2021/src/main/scala/task1/readme.txt

In this task you will implement the functionality of a simple Todo-App in a language of your preference. You can assume that the UI and network of the Todo-App already exist and you are free to make assumptions about their implementation.

This task meant as a warm-up task for you. Furthermore, it helps us to understand what your expectations and assumptions on preexisting features are and what language and programming paradigms you choose if given the choice.

a) Implement a list containing all previously added to-dos.

b) Implement the functionality that adds a new to-do to the list.
The to-do will be entered over a *text input field* and will be added to the list after confirming the input with a *button*.
You can assume that all user inputs from the UI are directly available to you and don't need to be implemented.

c) Implement the functionality to synchronize all added to-dos amongst other devices in a network.
You can assume that network functionalities and devices are available to you and don't need to be implemented.

Task 2: Event Expressions

Task description can also be found in: empirical-studies-2021/src/main/scala/task2/EventExpressions.scala

REScala has event expressions *Event*{ } as a generic mechanism to declare events, but also provides many specific combinators for often used functionality.
Event expressions can be used to define the same semantics as combinators.

Your task is to state which combinators (c1-c13) correspond to which event expressions (d1-d13). There are also 4 extra expressions (e1-e4) that are alternative forms of the above, also assign those.

You may use *assertEquals(cN, dN)* to test you assumption.

Task 3: Fold Expressions

Task description can also be found in: empirical-studies-2021/src/main/scala/task3/FoldExpressions.scala

Assume you ingress events from the monitor mode of a wifi chip.
The system should observe traffic for every source address in between matching start/stop events.

Compute the following values (as signals) resetting every time the timer triggers

- The average frame size for each source address
- The number of frames for each source address
- The total size of data per source address
- The bytes per second per source address (the timer provides the passed time in milliseconds)

When the timer triggers, output all three values.

Note: try using "Events.foldAll" to combine multiple events into a signal.

Task 4: Chat Application

Task description can also be found in: empirical-studies-2021/src/main/scala/task4/ChatApp.scala

This application models a very simple chat application where each user may enter a name, and then send messages which are replicated to all other devices.
In the current version, the two definitions in line 61 and 65 create a *Chatline* for each new message a user enters. A chatline is just the message together with the date and author of the message.
Then, all chatlines are collected into a history. The history is an RGA, which is an expensive replicated data structure that ensures correct order.
However, you note that each chatline has an associated data, and chat messages could just be ordered by that date, thus not requiring the cost of the RGA.

a) Replace the *RGA[Chatline]* in the history with a simple *List[Chatline]*. You may assume that the creation date of new messages on the current device is always ascending. You may also assume that clocks between different devices are reasonably synchronized.

   To enable replication of your new list of chatlines, the system requires an implicit instance of *Lattice[List[Chatline]]*. Implement that instance ensuring that the merge method is associative, commutative, and idempotent.

b) It's annoying, that the chat history is lost every time the browser window is restarted!
   Fix that by storing and loading the history from local storage.

   You may use the following constructs for reading and writing objects of type 'A' to local storage:

   - *readFromString[A](dom.window.localStorage.getItem(key))*
   - *dom.window.localStorage.setItem(key, writeToString(ft))*

Task 5: To-do-App

Task description can also be found in: empirical-studies-2021/src/main/scala/task5/Todolist.scala

This to-do-list application is a full implementation of a common case study for interactive applications.
Your task is to document the dataflow in this application by creating a diagram (either in text or as an image) of how interactions (such as user adding new tasks) travel through the program.
Target the level of detail of your description to someone else who wants to understand this particular codebase.

Consider using `https://www.yworks.com/yed-live/` to draw the graph.