UNIVERSIDADE FEDERAL DE PERNAMBUCO
Centro de Informática
Ciência da Computação

Vinícius Emanuel M. Silva

Um Comparativo Entre Técnicas de Processamento de Streamings em Swift

RECIFE
2018

Vinícius Emanuel M. Silva

Um Comparativo Entre Técnicas de Processamento de Streamings em Swift

Trabalho de Conclusão de Curso submetido à Universidade Federal de Pernambuco, como requisito necessário para obtenção do grau de Bacharel em Ciência da Computação.

Autor: Vinícius Emanuel M. Silva
(vems@cin.ufpe.br)
Supervisor: Kiev Santos da Gama
(kiev@cin.ufpe.br)

Recife, dezembro de 2018

"Porque cada um conhece o seu limite."
Leal, Victor

Agradecimentos

Obrigado a todos que me acompanharam durante todo o caminho. Obrigado Erik e Daniel, minhas primeiras influências em computação. Obrigado Joseildo, professor e pai, queria lhe dizer que hoje em dia passo mais de uma hora no computador. Obrigado Norma, mãe, queria dizer que já pode parar de perguntar quando, porque finalmente estou me formando aqui nessa cesta do alto do mastro das caravelas. Obrigado a Josinaldo e Fabiola, meus segundos pai e mãe por muito tempo. Obrigado a equipe do Voxar Labs que me guiou nos primeiros passos da vida acadêmica, VT, Joma, Chico, Lucas, Mozart e as outras pessoas com quem trabalhei. Obrigado a todos do klwcin pelos cafés tomados, noites viradas e risadas dadas. Obrigado aos professores do BEPID, Formiga, Francisco, Castor, Kiev e Pedro, que me deram a base do que é meu trabalho hoje. Obrigado também aos meus amigos que me ouviram reclamar da vida e que não me viram com a frequência que eu gostaria durante o desenvolvimento desse trabalho.

Resumo

Cada vez mais estão surgindo fluxos de dados cada vez mais dinâmicos e a necessidade de ferramentas capazes de lidar com esses fluxos. Esses dados podem vir das mais variadas fontes em vários formatos e as ferramentas que lidam com isso devem ser capazes de interpretar e combinar esses fluxos para compreender o contexto do que está acontecendo e reagir de acordo.

São muitas as formas que as ferramentas escolhem para trabalhar com streamings de dados. Podemos citar, por exemplo, Linguagens Reativas (RL) que foca nos fluxos de mudanças de valores e Processamento de Eventos Complexos (CEP) que foca em tratar todos os elementos dos fluxos como um evento e trabalhar em cima desses eventos.

No contexto mobile existe um conjunto de ferramentas capazes de trabalhar com esses fluxos de dados que podem vir de várias fontes como interações do usuário, leitura de sensores, atualizações de status do smartphone dentre outras. Este trabalho faz um comparativo entre duas ferramentas para tratamento de streamings no ecossistema iOS, CEPSwift e RxSwift que focam em CEP e RL respectivamente.

Palavras-chave: Eventos Complexos, Linguagens Reativa..

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CEP	Complex Event Processing
RL	Reactive Languages
API	Application Programming Interface
PSSUQ	Post-Study System Usability Questionnaire
UI	User Interface

Sumário

1 Introdução

Cada dia mais estão surgindo diversas demandas para processar fluxos de informações em diversos contextos e aplicações. Esses fluxos podem vir das mais variadas fontes e em ritmos não previsíveis.

No contexto de aplicações móveis, essas sequências de dados podem ser criadas de várias fontes, tanto externas ao aparelho como internas, podendo ser desde interações do usuário com a aplicação até a leitura de sensores ou atualizações de status do próprio aparelho. Existem várias formas de conseguir criar aplicações para trabalhar com esses fluxos de dados, e dentre essas podemos citar Programação Reativa (RL) e Processamento de Eventos Complexos (CEP).

A Programação Reativa foca em reagir à mudança nos valores de variáveis. A ideia é observar uma variável e, sempre que seu valor for atualizado, realizar uma ação relacionada a esse novo valor. Imagine um campo de busca: existe uma variável que guarda o valor desse campo, e sempre que o campo é atualizado, a variável é modificada. Da mesma forma, sempre que ela é modificada, uma ação de busca é realizada e o resultado é exibido para o usuário que digitou a busca.

Já em um sistema de Processamento de Eventos Complexos, tudo é tratado como um evento, desde mudança de valores, até atualizações de status, leituras de sensores, interações com a aplicação e assim por diante. A ocorrência de eventos pode gerar ações relacionadas a esse evento ou criar novos eventos, e eventos podem também ser combinados entre si para agir como um evento único e assim mais uma vez gerar novos eventos ou ações. No mesmo contexto de campo de busca, imagine o evento de mudança no campo de busca gerando o evento que cria a ação de busca e o resultado desse gerando o evento que mostra o resultado para o usuário.

Dentre as várias opções para trabalhar com streamings de dados no ecossistema Apple, este trabalho foca em dois frameworks, RxSwift e CEPSwift, que são ferramentas feitas em Swift para soluções utilizando Linguagens Reativas e Processamento de Eventos Complexos, respectivamente. Este trabalho é o primeiro a fazer um comparativo inicial sobre o aprendizado de CEP e programação reativa.

No capítulo de Contextualização deste trabalho é explicado o que são Eventos e Eventos Complexos, o que é uma Linguagem Reativa e algumas diferenças entre CEP e RL. No capítulo seguinte são descritos os trabalhos usados como base no desenvolvimento da avaliação aqui feita. Em seguida, é descrito como foi feita a introdução ao tratamento de streaming a um grupo de alunos, como os conceitos de CEP e RL foram apresentados a eles, como CEPSwift e RxSwift foram demonstrados e como a interação dos alunos com os

frameworks foi avaliada. Então, é descrito todo o procedimento que ocorreu com os alunos e os resultados da avaliação. São apresentadas também as possíveis ameaças à validade desse trabalho e, finalmente, no último capítulo é dada uma conclusão sobre o trabalho, apresentadas as dificuldades enfrentadas e são descritos alguns trabalhos que podem ser iniciados a partir do que foi aqui feito.

2 Contextualização

Este capítulo fala sobre os tópicos que definem o que é o processamento de eventos. Primeiro será definido o que é um evento, e depois será apresentada a arquitetura orientada a eventos. Com a arquitetura definida, serão apresentadas uma visão sobre o Processamento de Eventos complexos (CEP) e Linguagens Reativas (RL). E, finalmente, com a definição de CEP e RL, serão feitas algumas comparações entre esses modelos.

2.1 Evento

A palavra "Evento" é usada para representar, em um sistema computacional, a ocorrência de uma entidade [Pillet 2017]. Eventos podem ser, por exemplo, mudanças de valores de uma variável, interações de usuários com a aplicação ou leituras recorrentes de sensores. Tudo o que acontece dentro de uma aplicação pode ser representado como um evento, e todo evento tem uma propriedade temporal associada a ele, e a ordem de ocorrência é definida por essa propriedade.

Eventos podem interagir entre si, de forma que a combinação de um ou mais deles podem gerar outros. A ocorrência de eventos pode disparar processos que vão desdes pequenas alterações ou todo um caso de regra de negócio [Michelson 2006]. Essa interação entre eventos e processos é chamada de Arquitetura orientada a Eventos.

2.2 Arquitetura orientada a Eventos

A Arquitetura orientada a eventos é uma arquitetura de software onde programas são construídos baseados na ocorrência de eventos. Essa arquitetura pode ser dividida em quatro componentes lógicos, cada um deles com suas próprias responsabilidades para que se possa ter uma boa modularização [Michelson 2006].

- **Event generator:** Esse componente é o responsável por acompanhar a fonte de dados de interesse e gerar o evento a partir dela, fazendo, se necessário, transformações nos dados vindos da fonte.
- **Event channel:** Esse componente é responsável por transportar o evento criado até a camada de processamento.
- **Event processing:** Esse componente recebe o evento e o avalia segundo regras definidas para tomar medidas apropriadas de acordo com o evento e as regras.

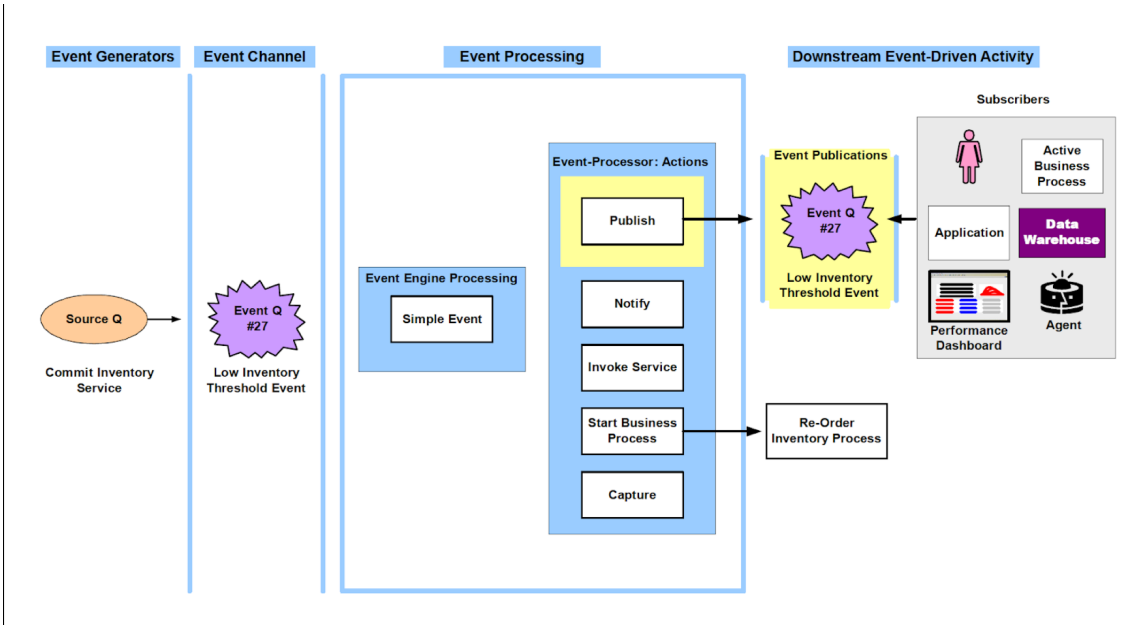

Figura 1 – Fluxo de processamento de um evento simples e seus componentes lógicos [Michelson 2006].

- **Downstream event-driven activity:** Um evento pode afetar uma ou mais partes interessadas na sua ocorrência. A interação dos componentes que serão afetados pelo evento pode ocorrer como uma ação de Pull ou de Push do evento com os componentes. Em uma ação de Push, a ocorrência do evento realiza a mudança nos componentes interessados na mudança, usando um callback, por exemplo. Em uma ação de Pull, o evento notifica os componentes interessados. Nesse caso, o componente interessado na ocorrência da mudança observa quando essa acontece para depois realizar sua atualização.

A figura 1, representa o fluxo do tratamento da ocorrência de um evento no sistema de estoque de uma livraria. No **Event generator**, uma ação é disparada, gerando um evento de "low inventory" no **Event channel**. No **Event processing**, ações são tomadas baseadas no evento recebido, sendo mostradas no **Downstream event-driven activity**. Um novo evento, desta vez de regra de negócio, é criado com ação de Push e elementos interessados na ocorrência do evento de estoque são notificados da sua ocorrência como ação de Pull.

Uma arquitetura orientada a eventos tem como característica marcante ser extremamente modularizada e desacoplada, já que tudo é tratado como um evento, e produtores e consumidores desses eventos não precisam se conhecer para realizarem suas funções [Guedes 2017]. Nesta arquitetura, existem três formas básicas de se tratar eventos, sendo elas não exclusivas: **Simple Event Processing**, **Complex Event Processing**, e **Stream Event Processing**.

- **Simple Event Processing:** Quando um evento ocorre uma única vez sem a interação com outros eventos e gera uma ou mais ações.
- **Complex Event Processing:** Quando um evento ocorre uma única vez, sendo gerado pela interação de diferentes eventos, e uma ou mais ações são disparadas.
- **Stream Event Processing:** Quando um evento, seja complexo ou não, ocorre continuamente, e uma ou mais ações são disparadas sempre que esse evento ocorre.

Existe uma série de formas de se implementar uma arquitetura orientada a eventos. Duas delas são Complex Event Processing (CEP) e Reactive Languages (RL), e o restante deste capítulo trata de definir esses dois métodos de se trabalhar com Eventos.

2.3 Processamento de Eventos Complexos

Convencionalmente, no tratamento de eventos complexos, uma série de eventos ocorrem no mundo externo, e esses devem ser filtrados e combinados para que se possa entender o que está acontecendo em um nível de abstração mais elevado [Cugola e Margara 2012]. O foco desse modelo é detectar, em baixo nível de abstração, a ocorrência de eventos particulares que, juntos, representam a ocorrência de um evento de mais alto nível, e assim notificar essa ocorrência.

Eventos podem se relacionar entre si de várias maneiras, como relação de tempo, agregação ou causa e consequência [Guedes 2017]. Essas relações são tratadas e definidas por um conjunto de regras que podem levar em consideração uma variedade de fatores, como contextos, similaridades, sequências e regras de negócio, para reagir de acordo.

As regras definidas podem ser separadas em dois tipos: **Deductive Rules**, que geram novos eventos baseados na interação ou na filtragem de eventos atuais; e **Reactive rules**, que definem como o sistema deve reagir ao evento, normalmente com a chamada de algum procedimento que deve ser executado [Eckert e Bry 2009].

Na figura 2 é possível ver um modelo de como o CEP funciona. O “Input” define um série de eventos, o “Pattern” define o que seriam as regras e o “Output” representa os “inputs” que se adequam às regras definidas.

CEP vem sendo amplamente usado em alguns contextos que geram fluxos variados de dados e que precisam de uma resposta rápida baseada nesse fluxo. Exemplo disso é a leitura de sensores que coletam dados do ambiente ao longo do tempo e muitas vezes leva em consideração dados passados para chegar a uma conclusão. Uma outra aplicação na qual

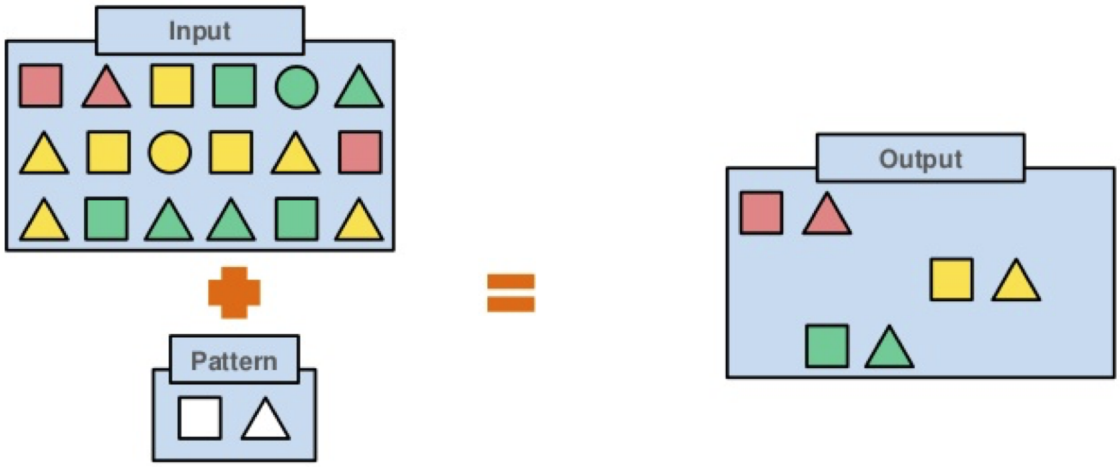

Figura 2 – Modelo lógico de CEP indicando a aplicação de regras aos eventos.

o CEP se encaixa muito bem é na bolsa de valores, onde acompanhar o fluxo dos dados é essencial para ter uma resposta adequada às variações o mais rápido possível [Eckert e Bry 2009].

2.4 Linguagens Reativas

RLs fornecem uma abstração para modelos de valores que mudam ao longo do tempo [Margara e Salvaneschi 2013]. Esses modelos podem ser valores de variáveis, um status de um botão, uma entrada de dados do usuário, um resultado de um processo assíncrono, e assim por diante. Um uso de RL pode ser encontrado nas aplicações de algumas redes sociais. No Twitter, por exemplo, existem contadores de "Likes", "Replies" e "Retweets" para cada novo post, contadores que são atualizados ao longo das interações com os usuários. Na aplicação mobile do Twitter, existe um componente de UI para cada contador, e, usando RL, esses componentes são atualizados automaticamente sempre que o valor que ele mostra é modificado [The introduction to Reactive Programming 2018]. Esses elementos de interface podem ser vistos na metade inferior da Figura 3.

RL estende o padrão **Observer** [Gamma et al. 1995], mas de uma maneira que facilita a leitura e a composição dos eventos [Margara e Salvaneschi 2013]. Pelo padrão Observer, RL funciona com dois elementos de base, o **Observable** e o **Observer**. O Observable é o objeto de interesse, é o fluxo ou a simples mudança que se deseja acompanhar. O Observer é aquele que acompanha o fluxo de mudanças e atualiza as variáveis que estão interessadas nesse fluxo, podendo as atualizações nessas variáveis serem objetos de interesse de outros Observer ou não.

Figura 3 – Print de um post no Twitter mostrando elementos de UI atualizados com uso de RL.

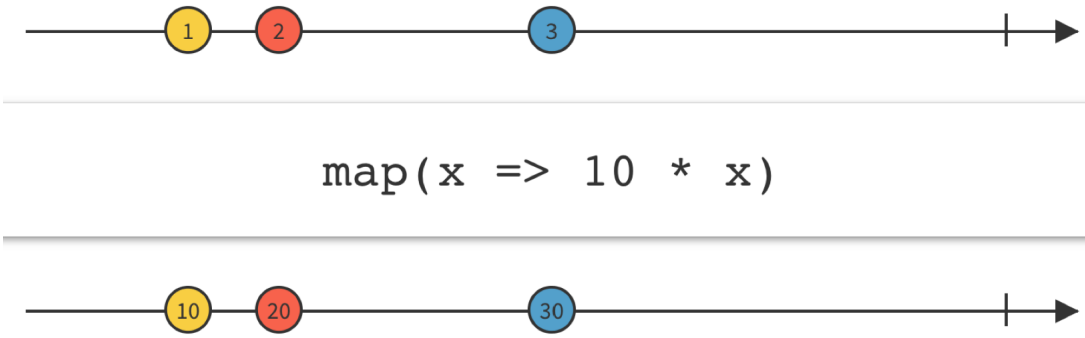

Figura 4 – Diagrama da função Map e os fluxos de entrada e resultantes indicados pelos valores da esquerda para a direita.

O Observer pode trabalhar livremente com os valores obtidos do Observable sem modificar seu fluxo. Dentre o que pode ser feito, podem ser citadas situações como alimentar outros Observables, atualizar valores em tempo real, ou até aplicar funções de alta ordem sobre o Observable e assim gerar um novo fluxo. Esta última aplicação pode ser vista na Figura 4, que demonstra a aplicação da função "Map" em um fluxo de dados, criando assim outro fluxo com os novos valores resultantes da função aplicada.

2.5 Comparando CEP e RL

Para comparar CEP e RL são levados em conta 5 passos no comportamento reativo: **Observation**, **Notification**, **Processing**, **Propagation** e **Reaction** [Margara e Salvaneschi 2013].

- **Observation:** É quando o fato de interesse acontece.
- **Notification:** Quando o fato de interesse acontece, ele é encapsulado em uma notificação.
- **Processing:** A notificação dispara uma ação.

- **Propagation:** O resultado da ação é propagado às partes interessadas.
- **Reaction:** As partes interessadas recebem o resultado da ação e agem de acordo.

Tanto RL quando CEP passam por esses cinco passos durante o ciclo reativo, mas com algumas diferenças em cada passo.

Na fase Observation em CEP, o objeto de interesse é genérico e pode ser, dentre varias coisas, leitura de sensor, uma atualização em banco de dados, um status de interação de usuário, uma sequência de ações e assim por diante. Em RL, o objeto de interesse é o fluxo de mudança de valores de objetos primitivo.

Na fase Notification em CEP, a notificação é explicitamente mandada para a fase de Processing, que tem um conjunto de regras que definem como trabalhar com as notificações recebidas para produzir o resultado esperado. Por outro lado, em RL, a fase de Notification é implícita e a fase de Processing é formada por uma expressão que, assim como a saída, é baseada no fluxo de mudança de valor.

Similar à fase Notification, a Propagation é definida explicitamente em CEP, implementada usando composição de operações e eventos, e implícita em RL. Em CEP, a fase de Propagation usa uma abordagem de Push para a fase de Reaction. Já em RL, é possível usar a abordagem Pull, onde a fase de Reaction só acontece quando algum componente requisita. Na fase Reaction, CEP não coloca nenhuma limitação e quem recebe o resultado tem total liberdade para trabalhar como achar necessário. Já em RL, na fase Reaction existe no mínimo uma mudança envolvida, e, já que a mudança de valor está associada a uma instância de tempo, mesmo que o valor não mude na Reaction, a instância de tempo é atualizada.

3 Trabalhos Relacionados

Como referências de métricas de avaliação, foram usados outros trabalhos com foco em comparativos na área de linguagens de programação. Dentre os trabalhos encontrados, podemos citar o de Wiedenbeck e Ramalingam [Wiedenbeck et al. 1999], que compara o uso, por iniciantes em programação, de uma linguagem orientada a objetos a uma linguagem procedural. Outro trabalho com foco comparativo é o de Acar e Fahl [Acar et al. 2017], que compara o uso de APIs de criptografia. Existe também o trabalho de Brandão [Lôbo 2015], que faz um comparativo do uso por dois grupos de alunos de uma solução para detecção de deadlocks.

Wiedenbeck e Ramalingam usam como objeto de estudo a implementação de dois programas, um menor e mais simples e outro grande e mais intricado, por grupos de alunos. Os alunos são iniciantes em programação, um grupo com Pascal como linguagem procedural e outro com C++ como linguagem orientada a objeto, devem implementar os dois programas usando a linguagem que estão aprendendo. O objetivo é comparar como os alunos de cada estilo de programação formam os modelos mentais e como é o entendimento deles sobre o problema.

Acar e Fahl comparam o uso de APIs de criptografia em Python. Para o estudo, vários desenvolvedores de Python foram convidados a desenvolver online uma série de pequenas soluções de criptografia usando um ambiente controlado pelo pesquisador com Jupyter [jupyter Notebook]. O objetivo é comparar usabilidade, eficiência, segurança e facilidade de uso.

Brandão compara uma nova versão e a versão anterior da implementação da classe ReentrantLock de Java para detecção de deadlocks. A avaliação do seu trabalho é feita comparando o uso das versões da classe por dois grupos de alunos, sendo um deles formado por alunos de graduação do Centro de Informática da Universidade Federal de Pernambuco e o outro de alunos de mestrado do Programa de Pós-Graduação do mesmo centro. Foram usados dois problemas para serem resolvidos usando as ferramentas de detecção de deadlocks, sendo o primeiro mais simples e o segundo mais complexo. O objetivo era que os alunos conseguissem localizar o deadlock de cada problema. A corretude do trabalho de cada aluno era medida por três critérios que se resumem em: “ponto de deadlock encontrado”, “métodos e chamadas contêm o deadlock encontrado” e “deadlock não encontrado”.

Os três trabalhos serviram de base para o planejamento da parte avaliativa desse trabalho, já que, até o momento, não existe na literatura trabalho que faça algum tipo de avaliação comparativa do uso de CEPSwift e RxSwift. O que foi levado em conta

em cada trabalho foi, a ideia de Wiedenbeck e Ramalingam, em que grupos de alunos implementarem uma solução depois compara-se como chegam a ideia da resolução usando as ferramentas dadas. A ideia de Brandão, de medir o tempo de resolução e considerar essa medida como métrica importante no estudo. A ideia de Acar e Fahl de entregar um ambiente pré configurado, com tudo que não faria parte do foco do trabalho, para aqueles que vão fazer parte da pesquisa.

4 Proposta e Método

A proposta deste trabalho é fazer um comparativo inicial entre RxSwift [RXSwift] e CEPSwift [CEPSwift], dois frameworks para processamento de streaming de dados em Swift. RxSwift é um framework feito em Swift [Swift] para trabalhar com streaming de dados usando com RL usando o padrão Observer. O framework atualmente se encontra na versão 4.4.0. RxSwift faz parte de um conjunto de ferramenta ReactiveX [ReactiveX], que implementa as mesmas ideias de processamento de streaming com RL em uma série de outras linguagens de programação. CEPSwift é uma framework para trabalhar com streaming em Swift usando o conceito de CEP, e atualmente se encontra na versão 0.1.0, sendo desenvolvida inicialmente por George Belo Guedes, no Centro de Informática da Universidade Federal de Pernambuco [Guedes 2017].

4.1 Piloto

Primeiramente, foi realizado um piloto com um grupo preliminar, composto por pessoas com mais de dois anos de experiência trabalhando com Swift. O piloto foi executado para encontrar melhorias na aula, no exercício e na forma de avaliação. Além disso, também estavam entre os objetivos buscar falhas na aula introdutória que pudessem comprometer o entendimento do conceito de streaming de dados, RL ou CEP; eliminar algum ponto que pudesse deixar a escolha da atividade a ser completada tendenciosa para CEPSwift ou RxSwift; avaliar o nível de complexidade das atividades dadas como opção, e medir o tempo médio necessário para executar a aula introdutória e completar a aplicação.

4.2 Atividade

Sobre a atividade, a ideia inicial era implementar o jogo Genius [Jogo Genius 2018], no qual, ao invés da interação com botões, o jogador iria usar a inclinação do smartphone para repetir a sequência indicada de inclinações para a direita, esquerda, para cima ou para baixo. O jogo seria composto pela interação do streaming da sequência que o jogador deveria replicar, o streaming dos dados da inclinação do aparelho, que viria do sensor, e um terceiro streaming, gerado pelo sensor, indicando a inclinação pelos movimentos do jogador. Os dados do streaming da sequência seriam disparados a cada segundo em uma função temporal, enquanto os dados do streaming da inclinação vindos do sensor seriam filtrados e mapeados em um outro streaming, que indicaria os movimentos do jogador. Esse último seria combinado com o streaming da sequência para validar os movimentos do jogador.

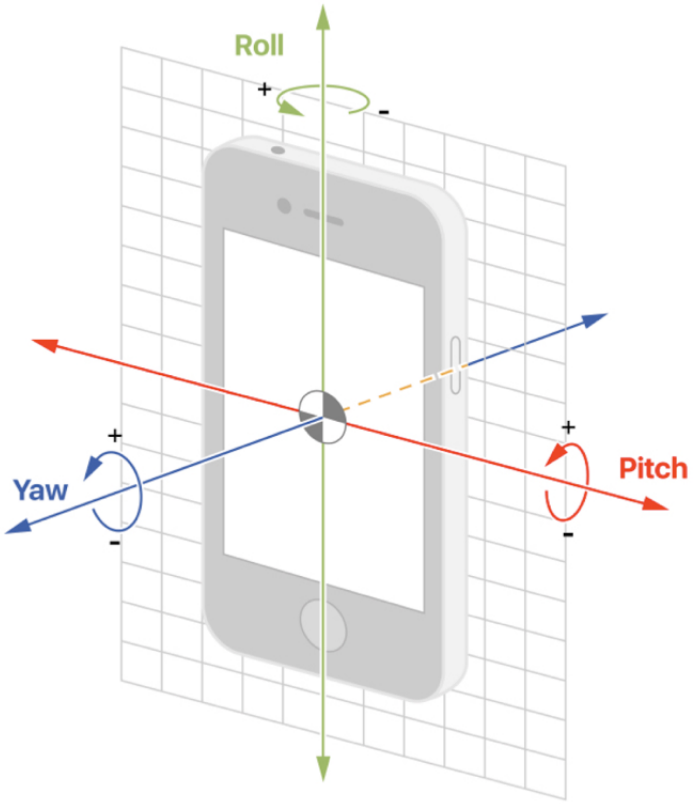

Figura 5 – Eixos de inclinação do smartphone.

Esse projeto pôde ser construído em RxSwift, mas não em CEPSwift, por limitações da atual versão do framework, que não implementa alguns operadores sobre o streaming que eram necessários para o projeto. RxSwift tem esses operadores implementados: um operador de tempo, que dispara um evento pré estabelecido a cada intervalo definido; e um operador de combinação, que recebe dois streamings e cria um terceiro que é disparado baseado na ocorrência dos dois recebidos. CEPSwift ainda não tem implementado nenhum operador de tempo com o intuito de gerar um evento depois de um intervalo definido e o único operador de combinação de streamings não externa os valores dos elementos combinados, apenas realiza a chamada de uma função anônima quandos os eventos acontecem ao mesmo tempo.

Já que a versão a atual do CEPSwift não permite a implementação do jogo completo apenas usando as funções para trabalhar com streaming desse framework, a ideia foi simplificada de forma que pudesse ser feita tanto em CEPSwift como em RxSwift, trabalhando com as funções disponíveis em ambos os frameworks. O novo projeto contém apenas a parte de interação com os movimentos do aparelho. O usuário pode inclinar o aparelho nos eixos de Pich e Yaw, como mostrado na Figura 5, e as variações nessa inclinação são tratadas como um fluxo de pontos, onde valor dado em Yaw vai para o X e o valor dado em Pitch vai para o Y.

Os valores dados podem variar de -1 a +1, onde -1 e +1 em X são as inclinações

Figura 6 – Sequência de variações de inclinações.

máximas para a esquerda e direita, respectivamente, e -1 e +1 em Y são as inclinações máximas para baixo e para cima, respectivamente. O fluxo de pontos que a inclinação gera é exibido na tela, depois mapeado em outro fluxo com as indicações de que lado o aparelho foi inclinado, e essa informação é também exibida na tela. A figura 6 mostra uma sequência de telas exibindo valores de inclinação em X e Y e seus indicativos de inclinação, onde A, B, C, D e E são respectivamente o aparelho plano, inclinado para esquerda, para baixo, para direita e para cima.

Como o objetivo da atividade era o mesmo - tratar o streaming dos dados da inclinação - tanto usando RxSwift como usando CEPSwift, tudo o que não envolvia diretamente o streaming dos dados já foi previamente implementado para os alunos. A configuração de acesso aos dados do sensor foi a mesma, tanto para o projeto em CEPSwift como em RxSwift, com a diferença que em um o fluxo de pontos é criado em um **EventManager**, Figura 7(A), e no outro em um **PublishSubject**, Figura 7(B), respectivamente.

O PublishSubject é uma classe de RxSwift responsável por criar um Observable com o fluxo de pontos e o EventManager é a classe responsável por criar e gerenciar o fluxo

```
self.manager.startDeviceMotionUpdates(to: OperationQueue.current ?? OperationQueue.main) {
    (motion, error) in
    if let motion = motion{
        let point = (motion.gravity.x, motion.gravity.y)
        self.motionEventManager.addEvent(event: MotionEvent(data: point))
    }
}
```

A: Recebendo informações do sensor de inclinação com CEPSwift

```
self.manager.startDeviceMotionUpdates(to: OperationQueue.current ?? OperationQueue.main) {
    (motion, error) in
    if let motion = motion{
        let point = (motion.gravity.x, motion.gravity.y)
        self.publishPoint.onNext(point)
    }
}
```

B: Recebendo informações do sensor de inclinação com RXSwift

Figura 7 – Criação do streaming de pontos com os dados da inclinação.

```
class SideEvent: Event {
    var timestamp: Date
    var data: Side

    init(data: Side) {
        self.data = data
        self.timestamp = Date()
    }
}
```

A

```
class MotionEvent: Event {
    var timestamp: Date
    var data: Point

    init(data: Point) {
        self.data = data
        self.timestamp = Date()
    }
}
```

B

Figura 8 – Classes que representam eventos em CEPSwift; A: SideEvent para o sentido da inclinação; B: MotionEvent para o valor da inclinação.

de eventos em CEPSwift. CEPSwift precisa de uma classe que implemente o protocolo **Event** e tenha o dado do evento e uma instância de tempo. No projeto em CEPSwift existe uma classe chamada **SideEvent** que faz esse papel de abstração para o streaming de informações dos lados Figura 8(A) e **MotionEvent**, Figura 8(B), que faz esse papel de abstração para o streaming de pontos.

Finalmente, o streaming do fluxo de pontos que indica a inclinação do aparelho é processado. Nesse ponto, tanto em RxSwift como em CEPSwift, tudo o que se faz é exibir na tela o valor dos pontos no streaming à medida em que o fluxo de dados acontece, ver Figura 9. É nessa parte que os alunos devem complementar o exemplo à sua escolha para que, além de exibir na tela os valores das inclinações em X e Y, ele também mostre a direção da inclinação.

```
self.subscribePoint = self.publishPoint.asObservable()
subscribePoint.subscribe {(value) in
        if let value = value.element{
            print(value)
            self.labelX.text = String(format: "%.2f",value.x)
            self.labelY.text = String(format: "%.2f",value.y)
        }
    }.disposed(by: self.disposeBag)
```

A

```
self.motionEventManager.asStream().subscribe { (motionEvent) in
    let point = motionEvent.data
    self.labelX.text = String(format: "%.2f",point.x)
    self.labelY.text = String(format: "%.2f",point.y)
    print(point)
}
```

B

Figura 9 – Exibindo na tela os dados de inclinação; A: RxSwift; B CepSwift.

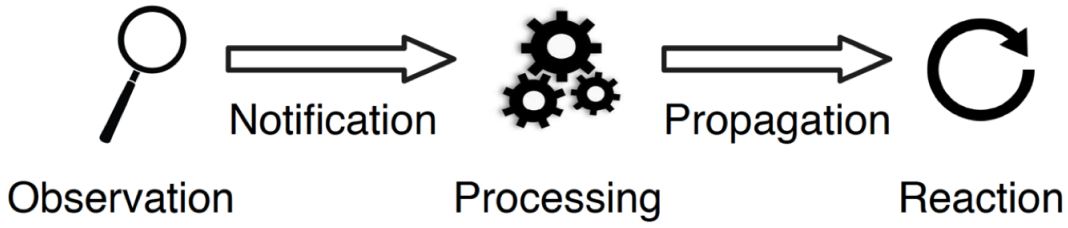

Figura 10 – Etapas do trabalho com streaming.

4.3 Aula introdutória

A aula antes do exemplo tem o objetivo de mostrar conceitos iniciais de fluxo de dados, linguagem reativa e eventos complexos. Mostra inicialmente a ideia de fluxo de dados, depois mostra dois exemplos de aplicações, sendo um focado em linguagem reativa e o outro em eventos complexos, indicando que RL é mais focado em fluxo de mudanças de valores enquanto CEP é voltado para fluxos mais gerais. Em seguida, são introduzidos aos alunos CEPSwift e RxSwift, mostrando as cinco etapas de trabalho com streaming que ambos os frameworks realizam, ver Figura 10, e falando sobre a diferença na etapa de Observation.

Finalmente, são mostrados exemplos de streamings nas duas abordagens. Primeiro é mostrado um exemplo de um streaming simples, apenas um fluxo de valor de inteiros feito tanto em RxSwift como em CEPSwift. Depois são mostradas duas funções de alta ordem, Map e Filter, que podem ser aplicadas aos streaming para transformações e filtragens, e

geram como resultado outro streaming. Essas funções estão presentes tanto em CEPSwift como em RxSwift e podem ser usadas na resolução do exercício, sendo aplicadas ao streaming de pontos.

4.4 Formulário de avaliação

Após os alunos terminarem de complementar o projeto de exemplo escolhido, foi pedido que os mesmos preenchessem um formulário. Esse formulário teve o intuito de avaliar o entendimento de programação com streaming, do framework escolhido para o exercício e de saber o porquê desse framework ser o escolhido.

O formulário foi feito no Google Forms [Google Forms 2018] usando como referência o PSSUQ (Post-Study System Usability Questionnaire) da IBM [Post-Study System Usability Questionnaire 2018]. O PSSUQ tem uma lista de dezenove pontos referentes à usabilidade de sistemas, dos quais alguns foram considerados como irrelevantes para montar o formulário, como avaliação de mensagens de erro e interfaces, enquanto outros foram adaptados para a avaliação desejada, como avaliação de usabilidade e curva de aprendizagem. Nas perguntas para as quais a resposta era em escala, foi usada como base a escala Likert [Likert scale 2018].

5 Avaliação e Resultado

5.1 Avaliação

A avaliação foi feita em duas rodadas com dois grupos de alunos, sendo a primeira rodada um piloto com um grupo preliminar e a segunda, após modificações da aplicação do piloto, com um grupo maior. Usar um grupo preliminar em um piloto tinha como objetivo, estimar a média do tempo da atividade, que seria o tempo de aula e o tempo para os alunos completarem a atividade, a complexidade da aula e a do projeto de exemplo. As informações de avaliação vêm dos dados preenchidos no formulário, o tempo para completar o projeto de exemplo e a corretude da implementação. A corretude é medida com base no estado de acoplamento da solução após os alunos finalizarem o que é pedido.

A medida de acoplamento é feita com os seguintes critérios: se a análise dos dados da inclinação e a indicação de que lado o aparelho está inclinado são feitos no mesmo streaming, então a solução está acoplada; se a análise dos dados e a indicação das direções da são feitas em streamings diferentes, então a solução está desacoplada. A solução ideal seria aquela na qual o streaming com os dados cria um segundo streaming, da maneira que o aluno achar melhor, com as indicações da inclinação e esse segundo streaming é responsável por exibir essa informação na tela.

5.2 Resultados do Piloto

O piloto foi aplicado com um grupo preliminar composto por quatro alunos. A atividade foi realizada em uma sala de reunião no Centro de Informática da Universidade Federal de Pernambuco. Todos tinham mais de dois anos de experiência trabalhando com Swift. Três deles já tiveram contato com programação reativa, mas apenas com RxSwift, e para um deles foi o primeiro contato.

Sobre a medida de tempo da atividade, a aula introdutória levou cerca de 15 minutos e o tempo máximo que um dos alunos levou para completar a atividade foi de 42 minutos. Sobre a complexidade da aula, foi considerada simples e de fácil compreensão pelo grupo, mas alguns pontos foram levantados já que no grupo maior todos os alunos têm uma média de tempo de 6 meses a um ano de experiência com Swift e, para alguns deles, esse tempo coincide com o primeiro contato com programação.

Já sobre a complexidade do projeto, a ideia de como trabalhar com streaming ficou clara para a maioria dos membros do grupo, mas um deles teve dificuldade de entender como trabalhar com streamings na prática e não conseguiu completar o exemplo. Para ele,

a dificuldade de entender a ideia de programação reativa foi a "a forma de aplicação e um pouco da semântica usada", conforme ele explicitou no questionário de feedback.

Como resultado do trabalho com esse primeiro grupo, alguns pontos foram modificados ou complementados para a atividade com o grupo maior. Na aula introdutória foi adicionado mais conteúdo de introdução a streaming de dados e exemplos de aplicação de RL e CEP, além de mais um exemplo de streaming básico feito em RxSwift e CEPSwift. Com essas mudanças, o tempo estimado da aula ficou em torno de 20 minutos. Somando a estimativa de 50 minutos para a completar o exercício e o tempo de preencher o formulário, a estimativa de tempo da aplicação da atividade ficou em torno de uma hora e meia, enquanto o tempo máximo disponível com a turma seria de duas horas.

5.3 Grupo de Avaliação

O grupo de avaliação foi composto por 21 alunos do curso de extensão voltado para o desenvolvimento de aplicativos iOS da Universidade Federal de Pernambuco. A atividade aconteceu em dois momentos: primeiramente no turno da manhã, com 10 alunos, e num segundo momento no turno da tarde, com mais 11 alunos. Já que existe um turma do programa em cada turno, a atividade foi replicada nas duas turmas, mas sem que uma tivesse contato com a outra.

Dos 21 alunos que fizeram parte do estudo, todos têm em média um ano de experiência com Swift e apenas 4 deles tinham alguma experiência prévia trabalhando com streaming, como pode ser visto nos gráficos da Figura 11 e da Figura 12. Ambas as turmas são formadas tanto por pessoas que já tiveram ou têm contato com outras linguagens e conceitos de programação, quanto por pessoas que estão tendo seu primeiro contato com programação.

Durante as duas aulas os alunos se organizaram entre si em grupos, e dentro dos grupos, se ajudaram na compreensão do assunto e complementação do exemplo. Em cada aula havia dois alunos que já tinham experiência em trabalhar com alguma ferramenta de streaming de dados: na turma da manhã, ambos com React, e na turma da tarde, um deles com React e outro com RxJS. A proximidade e interação de outros alunos a esses que já tinham algum conhecimento de RL fez com que a compreensão dos conceitos de linguagem reativa fosse maior do que a dos conceitos de CEP, como pode ser visto no gráfico da Figura 13.

Apesar da influência dos que já trabalharam com RL, a escolha do framework para completar o exemplo não pendeu para o lado de RxSwift, como mostrado na Figura 14.

No formulário foi pedido aos alunos que comentassem a escolha do framework. Tanto de pessoas que escolheram CEPSwift quanto de pessoas que escolheram RxSwift

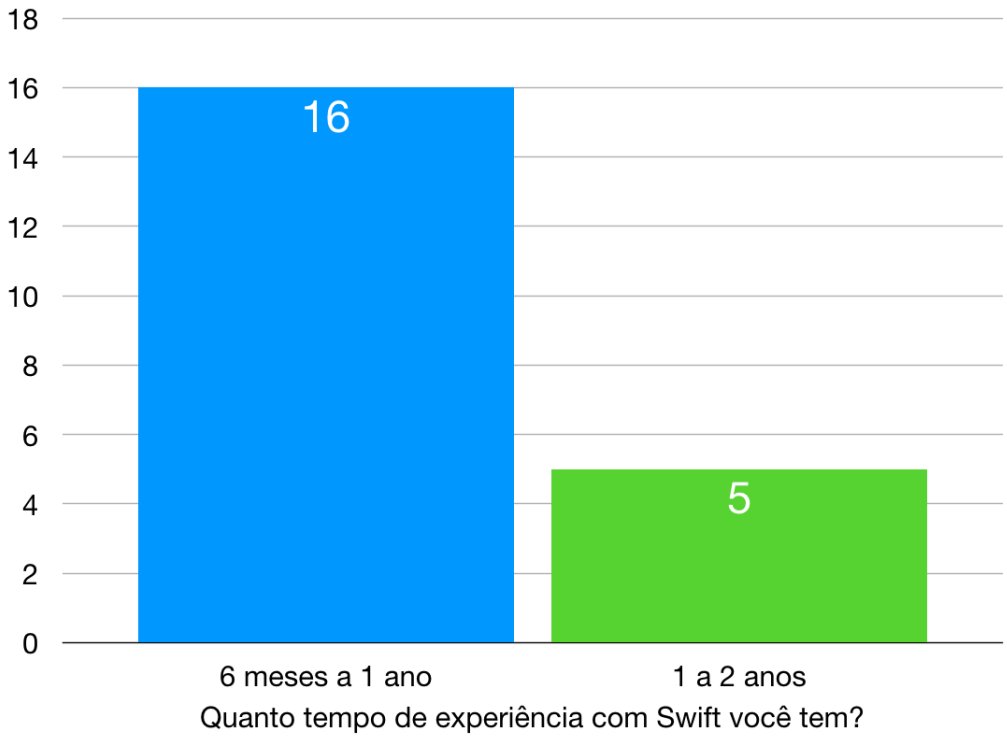

Figura 11 – Grafico sobre o tempo de experiência com Swift.

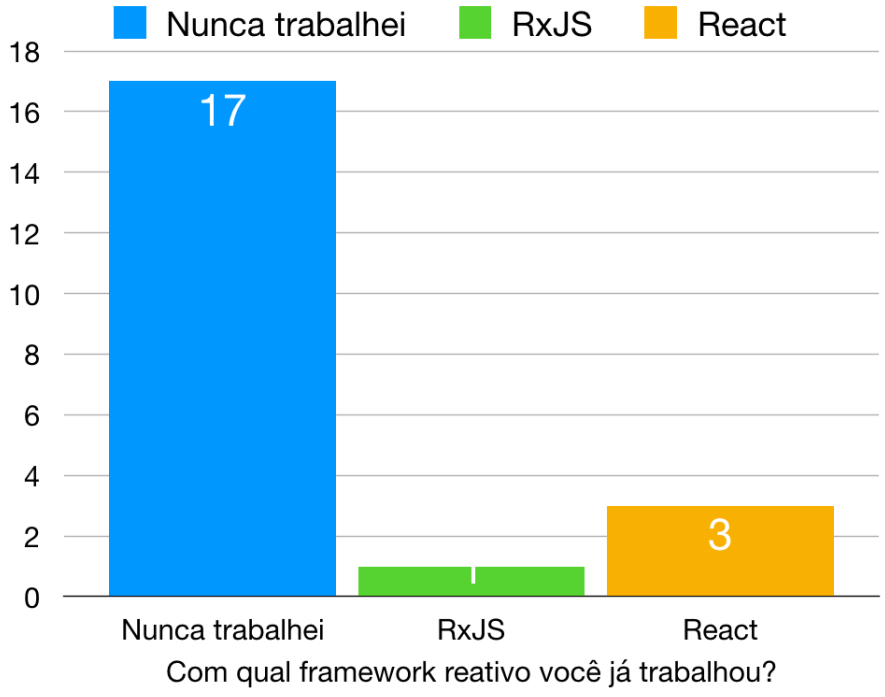

Figura 12 – Grafico sobre experiência com ferramentas de streaming.

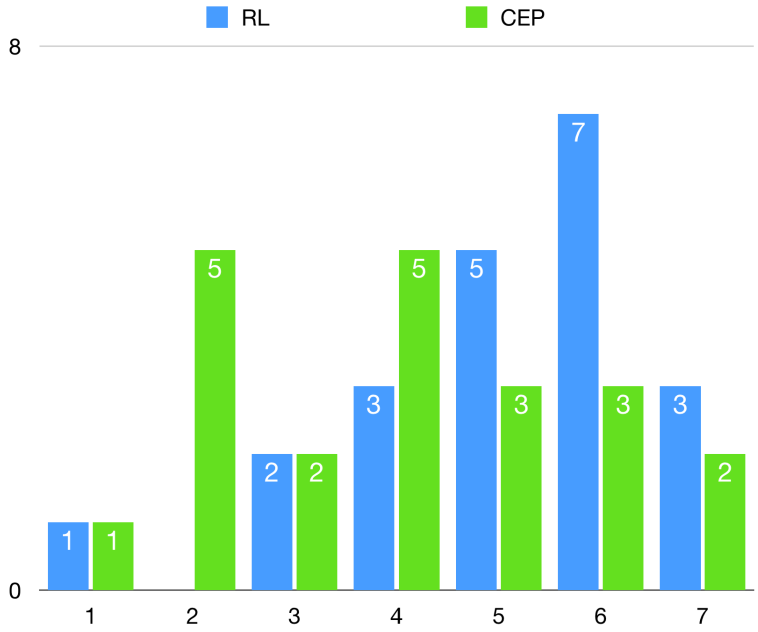

Figura 13 – A definição de RL/CEP ficou clara. (1 discordo totalmente, 7 concordo fortemente) do quão claro ficou cada definição.

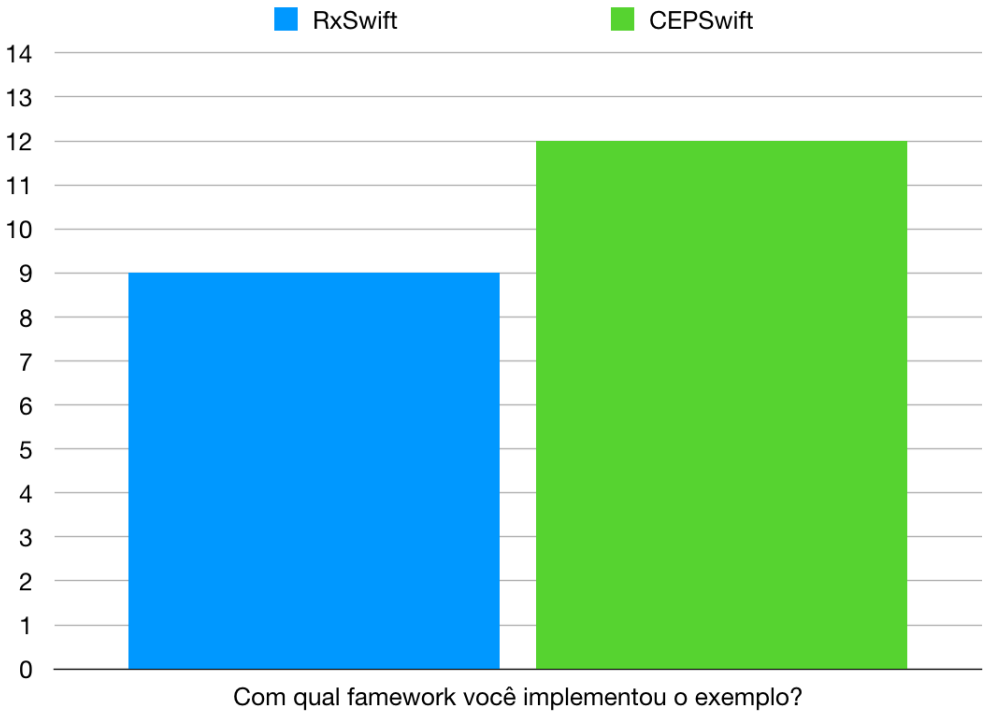

Figura 14 – Gráfico sobre as escolhas do frameworks.

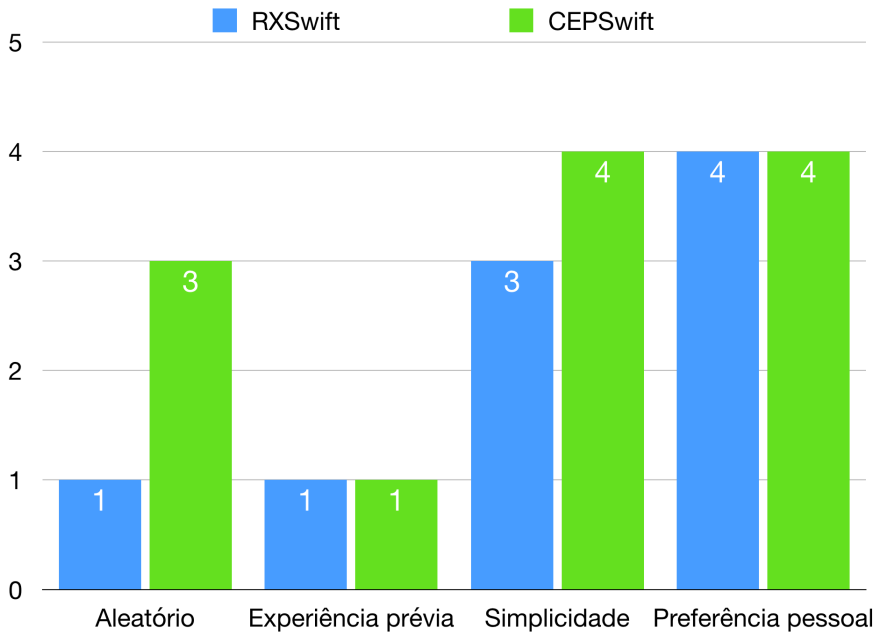

Figura 15 – Indicativo das escolhas.

	CEPSwift	RxSwift
Média	30 min 41 seg	30 min 44 seg
Desvio Padrão	11 min 56 seg	9 min 41 seg

Tabela 1 – Média e Desvio Padrão do tempo dos alunos.

havia comentários indicando que a escolha foi aleatória, mas para os dois frameworks havia também comentários que destacam a simplicidade de ambos. Assim, pelos comentários as escolhas foram classificadas em "Aleatória", "Experiência Prévia", "Preferência Pessoal" e "Simplicidade". O resultado pode ser observado no gráfico da Figura 15.

Sobre a atividade para completar o projeto exemplo com o framework escolhido, a média de tempo dos que escolheram RxSwift e CEPSwift pode ser vista na Tabela 1. Ambas as turmas com aproximadamente 20 minutos abaixo do tempo estimado para a resolução dessa etapa, mas mesmo assim três alunos não conseguiram concluir o exercício, dois alunos no grupo que escolheu RxSwift e um no que escolheu CEPSwift.

Esses alunos tiveram dificuldade de compreensão do conteúdo por falta de uma base mais sólida de programação. Mesmo todos tendo o mesmo tempo de experiência com Swift, alguns deles têm mais tempo de experiência com programação usando outras linguagens enquanto que para outros esse é o primeiro contato com programação. Os gráficos com os tempos podem ser vistos na Figura 16 e na Figura 17.

Sobre a corretude das implementações dos alunos, quase todos eles fizeram o código com acoplamento - apenas cinco alunos separaram em dois streamings os dados do sensor

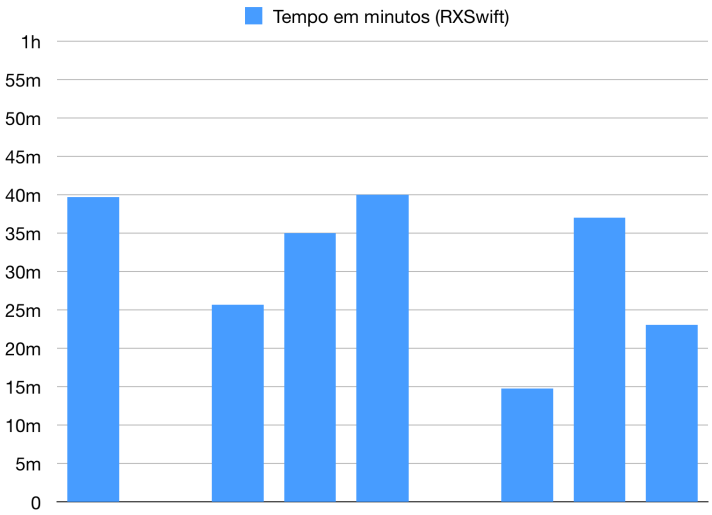

Figura 16 – Tempo de execução do exerício dos alunos que escolheram RX.

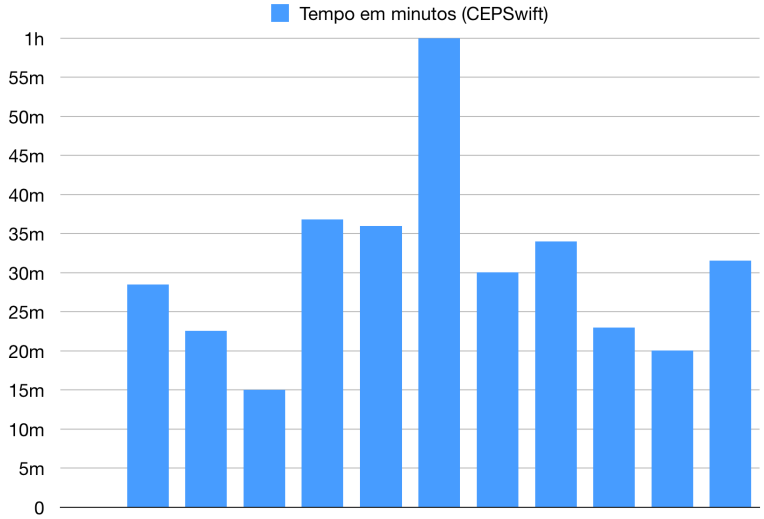

Figura 17 – Tempo de execução do exercício dos alunos que escolheram CEP.

das informações das inclinações do aparelho. Três alunos não conseguiram implementar a solução. As informações de corretude das respostas estão no gráfico da Figura 18.

Ao se observar os dados coletados dos alunos, é possível dizer que CEPSwift atraiu mais interesse dos alunos do que RXSwift, apesar de várias pessoas terem alegado compreender melhor RL do que CEP. Observando dados como, número de alunos que escolheram esse framework, motivos da escolha e tempo de resolução do exemplo pedido na atividade alguns pontos se destacam. Sobre as escolhas, alguns alunos que já tinham experiência com RL escolheram trabalhar com CEP por curiosidade, como se vê nos comentários no formulário sobre a escolha como "*Curiosidade, tentar entender melhor, pareceu mais interessante*". Sobre o tempo, CEPSwift e RxSwift tem uma média muito próxima, mas CEPSwift tem um desvio padrão mais elevado.

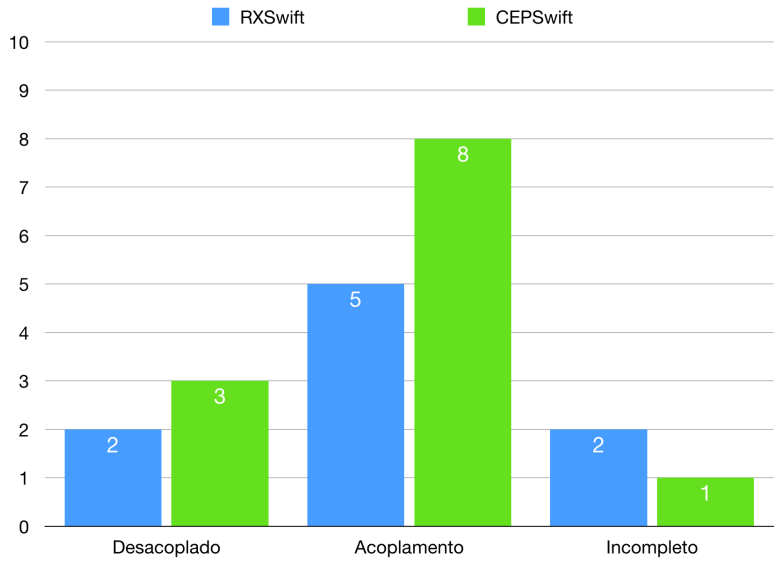

Figura 18 – Corretude das respostas.

6 Ameaças à Validade

Como ameaça à validade deste trabalho, é importante destacar alguns pontos como a escolha do exemplo a ser complementado, nível de conhecimento de programação dos alunos que participaram e influência dos alunos com experiência prévia.

A escolha do exemplo pode afetar a corretude das respostas dos alunos. Com o primeiro projeto escolhido a ser complementado pelos alunos a separação do streaming dos dados era um ponto importante para a resolução, pois facilitava trabalhar com as informações corretas no momentos oportunos. Já com o exemplo mais simplificado que foi utilizado, o acoplamento não causa grandes problemas à resolução e o projeto pode ser completado com um mesmo nível de dificuldade com ou sem a separação dos streamings. Isso pode ter feito com que os alunos fizessem o código usando apenas um streaming, pois não havia muita vantagem, além de organização de código, no uso de mais de um streaming.

Outro ponto a ser levado em conta é que, para alguns alunos do projeto de extensão, esse é o primeiro contato com programação. Alguns deles conhecem apenas conceitos básicos de programação e o assunto abordado usa conceitos mais complexos - mesmo que de forma não direta, foram apresentados conceitos de funções de alta ordem de programação funcional e closures de Swift, que podem ser complexos para programadores menos experientes e sem embasamento em programação funcional.

A interação entre os alunos também pode ter efeitos nos resultados deste trabalho. Alguns alunos já tinham experiência com outras ferramentas de linguagem reativa. Durante a aula e o exercício, esses alunos interagiram com outros, o que pode ter induzido a uma compreensão de RL melhor que aqueles que estavam ao redor e na escolha do framework para completar o projeto.

A falta de tempo para a realização das atividades com os alunos é outro fator que pode ter colocado em risco a validade do trabalho. O limite de tempo não permitiu que os alunos pudessem trabalharam com os dois frameworks e dar uma opinião sólida sobre o contato com as ferramentas, a limitação de tempo permitiu apenas umas avaliação superficial do contato dos alunos com os frameworks.

7 Conclusão

Esse trabalho é o primeiro que compara o aprendizado de duas técnicas, Processamento de Eventos Complexos e Programação Reativa, para o processamento de streaming de dados. A comparação é feita com base em dois frameworks, CEPSwift e RxSwift, que são focados nessas técnicas usando a lingugagem Swift.

Nele foi dada uma introdução do que é um streaming de dados e foram apresentadas duas formas de se trabalhar com streaming, Linguagem Reativa e Eventos Complexos, que se baseiam em RxSwift e CEPSwift, respectivamente. Foi apresentado o fluxo de trabalho com streaming em cinco etapas e como CEP e RL passam por esse fluxo.

Neste trabalho também foi descrita uma aplicação desenvolvida em duas versões, uma usando CEPSwift e outra com RxSwift. Essa aplicação foi usada como exemplo em uma atividade de avaliação com alunos. Aqui também foi descrito o processo avaliativo dividido em três partes e realizado com três grupos de alunos. Um grupo mais experiente em Swift, mas com muito pouca experiência trabalhando com streamings, que serviu como aplicação piloto, e dois grupos de alunos com pouca experiência com Swift, dos quais a grande maioria nunca tinha trabalho com streaming de dados. As três partes da avaliação foram uma aula introdutória com os conceitos de programação reativa CEP, RL e exemplos de código de CEPSwift e RxSwift trabalhando com streamings, um exercício no qual era preciso aplicar os conhecimentos da aula introdutória para realizar o que era pedido e um formulário para coleta de feedback das pessoas que participaram da atividade.

Neste trabalho é descrito ainda como foi a atividade com o grupo preliminar e os feedbacks com esse grupo. É descrito também a realização da atividade com os dois outros grupos maiores de alunos e são mostrados os resultados coletados no formulário preenchido por cada aluno após o término da atividade.

A principal dificuldade encontrada foi que a versão atual de CEPSwift limitou que tipo de aplicações poderiam ser feitas. A ideia inicial da aplicação teve que ser simplificada para que pudesse ser feita tanto em CEPSwift como em RxSwift, já que na versão 0.1.0 de CEPSwift uma série de operadores necessários ainda não foram desenvolvidos.

7.1 Trabalhos Futuros

Esse trabalho cria caminhos para uma série de trabalhos futuros complementares, seja seguindo a ideia do mesmo de comparativo ou usando o que foi aqui mostrado como guia para complementar o desenvolvimento dos frameworks.

No caso de seguir com comparativo, é possível testar outros aspectos de comparação

dos frameworks apresentados ou fazer um trabalho mais aprofundado do que já foi iniciado aqui. Seguindo a linha deste trabalho, é possível fazer uma avaliação semelhante à que foi feita aqui, mas com mais atenção aos pontos de ameaça à validade usando, por exemplo, um exemplo mais completo sobre o tratamento de streamings. Seguindo outra linha comparativa é possível também comparar aspectos como eficiência, organização, robustez, e assim por diante.

Como forma de guia para complementar os frameworks estudados é possível seguir alguns caminhos a partir do que foi mostrado. Pode-se usar esse trabalho como guia para priorizar o que deve ser feito nas próximas versões de CEPSwift, correção de operadores já existentes ou desenvolvimentos de novos operadores. É possível também, pelas respostas dos alunos aqui coletadas, ver o que torna um framework mais fácil de aprender e replicar essa característica para, se possível, criar uma curva de aprendizagem menos íngreme.

Com esse trabalho é possível também ter uma base de que aspectos estudantes têm mais dificuldade de aprender o uso de streamings de dados e assim trabalhar na identificação da melhor abordagem para o ensino do que é e como se pode trabalhar com fluxo de dados.

Referências

ACAR, Y. et al. Comparing the usability of cryptographic APIs. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017. Disponível em: <https://doi.org/10.1109/sp.2017.52>. Citado na página 17.

CEPSWIFT. <https://github.com/RxCEP/CEPSwift>. Accessed: 12/2018. Citado na página 19.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 44, n. 3, p. 15:1–15:62, jun. 2012. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2187671.2187677>. Citado na página 13.

ECKERT, M.; BRY, F. Complex event processing (CEP). *Informatik-Spektrum*, Springer Nature, v. 32, n. 2, p. 163–167, mar 2009. Disponível em: <https://doi.org/10.1007/s00287-009-0329-6>. Citado 2 vezes nas páginas 13 e 14.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2. Citado na página 14.

GOOGLE Forms. 2018. <https://www.google.com/forms/about/>. Accessed: 12/2018. Citado na página 24.

GUEDES, G. B. *CEPSwift: Complex Event Processing Framework for Swift*. 2017. Undergraduate thesis. Citado 3 vezes nas páginas 12, 13 e 19.

JOGO Genius. 2018. <https://pt.wikipedia.org/wiki/Genius_(jogo)>. Accessed: 12/2018. Citado na página 19.

JUPYTER Notebook. <http://jupyter.org>. Accessed: 12/2018. Citado na página 17.

LIKERT scale. 2018. <https://en.wikipedia.org/wiki/Likert_scale>. Accessed: 12/2018. Citado na página 24.

LôBO, R. B. *DEADLOCKS AS RUNTIME EXCEPTIONS*. 2015. M.Sc. Dissertation. Citado na página 17.

MARGARA, A.; SALVANESCHI, G. Ways to react: Comparing reactive languages and complex event processing. *REM*, 2013. Citado 2 vezes nas páginas 14 e 15.

MICHELSON, B. *Event-Driven Architecture Overview*. [S.l.], 2006. Disponível em: <https://doi.org/10.1571/bda2-2-06cc>. Citado 3 vezes nas páginas 6, 11 e 12.

PILLET, F. *RxSwift*: Reactive programming with swift. [S.l.]: raywenderlich.com, 2017. Citado na página 11.

POST-STUDY System Usability Questionnaire. 2018. <https://www.conetrees.com/ux-glossary/post-study-system-usability-questionnaire-pssuq/>. Accessed: 12/2018. Citado na página 24.

REACTIVEX. <http://reactivex.io>. Accessed: 12/2018. Citado na página 19.

RXSWIFT. <https://github.com/ReactiveX/RxSwift>. Accessed: 12/2018. Citado na página 19.

SWIFT. <https://swift.org>. Accessed: 12/2018. Citado na página 19.

THE introduction to Reactive Programming. 2018. <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>. Accessed: 12/2018. Citado na página 14.

WIEDENBECK, S. et al. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, Oxford University Press (OUP), v. 11, n. 3, p. 255–282, jan 1999. Disponível em: <https://doi.org/10.1016/s0953-5438(98)00029-0>. Citado na página 17.