Detecção de Event Races para Aplicações Móveis baseadas em JavaScript

Event Races Detection for JavaScript based Mobile Applications

Henrique Mandelli Canella*, André Takeshi Endo†

RESUMO

O processo de desenvolvimento de aplicações móveis utilizando *frameworks* JavaScript se mostra cada vez mais presente. Sendo assim, tais *frameworks* facilitam o desenvolvimento devido à capacidade de exportação para mais de um sistema operacional utilizando o mesmo projeto. Embora muitas bibliotecas e ferramentas existam para facilitar o desenvolvimento dos projetos, há também uma vasta gama de oportunidades, levando em consideração que são tecnologias recentes. Dentre essas oportunidades, condições de corrida baseadas em eventos que podem ocorrer em aplicações JavaScript. Este trabalho visa a criação de uma ferramenta para auxiliar na detecção de condições de corridas em meio ao ambiente JavaScript, baseando-se em aplicações desenvolvidas utilizando o *framework* React-Native. Com o auxílio da ferramenta Monkey para testes automatizado em sete aplicações de código aberto, e o desenvolvimento de um *script* para leitura de arquivos gerados a partir da execução da aplicação, foram geradas representações em forma de grafos que relatam a possível existência de condições de corrida na aplicação por meio de um arquivo que pode ser acessado no navegador de *internet*.

Palavras-chave: Scripts, Teste automatizado, React-Native, Node.js, Android

ABSTRACT

The process of developing mobile applications using JavaScript frameworks is increasingly present. Thus, such frameworks facilitate the development due to the ability to export to more than one operating system using the same project. Although many libraries and tools exist to facilitate the development of projects, there is also a wide range of opportunities, considering that they are recent technologies. Among these, tools used for identifying and correcting errors related to event race conditions that can occur in JavaScript applications. This work aims to develop a tool that helps detect race conditions in the JavaScript environment, based on applications developed using the React-Native framework. With the aid of the Monkey tool for automated tests in seven open-source applications and the development of a script for reading files generated after the application execution, graph representations were generated reporting the potential presence of race conditions in the application through a file that can be accessed in the browser.

Keywords: Scripts, Automated tests, React-Native, Node.js, Android

1 INTRODUÇÃO

As plataformas de computação móvel são cada vez mais importantes. A porcentagem de pessoas que possuem um *smartphone* alcançou 56% em 2013, e há mais de 1.7 milhões de aplicações móveis disponíveis na Apple Store ou na Google Play (Hsiao et al., 2014). Sendo essas lojas de aplicativos dos sistemas operacionais de dispositivos móveis mais populares atualmente; iOS e Android.

Segundo a IBM Corporation (2012), existem três principais procedimentos para o desenvolvimento de aplicações móveis, sendo esses procedimentos definidos como nativo, web ou híbrido. As aplicações nativas interagem diretamente com o sistema operacional sem intermediário e são capazes de acessar todas as funcionalidades dos dispositivos móveis (MENEGASSE e ENDO, 2020). As aplicações web utilizam as tecnologias HTML, CSS (W3C, 2021) e JavaScript (W3C, 2021), sendo executadas diretamente nos navegadores dos dispositivos. Já as aplicações híbridas contêm a combinação de tecnologias nativas e web, sendo que, com esse procedimento, todas as funcionalidades dos dispositivos móveis podem ser acessadas com o auxílio de uma ponte JavaScript para a comunicação (MENEGASSE e ENDO, 2020).

Portanto, devido à alta crescente de uso de computadores portáteis e suas aplicações disponíveis para facilitar o desenvolvimento de aplicações independentemente do sistema operacional e alocando-se menos recursos, o desenvolvimento multiplataforma pode ser utilizado, uma vez que a maior parte do código serve para os principais sistemas operacionais *mobile*, que são Android e iOS (BRITO et al., 2018).

Com a propagação das aplicações móveis, a exigência por qualidade vem crescendo junto. Seus usuários buscam confiança, robustez e eficiência, portanto os desenvolvedores de *software* devem adotar técnicas que garantam as características adequadas (AMALFITANO et al., 2017). Para atingir o nível de excelência almejado, é desejável que as aplicações móveis passem por fases de testes durante seu processo de desenvolvimento. O teste de software pode ser definido como o processo de execução de um sistema com o objetivo de encontrar defeitos (MYERS; THOMAS e WILEY, 2004). O uso de testes automatizados pode ser útil para que exista um aumento de produtividade, uma vez que eles são realizados com a utilização de *software* para executar os casos de teste o quanto for necessário e os resultados podem ser avaliados com o objetivo de identificar se o teste foi bem-sucedido ou falhou (HOFFMAN, 2001).

É conhecido na literatura que o modelo de execução do JavaScript é suscetível a condições de corrida que impactam a linguagem, não por ela conter múltiplos threads que possam acessar memória compartilhada de forma concorrente, mas sim porque o código é executado de forma assíncrona (ENDO e MØLLER, 2020). Essas condições de corrida existentes em aplicações JavaScript são chamadas de *event races*, condições que ocorrem devido à ordenação não determinística de *event handlers* (*callbacks*) que podem gerar mau funcionamento na aplicação (ADAMSEN; M ØLLER e TIP, 2017), sendo exemplo disso um possível acesso a um servidor de banco de dados, onde o tempo de resposta para o retorno dos dados desejados pode ser o suficiente para causar um erro.

Existem também neste contexto, diversas ferramentas conhecidas para identificar e tratar esse tipo de situação em aplicações JavaScript, podendo ser web ou com o uso em servidores com o *runtime* Node.js, como NodeRacer (ENDO e MØLLER, 2020), AjaxRacer (ADAMSEN et al., 2018), InitRacer (ADAMSEN; MØLLER e TIP; 2017), RClassify (ZHANG e WANG, 2017), dentre outras.

Contudo, em aplicações móveis desenvolvidas com *frameworks* que utilizam JavaScript, é possível identificar o mesmo comportamento suscetível a condições de corrida, sendo que, até o momento não existem estudos a respeito de ferramentas tais como as já apresentadas no contexto abordado neste projeto. Portanto, este trabalho faz uma contribuição a este tema, apresentando o desenvolvimento de uma nova ferramenta para auxiliar a detecção de condições de corrida em aplicações mobile desenvolvidas utilizando o *framework* React-Native.

2 MÉTODO (OU PROCEDIMENTOS OPERACIONAIS DA PESQUISA)

Para o desenvolvimento do código referente à instrumentação das aplicações desenvolvidas utilizando o *framework* React-Native, foi utilizado o Node.js, na versão 14.17.4. O Node.js é uma *runtime* amplamente

utilizada baseada em JavaScript (NODE.JS, 2021), com isso o mesmo foi então injetado nas aplicações gerando resultados de *logs* que foram utilizados para geração de grafos.
Em operações de leitura e gravação de arquivos, foram utilizadas bibliotecas nativas do Node.js referente à manipulação de arquivos, onde os arquivos de *log* gerados foram formatados de melhor forma.

Para manipulação dos dados obtidos a partir da instrumentação do código das aplicações, foram utilizadas a linguagem de marcação HTML e a linguagem de programação JavaScript, sendo as principais bases para construção de *websites* (MDN, 2021). E como meio de facilitar a visualização dos dados obtidos, foi utilizada a biblioteca Cytoscape.js, uma biblioteca *open-source* sob licença MIT para visualização e análise sobre teoria dos grafos (CYTOSCAPE, 2021).

Já na parte de testes para validação dos resultados obtidos, foi utilizado o programa Monkey, que é executado no emulador ou dispositivo Android para gerar fluxos pseudoaleatórios de eventos do usuário, podendo ser: cliques, toques ou gestos, assim como vários eventos do sistema (ANDROID, 2021).

Com isso, a ferramenta foi desenvolvida a partir do estudo de 7 aplicações *mobile* de código aberto, desenvolvidas com o *framework* React-Native, onde foi realizado um estudo a fim de entender o método utilizado para execução delas. A partir das ferramentas selecionadas, foi então realizada a instrumentação da classe Promise, que é utilizada para realização de ações assíncronas no JavaScript, a partir do uso de *proxies* com o objetivo de gerar situações customizadas para a operação (MDN, 2021). A partir desta instrumentação foi possível então receber os dados de ciclo de vida das promises criadas na aplicação React-Native.

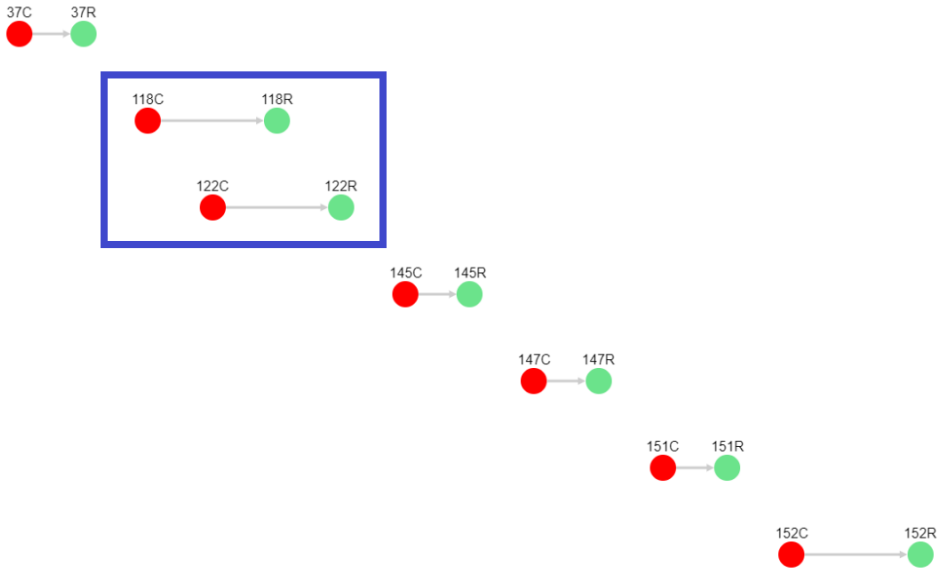
A partir destas instrumentações, foi então desenvolvida uma ferramenta a fim de analisar os dados obtidos das aplicações móveis, onde salvando o arquivo que contém as informações do ciclo de vida das Promises[1], foi possível executar a ferramenta, gerando então uma representação em forma de grafo das possíveis situações em que event races podem acontecer.

3 RESULTADOS

Após a execução dos casos de testes realizados nas aplicações React-Native, foram coletados os resultados e a representação gerada em forma de grafo. Nesta representação, é possível visualizar situações na aplicação analisada em que um possível *event race* possa ocorrer.

[1] O ciclo de vida de uma promise consiste na sua criação, o período em que fica pendente, e sua resolução ou rejeição.

Figura 1 – Representação em forma de grafos gerado pela ferramenta.



Fonte: Autoria Própria (2021).

Pode ser analisado na Figura 1 o exemplo de uma situação gerada na aplicação onde *event races* podem ocorrer. Na figura o marcador em vermelho representa o momento em que uma promise foi criada, e a representação em verde o momento em que ela foi resolvida ou rejeitada. O número presente no topo de cada marcador identifica cada promise e a letra ao lado seu estado, sendo C referente ao estado de criado e R referente à resolvido ou rejeitado. Com a ajuda deste identificador, pode-se comparar seu contexto no código da aplicação. Tais *event races* podem ocorrer quando uma nova promise é criada então, antes da anterior ser resolvida ou rejeitada, podendo ser visualizado no marcador 122C que é criado antes do 118C ser resolvido ou rejeitado, podendo assim gerar uma possível situação suscetível a condição de corrida, devido a possível ocorrência de *interleavings* entre os *callbacks* das promises.

Tabela 1 – Resultados obtidos.

	BENCHMARKS* 100.000 Eventos					
ID	**Aplicativo**	**P-Criadas**	**P-Resolvidas**	**P-Rejeitadas**	**P-Pendentes**	**Suscetível a Event Race**
1	Cosmos	153	146	4	3	Sim
2	Covid19	755	755	0	0	Sim
3	Spotify-lite	10013	10013	0	0	Sim
4	thecatnative	1441	1255	104	82	Sim
5	Finance	265	29	141	95	Nenhuma das promises foram analisadas
6	DuckDuckGo	112	108	0	4	Nenhuma das promises foram analisadas
7	MoviesDaily	402	401	1	0	Sim

Fonte: Autoria própria (2021).

Após a realização dos testes em todas as aplicações, os resultados obtidos podem ser analisados a partir do Quadro 1, gerado a partir da execução da ferramenta Monkey com 100.000 eventos pseudoaleatórios. O quadro apresenta as aplicações com seus nomes, a quantidade de promises: criadas, resolvidas, rejeitadas ou pendentes e a informação se a aplicação é suscetível a *event races* ou não. Para chegar nestes resultados foram limitadas também as promises a serem analisadas, seguindo como meio de análise apenas aquelas que têm alguma relação a utilização de requisições, sendo elas as promises referentes a: fetch e xhrAdapter. Mecanismos estes utilizados normalmente em requisições a servidores.

Portanto com essas limitações, duas das aplicações analisadas, sendo elas a Finance e a DuckDuckGo não possuíam promises que foram criadas neste contexto, sendo assim a análise não se tornando relevante em contexto de *event races.* As outras cinco aplicações que continham promises propensas a serem analisadas, possuem então com base em sua execução situações que poderiam levar a condições de corrida.

4 CONCLUSÃO

Mediante a todo aprendizado obtido até o momento, no que tange a análise e detecção de *event races* em aplicações móveis desenvolvidas com React-Native, os resultados positivos e que cumprem o objetivo deste trabalho a partir da ferramenta desenvolvida, serão utilizados para evolução da ferramenta desenvolvida. Posteriormente, será utilizada na instrumentação das aplicações a injeção de atrasos intencionais nas execuções com foco de analisar o comportamento gerado em tais situações.

AGRADECIMENTOS

Henrique Mandelli Canella é bolsista e autor desta pesquisa de Iniciação Científica e agradece a Universidade Tecnológica Federal do Paraná e todos os órgãos responsáveis pela estrutura, recursos e financiamento para realização deste projeto, André T. Endo é parcialmente financiado pelo CNPq (processo n. 420363/2018-1).

REFERÊNCIAS

ADAMSEN, Christoffer Quis et al. Practical AJAX race detection for JavaScript web applications. **Proceedings Of The 2018 26Th Acm Joint Meeting On European Software Engineering Conference And Symposium On The Foundations Of Software Engineering**, [S.L.], p. 38-48, 26 out. 2018. ACM. http://dx.doi.org/10.1145/3236024.3236038.

ADAMSEN, Christoffer Quist; MØLLER, Anders; TIP, Frank. Practical initialization race detection for JavaScript web applications. **Proceedings Of The Acm On Programming Languages**, [S.L.], v. 1, n. , p. 1-22, 12 out. 2017. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/3133890.

AMALFITANO, Domenico et al. Why does the orientation change mess up my Android application? From GUI failures to code faults. **Software Testing, Verification And Reliability**, [S.L.], v. 28, n. 1, p. 1654-1681, 6 nov. 2017. Wiley. http://dx.doi.org/10.1002/stvr.1654.

BRITO, Hugo et al. JavaScript in mobile applications: react native vs ionic vs nativescript vs native development. 2**018 13Th Iberian Conference On Information Systems And Technologies (Cisti)**, [S.L.], p. 1-6, jun. 2018. IEEE. http://dx.doi.org/10.23919/cisti.2018.8399283.

Cytoscape.js. Disponível em: <https://js.cytoscape.org/>. Acesso em: 15 ago. 2021.

ENDO, Andre Takeshi; MOLLER, Anders. NodeRacer: event race detection for node.js applications. **2020 Ieee 13Th International Conference On Software Testing, Validation And Verification (Icst)**, [S.L.], p. 120-130, out. 2020. IEEE. http://dx.doi.org/10.1109/icst46399.2020.00022.

HOFFMAN, Douglas. Using Oracles in Test Automation. In: NINETEENTH ANNUAL PACIFIC NORTHWEST SOFTWARE QUALITY CONFERENCE, 19., 2001, Oregon. **Proceedings Order Form Pacific Northwest Software Quality Conference.** Oregon: Pnsqc, 2001. v. 1, p. 1-514. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.2495&rep=rep1&type=pdf#page=103. Acesso em: 14 ago. 2021.

HSIAO, Chun-Hung et al. Race detection for event-driven mobile applications. **Proceedings Of The 35Th Acm Sigplan Conference On Programming Language Design And Implementation**, [S.L.], p. 326-336, 9 jun. 2014. ACM. http://dx.doi.org/10.1145/2594291.2594330.

IBM CORPORATION (org.). **Native, web or hybrid mobile-app development**. Disponível em: <http://cdn.computerworld.com.au/whitepaper/371126/native-web-or-hybrid-mobile-app-development/download/?type=other&arg=0&location=featured_list>. Acesso em: 14 ago. 2021.

MENEGASSI, Andre Augusto; ENDO, Andre Takeshi. Automated tests for cross-platform mobile apps in multiple configurations. **Iet Software**, [S.L.], v. 14, n. 1, p. 27-38, fev. 2020. Institution of Engineering and Technology (IET). http://dx.doi.org/10.1049/iet-sen.2018.5445.

Android. **Monkey para testes de IU/aplicativo**. Disponível em: <https://developer.android.com/studio/test/monkey?hl=pt-br>. Acesso em: 15 ago. 2021.

Node.js. Disponível em: <https://nodejs.org/en/>. Acesso em: 15 ago. 2021.

MDN. **Tecnologia Web para desenvolvedores**. Disponível em: https://developer.mozilla.org/en-US/docs/Web. Acesso em: 15 ago. 2021.

W3C. Disponível em: <https://www.w3.org/standards/webdesign/htmlcss>. Acesso em: 12 ago. 2021.

W3C. Disponível em: <https://www.w3.org/standards/webdesign/script>. Acesso em: 12 ago. 2021.

ZHANG, Lu; WANG, Chao. RClassify: classifying race conditions in web applications via deterministic replay. **2017 Ieee/Acm 39Th International Conference On Software Engineering (Icse)**, [S.L.], p. 278-288, maio 2017. IEEE. http://dx.doi.org/10.1109/icse.2017.33.

Myers, G. J., Thomas, T. M. e Wiley, J. (2004) The Art of Software Testing.