Renderização de interface gráfica em sistema embarcado utilizando *Node.js* e o elemento *Canvas*

Andrey K. Nakamura[1], **Álan R. B. Silva**[1], **Lucas Q. Correa**[1], **Marcos Y. Takeda**[1], **Adalbery R. Castro**[1], **Aldebaro Klautau**[1]

[1] Instituto de Tecnologia – Universidade Federal do Pará (UFPA)
Rua Augusto Corrêa, 01 – 66.075-110 – Belém – PA – Brazil

{adalbery, aldebaro}@ufpa.br, {lucas.correa, marcos.takeda}@itec.ufpa.br

Abstract. *Tools and techniques of high-level languages saw an increase in usage on embedded systems due to the rise in popularity and low cost of high performance development boards with embedded Linux distributions, for example Raspberry Pi and BeagleBone. This paper presents the use of HTML5 Canvas element, used on web pages, to create and render a graphical interface on a monocromatic LCD (Liquid Crystal Display) of an embedded system developed using the JavaScript language and the Node.js runtime system.*

Resumo. *Técnicas e ferramentas de linguagens de alto nível vêm sendo cada vez mais aplicadas a sistemas embarcados devido à popularização e baixo custo de placas com sistema Linux embarcado com maiores capacidade de processamento, tal como Raspberry Pi e BeagleBone. Desta forma, o presente artigo relata a utilização do elemento Canvas, normalmente utilizado em páginas Web, para desenho e renderização de interface gráfica em LCD monocramática de um sistema embarcado desenvolvida em linguagem JavaScript e na plataforma Node.js.*

Introdução

O uso de LCD (Liquid Crystal Display) está presente em dispositivos antigos, de telas monocromáticas e de baixa resolução, até monitores e televisões de alta definição presentes no cotidiano da população, sendo o primeiro muito popular entre profissionais de eletrônica e programação para sistemas embarcados.

O interfaceamento com esses LCD gráficos de baixa resolução é realizado através de *drivers* que facilitam o envio de instruções ao dispositivo. Contudo, a implementação destes *drivers* comumente depende de imagens existentes no dispositivo ou de formas geométricas simples, dificultando a criação de interfaces complexas e dinâmicas.

A utilização do elemento *Canvas* proporcinou para a computação gráfica uma evolução no sentido de possibilitar ao desenvolvedor uma abstração para trabalhos com interface gráfica[Levkowitz, H. kelleher, C. 2012]. Além disso, a possibilidade do uso de *Node.js*, um interpretador de *JavaScript*, em sistemas embarcados acrescenta maior abstração em relação a linguagens de alto nível a eles.

Este artigo propõe o uso do elemento *Canvas*, disponível através da tecnologia *HTML5*, assim como a linguagem de programação *JavaScript* e o interpretador *Node.js*

para realizar o processamento de imagens em tempo de execução e gerenciamento do modelo de interfaceamento utilizado no software.

O restante do artigo está organizado da seguinte maneira. A Seção 2 apresenta as tecnologias, ferramentas e a descrição dos modelos utilizados no software. Os resultados estão descritos na Seção 3.

Materiais e métodos

O hardware utilizado na interface é a *BeagleBone Black* [Molloy 2014], uma plataforma de desenvolvimento contendo a distribuição *GNU/Linux Debian 9 Stretch* embarcada, juntamente com o LCD ST7920 [Sitronix Technology]. O ST7920 é um LCD de 128x64 pixels monocromático capaz de renderizar seu display de maneira gráfica. A Figura 1 mostra o hardware em questão.

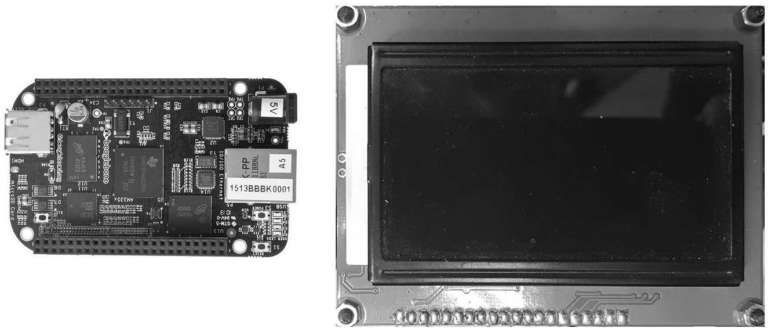

Figura 1. BeagleBone Black utilizada no desenvolvimento do programa exemplo para renderização no LCD à esquerda, e tela LCD ST7920 à direita.

Para o desenvolvimento do programa, foi escolhido o *JavaScript* como linguagem de programação devido à sua natureza de funcionamento direcionada a eventos multiparadigma [Brown 2016], juntamente com o interpretador *Node.js*, disponível em [Node.js], e optou-se por utilizar o elemento *Canvas*, que faz parte do *HMLT5*, uma linguagem para a criação e estruturação de páginas *Web*.

O elemento Canvas possui diversos métodos de controle de texto, posicionamento e figuras geométricas que permitem desenhar imagens em tempo real. Para utilizá-lo, foi necessário o uso da biblioteca *canvas*, disponível através do gerenciador de pacotes do *Node.js npm* (*node package manager*). Este pacote está implementado de acordo com os padrões estabelecidos em [W3C]. Uma das alternativas para comunicar-se com o LCD é a biblioteca *u8glib*, disponível em [olikraus]. Contudo, a biblioteca depende de imagens pré-existentes para utilizar o modo gráfico do *display*, além de não ser capaz de gerá-las dinamicamente.

Para gerenciar a interface, classes contendo um método capaz de renderizar o elemento correspondente a si foram criadas. Elas são responsáveis por implementar componentes específicos, como uma caixa de diálogo pedindo confirmação ou de texto para entrada de dados, um menu ou outra implementação. A Figura 2 mostra o diagrama de classes do módulo responsável pela gerência de telas, renderização e chamada de funcionalidades [Seidl et al. 2015].

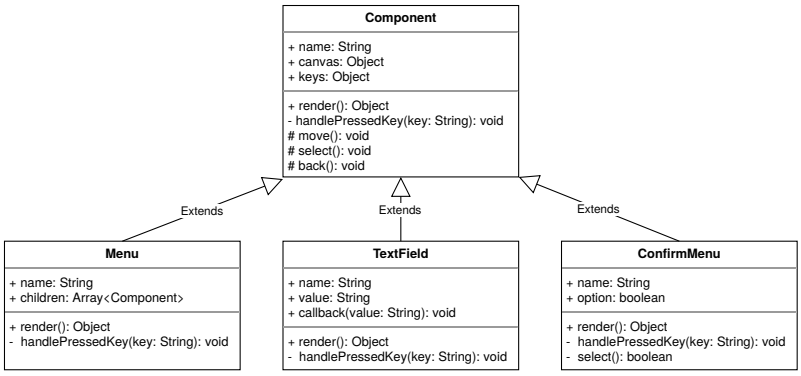

Figura 2. Diagrama das classes utilizadas para escrita no elemento *Canvas*.

Para organizar os componentes na interface LCD, foi desenvolvido um gerenciador de telas cujo funcionamento baseia-se em uma pilha. Ao selecionar um item do menu, o objeto correspondente ao item é movido para o topo da pilha, impedindo o método *handlePressedKey* dos itens que estão abaixo de capturar eventos, redirecionando-os ao objeto no topo. O mesmo é realizado ao voltar ao menu anterior a este objeto. Um exemplo do funcionamento é mostrado na Figura 3.

Os atributos e métodos em comum das classes estão disponíveis na Tabela 1 e Tabela 2, respectivamente.

Tabela 1. Atributos em comum para todas as classes.

Nome do atributo	Descrição
canvas	Contém a interface de sua classe
name	Nome do item em objetos da classe *Menu*
keys	Objeto com as teclas aceitas como entrada para chamada de uma funcionalidade

Tabela 2. Métodos em comum para todas as classes.

Nome do método	Descrição
back	Remove o próprio objeto da pilha
select	Move o item em destaque para o topo da pilha
move	Altera o item em destaque no componente
render	Retorna um objeto contendo o atributo *canvas* e a posição nos eixos x e y no LCD

A classe *Menu* lista os itens na interface de acordo com o atributo *children*, um vetor contendo objetos com o atributo *name* utilizado para a renderização. Para obter esse vetor, o programa deve fornecer os objetos para o método construtor da classe. A classe

TextField fornece uma caixa de diálogo. A classe *ConfirmMenu* apresenta uma caixa de diálogo contendo duas opções, *Yes* e *No*. Ao selecionar a opção *Yes*, o atributo *callback* é executado.

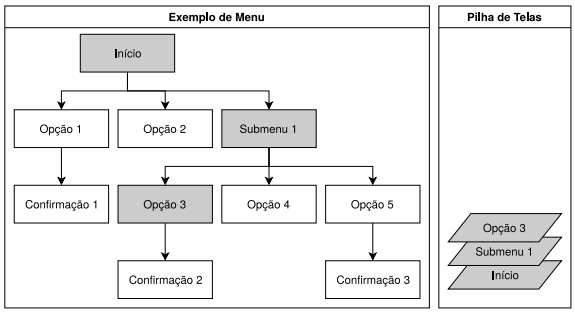

Figura 3. Demonstração do funcionamento da pilha de elementos. Na figura, os itens selecionados estão em cinza e o item de maior profundidade é colocado no topo da pilha para ser renderizado.

Ao receber uma ação do usuário através do método *handlePressedKey*, o componente que recebe-as processa a ação e cria uma nova tela. Após o processamento, é enviado um objeto para um *buffer* em forma de fila contendo a tela resultante e a posição nos eixos x e y dessa tela para ser utilizado no display, disparando um evento. Outra parte do programa captura o evento, processa os dados e envia-os de maneira assíncrona ao LCD. Esse processo é mostrado na Figura 4.

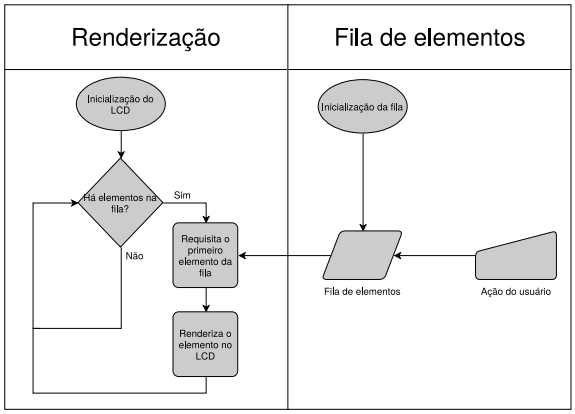

Figura 4. Fluxograma de renderização da interface LCD.

Para realizar a conversão dos dados do Canvas para o display, utiliza-se de um método do próprio elemento para transformar a imagem em um vetor v, onde os índices v_{i*4} são os valores em vermelho, v_{i*4+1} em verde, v_{i*4+2} em azul e v_{i*4+3} do canal alfa, sendo i o índice do pixel do Canvas no vetor. Para acelerar o processamento utilizou-se apenas as cores preto e branco para desenhar, necessitando processar apenas uma das cores presentes no vetor.

Como apenas as cores preto e branco são consideradas no processmento, tendo em vista que a saída deverá ser monocromática, o vetor contendo as intensidades da cor é convertido para um vetor de 0 e 1, que são agrupados para formar bytes a serem enviados ao LCD através do *driver*, escrevendo na tela de acordo com os parâmetros de deslocamento em x e y no LCD.

Resultados

Os resultados obtidos são demonstrados através da implementação de um exemplo de interface para um sistema que contém dois itens, *name* e *position*, ambos os quais podem ser editados, pedindo por confirmação para realizar a edição. A implementação deste sistema demonstra o uso de todas as subclasses mencionadas na Seção 2. A Figura 5 mostra a classe *Menu* renderizada na tela, a Figura 6 a classe *TextField* e a Figura 7 a classe *ConfirmMenu*.

Para avaliar o desempenho, realizou-se a medição do consumo de memória e do uso do processador pelo programa, no número de 10 medições cada. Avaliou-se o consumo de memória utilizando o programa *free*, disponível na distribuição Linux utilizada. Para tal, o consumo de memória foi monitorado em dois momentos, ao iniciar o microcontrolador e ao executar o programa sob estresse. Esse estresse foi obtido ao executar um script para simular a entrada de um usuário 30 vezes por segundo. Com o intuito de se obter uma confiabilidade maior dos dados, o microcontrolador foi reiniciado a cada teste realizado.

A memória RAM disponível é de 512 MB DDR3. A Tabela 3 demonstra os consumos de memória obtidos através da ferramenta. Avaliou-se o uso da CPU (*central processing unit* utilizando o programa *top*, também disponível na distribuição Linux. O uso de processamento foi medido em dois instantes, durante o microcontrolador em *stand-by* e sob estresse. O processador utilizado é o AM335x 1 GHz ARM® Cortex A8 [Texas Instruments] [BeagleBoard.org Foundation]. Os resultados obtidos estão na Tabela 4.

Figura 5. Exemplo da classe *Menu* na interface LCD. A figura mostra dois itens, *name* e *position*, e seus respectivos valores.

As tabelas mostram o uso de memória e do processador ao executar o programa sob estresse. No entanto, parte do consumo dos recursos está relacionado às ferramentas e à linguagem de alto nível utilizadas. Ao avaliar o programa exemplo como um todo, o impacto nos recursos da CPU e no consumo de memória são baixos. O uso majoritário da CPU está relacionado à renderização das telas no LCD, e o uso de memória está relacionado ao objeto utilizando o Canvas para desenhar, controlado pela pilha. Desta maneira, o modelo proposto apresenta escalabilidade, pois o uso de memória é gerenciado através da pilha, e o consumo de CPU não excede o teto alcançado nos testes.

Figura 6. Exemplo da classe *TextField* na interface LCD. A figura mostra a alteração do nome anterior para um novo.

Figura 7. Exemplo da classe *ConfirmMenu* na interface LCD. A figura mostra um menu pedindo a confirmação da alteração do nome anterior para o novo, com a opção *Yes* em destaque.

Tabela 3. Consumo de memória do programa.

Teste Nº	Consumo de memória ao inicializar (kB)	Consumo de memória sob estresse (kB)
1	149.952	277.616
2	149.920	260.960
3	149.944	277.028
4	149.916	275.558
5	149.956	259.658
6	149.940	259.700
7	149.948	276.160
8	149.932	259.624
9	149.908	259.708
10	149.900	259.808

Tabela 4. Uso do processador do microcontrolador.

Teste Nº	Uso em stand-by da CPU (%)	Uso sob estresse da CPU (%)
1	2,6	20,8
2	3,3	21,5
3	2,0	20,7
4	2,6	20,6
5	3,3	19,8
6	3,3	21,9
7	2,6	21,1
8	2,3	20,8
9	2,0	21,2
10	3,3	21,5

Conclusão

A dificuldade de criação de interfaces complexas e dinâmicas utilizando linguagens de baixo nível em tela LCD monocromática expõe a necessidade de estratégias como a apresentada, que aceleram o processo de desenvolvimento de interfaces em sistemas embarcados. Neste trabalho, através do Canvas e do *Node.js* foi possível realizar a criação e o design de interfaces gráficas ituitivas, respondendo em tempo real à entradas do usuário e utilizando pouca memória e pouco uso do processador do microcontrolador.

A performance obtida pela interface provou-se satisfatória, visto que foi possível obter através do *driver* uma taxa de atualização de 30 *fps* (quadros por segundo), ao utilizar *scripts* para simular as entradas de um usuário, tornando a interface dinâmica e responsiva, melhorando a experiência do usuário com o programa, demonstrando-se uma alternativa eficiente e moderna para LCD gráfico monocromático.

Referências

BeagleBoard.org Foundation. BeagleBone Black. `http://beagleboard.org/black`. Acesso em: 2018-10-18.

Brown, E. (2016). *Learning JavaScript: JavaScript Essentials for Modern Application Development*. O'Reilly Media.

Levkowitz, H. kelleher, C. (2012). Cloud and mobile Web-based graphics and visualization.

Molloy, D. (2014). *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley.

Node.js. Node.js. `https://nodejs.org/`. Acesso em: 2018-10-19.

olikraus. u8glib. `https://github.com/olikraus/u8glib`. Acesso em: 2018-10-21.

Seidl, M., Scholz, M., Huemer, C., and Kappel, G. (2015). *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Undergraduate Topics in Computer Science. Springer International Publishing.

Sitronix Technology. ST7920. `https://www.crystalfontz.com/controllers/Sitronix/ST7920/323`. Acesso em: 2018-10-17.

Texas Instruments. Am3358 sitara processor: Arm cortex-a8. `https://www.ti.com/product/am3358`. Acesso em: 2018-10-20.

W3C. HTML Canvas 2D Context. `https://www.w3.org/TR/2dcontext/`. Acesso em: 2018-10-20.