ROKU DEVELOPMENT - WPROWADZENIE

Firma Roku jest działającym głównie na rynku amerykańskim producentem technologii obsługujących streamingi video poprzez internet. Produkty Roku to przede wszystkim urządzenia typu player STB. W wyniku współpracy z firmą Hisense, w ofercie pojawił się niedawno Roku TV – pełnoprawny smart TV. Wsród konkurencji (Google Chromecast, Apple TV, etc.) Roku wyróżnia się:
- największą ilością możliwych do pobrania aplikacji (w Roku zwanych kanałami),
- prostotą i intuicyjnością interfejsu,
- stosunkowo niską ceną;

W efekcie jest to technologia lubiana przez użytkowników. Szacuje się, że w co drugim gospodarstwie domowym w USA występuje jedno urządzenie Roku. W związku z taką popularnością na rynku rodzimym firma zdecydowała się na ekspansję na rynek europejski.

1. TECHNOLOGIA

1.1. Hardware

Przystępna cena urządzenia jest związana z niskim kosztem jego produkcji. Jego komponenty to nienajświeższe nowinki technologiczne. Przeciętne urządzenie Roku jest wyposażone w 1 GB RAM i 4-rdzeniowy procesor z architekturą ARM Cortex A53 (2012 r.). Pomimo tych stosunkowo skromnych parametrów jest ono szybkie i wydajne. 4-rdzeniowa budowa procesora przekłada się na koncepcje architektury platformy Roku jak i determinuje sposób projektowania nań aplikacji.

* więcej: https://developer.roku.com/en-gb/docs/specs/hardware.md

1.2. Software

Urządzenia Roku posiadają dedykowany im system operacyjny – obecnie Roku OS 9.3. Kanały z kolei programowane są w autorskim Roku języku BrightScript wspieranym przez równie autorski framework SceneGraph. Uruchomiona aplikacja działa bezpiecznie w izolacji od pozostałych elementów systemu, przy czym jej skrypt ma dostęp do interfejsów jego usług poprzez wbudowane komponenty BrightScript.

Wspomnianym 4 rdzeniom odpowiada praca 4 wątków całej platformy. 1 rdzeń zarezerwowany jest dla pracy Roku OS, 3 dla działającego kanału. 3 wątki aplikacji:

1. Main BrightScript thread - uruchomia aplikację, podtrzymuje jej działanie (pętla while) i zamyka ją – w reakcji na odpowiedni event (naciśnięcie przycisku Home na RC).

Uruchomienie aplikacji inicjalizuje stworzenie sceny, co zarazem uruchamia wątek 2;

2. SceneGraph render thread - odpowiada za renderowanie wszystkich elementów wizualnych kanału;
3. Task node threads – odpowiada m.in. za tworzenie requestów http, zapisywanie i odczytywanie danych z rejestrów. Wątki tasków mogą działać synchronicznie i asynchronicznie.

* więcej: https://developer.roku.com/en-gb/docs/developer-program/core-concepts/threads.md

2. NARZĘDZIA DEVELOPERSKIE

2.1. Współpracujące z BrightScript'em edytory i przydatne dlań wtyczki

Visual Studio Code:
- BrightScript Language: https://marketplace.visualstudio.com/items?itemName=celsoaf.brightscript
- Roku Development: https://marketplace.visualstudio.com/items?itemName=fuzecc.roku-development
- XML: https://marketplace.visualstudio.com/items?itemName=redhat.vscode-xml

Eclipse:
- Roku plugin: http://devtools.web.roku.com/ide/eclipse/plugin
  https://developer.roku.com/en-gb/docs/developer-program/getting-started/ide-support.md

Sublime:
- Roku plugin: http://devtools.web.roku.com/#sublime-plugin-tool

Atom.io:
- Roku plugin: http://devtools.web.roku.com/#atom-plugin-tool

2.2. Telnet client

* Eclipse z Roku pluginem nie wymaga - ma wbudowaną konsolę z dostępem do portów umożliwiających debugowanie.

2.3. Pozostałe narzędzia – bardziej wyspecjalizowane (niektóre przydatne w zaawansowanych pracach w BrightScript)

- http://devtools.web.roku.com

3. PRZYGOTOWANIE URZĄDZENIA DO PRAC DEVELOPERSKICH

instrukcja: https://blog.roku.com/developer/2016/02/04/developer-setup-guide

* TIP: Przy zakładaniu konta developerskiego pojawia się prośba o podanie numeru karty płatniczej - żadna opłata nie jest pobierana, ale można obejść ten krok rejestrując konto poprzez formularz pod adresem: https://my.roku.com/signup/nocc

4. BRIGHTSCRIPT I SCENEGRAPH

Trudność poznania języka BrightScript związana jest ze skromną ilością źródeł wiedzy oraz jakością publikowanych tam materiałów. Najsensowniejszym źródłem jest dokumentacja Roku, przy czym jej skala oraz nieprzenikniona logika układu treści jest - zwłaszcza na początku - dość ogłupiająca.

* pełna dokumentacja: https://developer.roku.com/en-gb/docs/features/features-overview.md
* forum developerskie: https://community.roku.com/

Poniżej jakoś uporządkowana lista zagadnień.

4.1. Krótko
BrightScript jest językiem interpretowanym, który rzekomo najprędzej podobny jest do Basic'a... Według samych twórców również do Python'a, Ruby i Lua. Obiekty Brightscript i określone struktury danych wejściowych to tablice asocjacyjne. Język nie rozpoznaje małych/dużych liter. Wspiera zarówno dynamiczne jak i statyczne typowanie. Ma wbudowany garbage collecting system. Komponenty aplikacji bazują na gotowych komponentach Roku. Definiowane są w xml'u, co pozwala zmniejszyć ilość kodu proceduralnego.
* więcej: https://developer.roku.com/en-gb/docs/references/brightscript/language/brightscript-language-reference.md

4.2. Tutorial wprowadzający w uroki programowania w BrightScript:
- https://github.com/learnroku/crash-course

4.3. BrightScript extended
4.3.1. Struktura plików aplikacji
Do uruchomienia aplikacji z gatunku „hello world" wystarczy struktura jak w załączniku.
* TIP: Załączony pusty projekt bywa przydatny, kiedy urządzenie się zawiesi - zdarza mu się pokazać czarny ekran w proteście na poważniejszy błąd skryptu. Uruchomienie takiej pustej aplikacji najczęściej je zreanimuje.
* ZAŁĄCZNIK: „empty project.zip"

manifest – plik ten musi istnieć na poziomie root i być niepusty; natomiast w pełnoprawnej aplikacji ma on postać analogiczną do tej z powyższego tutoriala, a jego wymagane pola wymienione są poniżej, w sekcji ad certyfikacja;
main.brs – punkt startowy aplikacji – opisany w punkcie 1.2 ad wątek main, musi się znajdować w folderze source (a ten na poziomie root);
* funkcja main może przyjmować argumenty, np. na potrzeby uruchomienia testów jednostkowych (patrz: sekcja 5)
components – pakiet na poziomie root, zawiera pliki właściwej logiki aplikacji;
resources – w pliku tym mogą znajdować się **images**, **fonts**, **pliki konfiguracyjne**;
locale – pakiet opcjonalnych słowników, pod taką nazwą framework będzie ich szukał;
makefile – opcjonalne skrypty do instalacji kanału

4.3.2. init()
Jeśli komponent ma zdefiniowaną taką funkcję, to będzie ona wywoływana przez framework za każdym razem, kiedy tworzona jest instancja tego komponentu. Typowe jej zastosowanie – inicjalizacja elementów komponentu oraz obserwatorów pól, które uruchamiają inne funkcje w reakcji na zdarzenie (np. zmianę fokusa).
* więcej: https://developer.roku.com/en-gb/docs/references/scenegraph/component-functions/init.md

4.3.3. Node'y
Wszystkie komponenty to w istocie node'y. Każdy z nich rozszerza abstrakcyjną, bazową i nierenderowalną klasę Node, która dostarcza podstawowe funkcjonalności relacji między komponentami oraz zarządzania fokusem. W .brs odpowiednikiem Node jest roSGNode.
* Wszystkie node'y opisane są pod adresem: https://developer.roku.com/en-gb/docs/references/scenegraph
Relacje między node'ami polegają na rozszerzaniu. Rodzina node'ow ma pewnego rodzaju hierachię:
* ZAŁĄCZNIK: „SG inheritance scheme.docx"
Można swobodnie tworzyć własne komponenty, ale muszą one rozszerzać któryś z predefiniowanych.
Dla wygody uporządkowane „drzewo" node'ów w wersji z polami:
* ZAŁĄCZNIK: „SG inheritance scheme with fields.docx"

* TIP: Na początku może się wydawać, że niektóre predefiniowane komponenty są bardzo wygodnym ułatwieniem pracy. Jednak próby ich dostowywania do projektów graficznych aplikacji bywają prawdziwie upiorne, a czasem niemożliwe. Przykładem jest Button... - łatwiej złożyć i edytować własny z prostszych komponentów jak np. Poster i Label.

4.3.4. Struktura pliku .xml
Plik .xml stanowi definicję komponentu. Format ten jest silnie typowany i rozpoznaje małe/duże litery.
Elementy:
<component> - wymagany, definiuje nazwę node'a – poprzez atrybut **„name"** oraz wskazuje rozszerzany komponent – **„extends**" (domyślnie jest to Group), w sekcji tej można też określić na jakim elemencie ma zostać ustawiony początkowy fokus – **„initialFocus";**
<interface> - definiuje interfejs pól (za pomocą tagu „**field**") lub funkcji (**„function"**) komponentu, które będą widoczne dla komponentów w całej aplikacji;
<function> - identyfikowana jest poprzez atrybut **„name"**;
<field> - można definiować pola własne dla całego komponentu, jak i wyeksponować pole któregoś z node'ów dzieci (za pomocą **„alias")**. Obowiązkowe są atrybuty **„id"** (nazwa) oraz **„type"**. Poprzez **„value"** można ustawić początkową wartość

pola, **„onChange”** – wskazuje funkcje obserwatora pola, natomiast **„alwaysNotify”** – dla wartości true powoduje reakcję na każdą aktywność dotyczącą pola, dla false - tylko kiedy jego wartość ulega zmianie;

<script> - wskazuje skrypt podpięty do komponentu (może być ich wiele, ale tylko jeden może mieć funkcje init()), definiują go atrybuty **„type”** (w Roku jest to zawsze wartosć: "text/brightscript") oraz **„uri”** – wskazuje na ścieżkę pliku.

* TIP: można zagnieżdżać kod BrightScript'owy w xml'u (widać to w przykładach od Roku – patrz punkt 4.3.12), niemniej traci się wtedy czytelność oraz i tak skromne już kolorowanie składni przez wcześniej opisane wtyczki;

* TIP: jeśli pliki .brs znajdują się w tym samym pakiecie co odpowiadające im pliki xml, to nie trzeba podawać pełnej do nich ścieżki – wystarczy nazwa, co np. znacznie ułatwia refaktor kodu;

<children> - komponenty "dzieci", ich hierachia w dokumencie odpowiada kolejności ich renderowania.

* więcej: https://developer.roku.com/en-gb/docs/references/scenegraph/xml-elements/component.md

Renderowanie całych komponentów odbywa się w kolejności:
<children>
<interface>
<script>
Wynika z tego, że node rodzic może zmienić stan komponentu dziecka.
Elementy, które są niewidoczne nie są renderowane.

4.3.5. BrightScript podstawy
Warto przejrzeć całą sekcję Language:
https://developer.roku.com/en-gb/docs/references/brightscript/language/brightscript-language-reference.md
Data scope, czyli magic m., jest omówione jakoś całkiem gdzie indziej:
https://developer.roku.com/en-gb/docs/developer-program/core-concepts/data-scoping.md

4.3.6. Events/Messages
Model interakcji jest zorientowany zdarzeniowo i umożliwia asynchroniczną komunikację między obiektami. Elementy UI odpowiadają na fokus lub wybór dokonany przez użytkownika via RC. W BrightScript ta technika to „field observer". Obserwowane pole może być wyeksponowane w interfejsie pliku .xml lub w kodzie .brs.

* więcej: https://developer.roku.com/en-gb/docs/developer-program/core-concepts/handling-application-events.md
https://developer.roku.com/en-gb/docs/developer-program/core-concepts/event-loops.md

4.3.7. onKeyEvent
Funkcja ta przechwytuje z Roku OS informacje o naciśnięciu przycisku na pilocie i umożliwia obsłużenie reakcji na to zdarzenie. Przyjmuje 2 parametry: key i press. Key określa, który przycisk został naciśnięty – wartość jest typu string. Press – ma wartość true po naciśnięciu i false, kiedy przycisk został zwolniony. Funkcja zwraca true, kiedy zdarzenie zostało obsłużone, false, kiedy nie zostało, wtedy oddaje sterowanie do komponentu parenta.
Niektóre predefiniowane komponenty obsługują te zdarzenia automatycznie (np. RowList, czy PosterGrid) i nie powinno im się funkcji onKeyEvent definiować. Do śledzenia eventów tych komponentów można używać obserwatorów pól.
Z kolei zachowania niektórych przycisków pilota nie można programować, są obsługiwane bezpośrednio przez Roku OS – jak np. „Home".

* więcej: https://developer.roku.com/en-gb/docs/references/scenegraph/component-functions/onkeyevent.md

4.3.8. ContentNode
- https://developer.roku.com/en-gb/docs/references/scenegraph/control-nodes/contentnode.md

4.3.9. Task
- https://developer.roku.com/en-gb/docs/references/scenegraph/control-nodes/task.md

4.3.10. Pamięć i przechowywanie danych
- https://developer.roku.com/en-gb/docs/developer-program/getting-started/architecture/file-system.md

4.3.11. Certyfikacja i inne ważne
Wszystkie komercyjne kanały podlegają certyfikacji i muszą spełniać określone zasady. Na początek warto wiedzieć:
- wielkość całej aplikacji to maksymalnie 4 MB,
- bezpieczne kolory: wszystkie jego składowe mogą mieć maksymalną wartość EB (dla zapisu heksadecymalnego) i 235 dla RGB, czyli np. najbielsza biel jakiej można użyć to: EBEBEB = RGB(235, 235, 235),
- w pliku manifest obowiązkowo muszą znajdować się informacje: nazwa aplikacji, jej wersja, ścieżka do ikony kanału oraz ścieżka do splash screena – „powitalnej grafiki", natomiast w folderze np. images – odpowiadajace im pliki graficzne.

* Kompletna lista wymagań: https://developer.roku.com/en-gb/docs/developer-program/certification/certification.md

Poza certyfikacją warto mieć na uwadze:
- safe zone'y związane z rozdzielczością różnych ekranów, co implikuje ograniczenia UI/UX:
https://developer.roku.com/en-gb/docs/specs/graphics.md

- Roku automatycznie dostosowuje się do rozdzielczości ekranu (choć assety graficzne muszą być przygotowane dla każdej rozdzielczości osobno). Stąd prace nad aplikacją wykonywane są pod kątem jednej – wyższej rozdzielczości i z małymi wyjątkami (np. maski) nie trzeba sie o tę drugą martwić. Na ten moment najczęściej obsługiwane to FHD i HD. Ich wymiary są w stosunku do siebie jak 3:2. W związku z tym - dla optymalnej jakości grafiki w rozdzielczości niższej - dobrze jest pamiętać o tym, by wymiary wszystkich elementów graficznych (jak również odległości między nimi) miały wartości podzielne przez 3.

4.3.12. Przykłady kodu od Roku
https://github.com/rokudev/samples#roku-sample-channels
* Tu warto mieć na uwadze, że nie są to jakieś wzorcowe czy szczególnie eleganckie rozwiązania, ale dają pogląd na pracę z BrightScript'em i jego komponentami.

5. ROOIBOS – FRAMEWORK DO TESTÓW JEDNOSTKOWYCH
- wstęp:
https://wiki.intive.com/confluence/display/ROK/Rooibos+test+framework
- dokumentacja:
https://github.com/georgejecook/rooibos/blob/master/docs/index.md
https://georgejecook.github.io/rooibos/index.html
- 4-częściowy cykl kursów video autorstwa George'a Cooka:
https://www.youtube.com/watch?v=0kJzYfdGhL4&list=PLJRLV4QDx83vsYMD9bIs-cjoDXmNmO8Jv&index=1