

**Instituto Superior de Engenharia de Lisboa**

**Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores**



**BrightScript Debugger**

Celso de Almeida Fernandes

(Licenciado Engenharia Informática e de Computadores)

**Resumo**

Orientadores:

Eng. Paulo Pereira

Eng. Pedro Pereira

Júri:

Presidente: Eng. Manuel Barata

# Introdução

O objetivo deste projeto é implementar uma ferramenta de desenvolvimento para BrightScript, a ferramenta suporta validação sintática, intellisense, debug com interação gráfica.

O BrightScript é uma linguagem baseada no JavaScript e no Visual Basic criada pela Roku. A Roku é uma empresa que produz boxs para ver filmes e televisão.

A solução baseia-se numa extensão para o Visual Studio, a extensão disponibiliza a criação de novos projetos, edição/compilação de código, publicação e debug.

O projeto está dividido em três partes, estudo da teoria de compiladores, analise de ferramentas de geração de compiladores e a implementação.

# Teoria de compiladores

Um compilador é um programa que processa código fonte escrito numa determinada linguagem e gera código que um comutador consegue correr. Um compilador é muito complexo, para simplificar é modularizado. Como mostra a figura.



Figure 1 – modulos dum compilador

## Analisador Léxico

O analisador léxico recebe uma stream de caracteres e gera uma stream de tokens, descarta os espaços em branco e os comentários entre os tokens. Um token é uma sequencia de caracteres que é a unidade base duma linguagem de programação.

Para especificar os tokens usa-se expressões regulares, que representam um conjunto de strings. Para determinar os tokens utilizasse o algoritmo finite autómata, que encontra o maior token.



Figure 2 - ﬁnite automaton

## Parser

O parser faz analise sintática, analisa a ordem das frases. Na analise sintática usa-se os algoritmo contexto-free grammars que descreve uma linguagem como um conjunto de produção do tipo

symbol → symbol symbol ···symbol

O resultado da analise sintática é Abstract Syntax Tree (AST).

## Abstract Syntax Tree

Abstract Syntax trees é uma estrutura utilizada pelos compiladores que representam uma arvore de código.



Figure 3 - Abstract Syntax Tree

# Geração de código

A implementação do Lexer e do Parser é uma atividade repetitiva que pode ser automatizada, para essa automatização utiliza-se duas ferramentas que geram código em C#, que pode ser utilizado nos plugins do Visual Studio. As ferramentas são o GPlex que gera um analisador léxico e o Gppg que gera um Parser. O GPlex gera um analisador léxico com base num conjunto expressões regulares, que utiliza o algoritmo finite automatá. O Gppg gera um Parser com base num conjunto de contexto-free grammers.

# Debugger

Para testar as funcionalidades de debug foi criada uma aplicação, que permite fazer deploy e debug das aplicações. A aplicação tem como componentes Telnet, Deploy e Remote como mostra o diagrama seguinte.

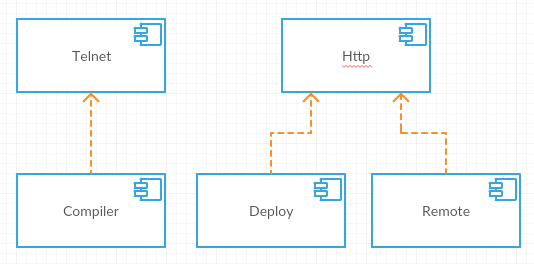


Figure 4 - Debuger diagram

O Telnet utiliza um compilador para gerar o contexto do debugger. O Deploy gera o package e faz upload para a box. O Remote emula um comando.

As funcionalidades do debugger foram integradas no plugin do Visual Studio.



Figure 5 - Debugger Application

## Telnet module

O modulo Telnet implementa duas funcionalidades: processar o output da box e enviar comandos de debug. Como resultado do processamento do output conseguimos obter as variáveis correntes e o call stack da execução.

O diagrama seguinte mostra os subcomponentes.



Figure 6 - Telnet component diagram

O Socket faz a ligação ao porto telnet para receber o ouput da box. O Compiler processa o output e gera o call stack e as variáveis correntes. As componentes Variables UI, Call Stack UI e Output Visualizer mostram os dados ao utilizador. O Input envia os comandos para a box. O UI Commands disponibiliza os comandos numa toolbar.



Figure 7 - Compiler output windows



Figure 8 - Output visualizer



Figure 9 - UI Commands

## Deploy

O processo de Deploy consiste em gerar um zip e fazer upload para a box.

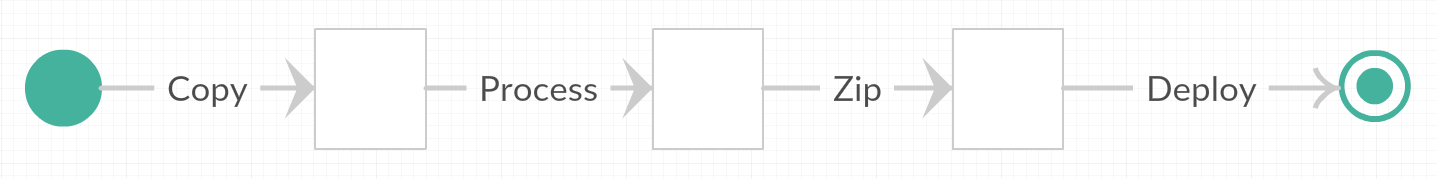


Figure 10 - Deploy process

Para fazer deploy da aplicação copiamos os ficheiros para uma pasta, processamos os ficheiros, geramos o zip e fazemos upload para a box.

O processamento serve para passar configurações para a box, através de substituição de código.

O Deploy utiliza o protocolo Http fazer upload do zip para o site disponibilizado pela box.

## Remote

O componente Remote utiliza os Roku External Control Services em Http para emular um comando da box, possibilitando a interação com a box utilizando só o PC de desenvolvimento.



Figure 11 – Remote

# Visual Studio Plugin

A implementação está dividia em quatro componentes.

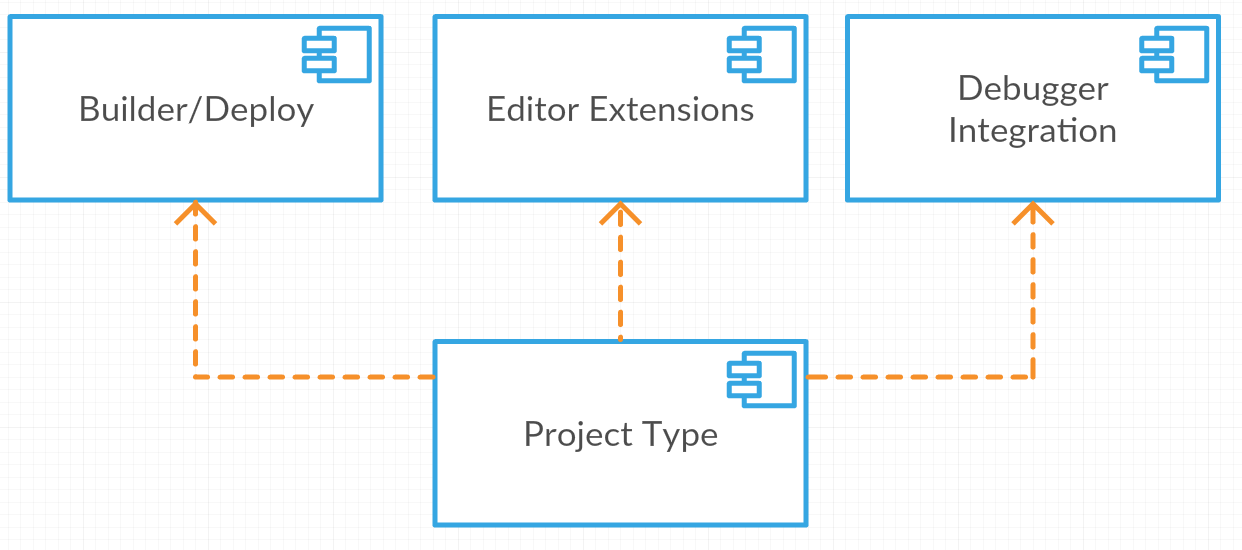


Figure 12 - Visual Studio Plugin diagram

O Builder/Deploy é responsável por compilar e publicar a aplicação. O Editor Exetensions é responsável pelo sintax highlighting. O debugger integration é responsável pela integração do debugger. O Project type é responsável pelos templates de projeto.

## Project Type

O project utiliza o VSProjectSystem para disponibilizar os templates de projeto de items. O template do projeto define a base do ficheiro de projecto, que vai ter todos os ficheiros, as configurações e ordem de execução das tasks de deploy. Os templates dos items definem a base dos ficheiros de codigo.

## Builder/Deploy

O componente Bulder/Deploy define as tasks de deploy. Este componente utiliza o compilador e o código desenvolvido no debugger.

## Editor Extension

O editor extension estende a funcionalidade do editor do Visual Studio disponibilizando sintax highlighting, erros e Intellisense.



Figure 13 - Syntax highlighting



Figure 14 - Editor error



Figure 15 - Error window



Figure 16 – intellisense

## Debugger integration

O debugger é baseado na implementação Visual Studio MI Debug Engine e usa o código desenvolvido na aplicação de debugger. A implementação usa as seguintes componentes que implementam as interfaces disponibilizadas pelo SDK do Visual Studio para comunicar com o Visual Studio: “AD7Engine”, “AD7Thread”, “AD7StackFrame”, “AD7DocumentContext”, “AD7Events”.

O core do debugger está implementado nas classes apresentadas no seguinte diagrama.

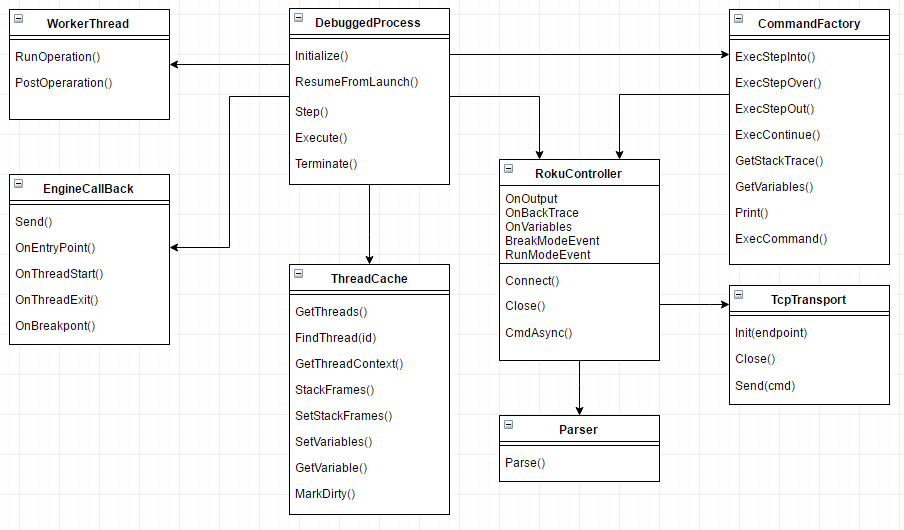


Figure 17 - Debug Engine Class Diagram

O DebuggedProcess gere todo estado recebe os comandos do AD7Engine, os eventos do RokuController, usa o EngineCallback para enviar os eventos para o Visual Studio.

EngineCallback envia os eventos para o Visual Studio.

O RokuController gere a conecção com a box, utilizando o TcpTransport, o Parser para processar o output e gere o envio de comandos e as respostas.

O CommandFactory envia os comandos específicos para a o RokuController.

A WorkerThread gere o background work.

O ThreadCache faz cache do processamento das variáveis e do call stack.

O VariableInformation representa uma variável na box.

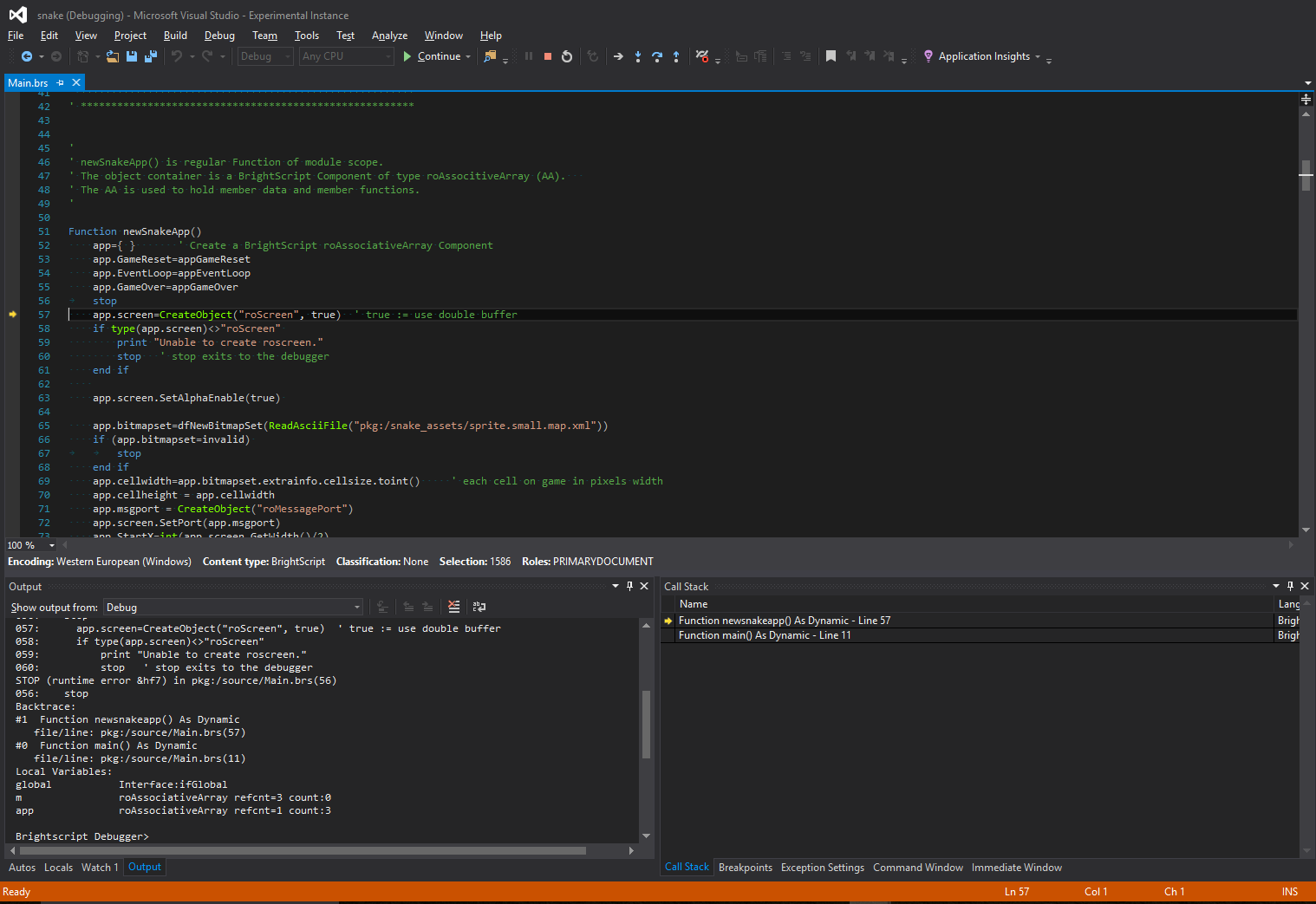


Figure 18 - Debugging BrightScript

O DebuggedProcess envia todo o output para o Visual Studio.

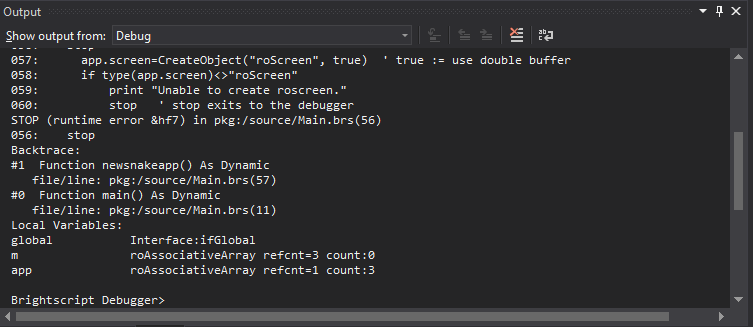


Figure 19 - Output Window

Quando a box entra em break mode o DebuggedProcess envia o call stack um evento AD7BreakpointEvent para o Visual Studio com o stack frame.

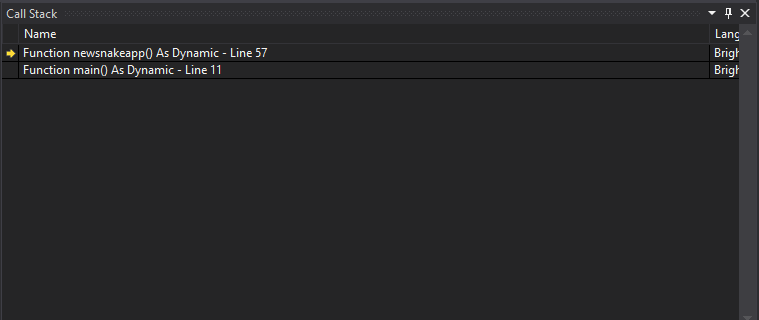


Figure 20 - Call Stack Window

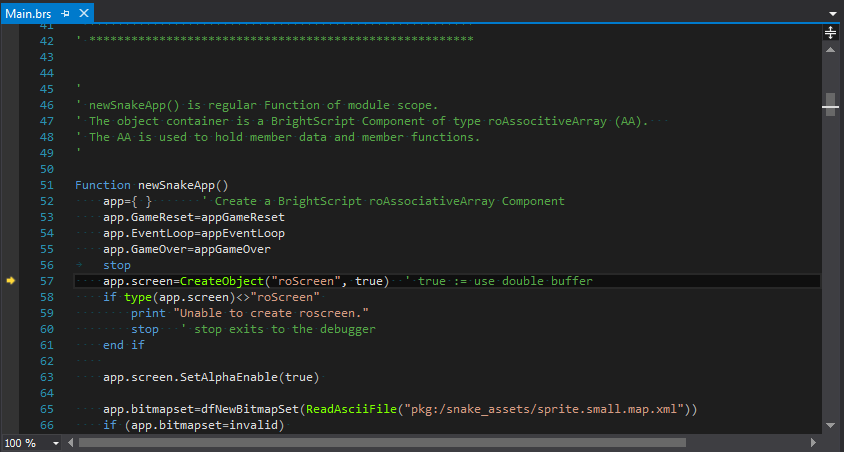


Figure 21 - Editor window showing break point

O call stack também tem a lista de variáveis locais que podem ser visualizadas no janela de Locals.

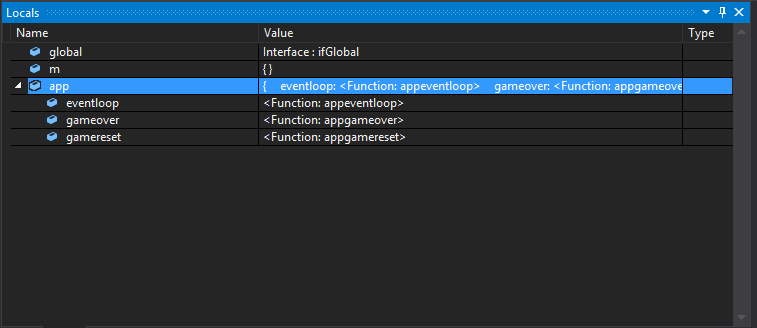


Figure 22 - Locals Window

Ou na janela de Watch.

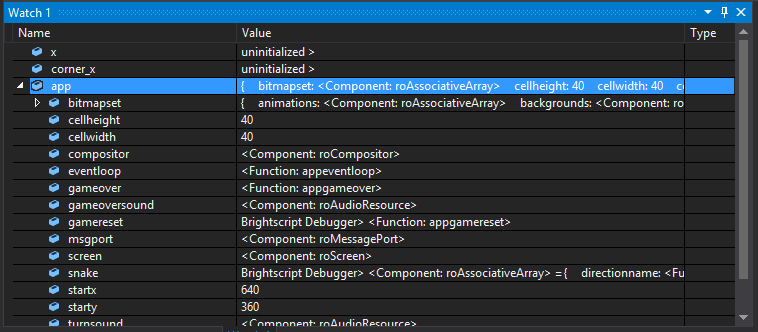


Figure 23- Watch Window

## Tool Windows

Foram criadas duas janelas para integrar o desenvolvimento no Visual Studio. Um é o Remote desenvolvido na Debugger e outra é um serviço que faz screen shots da box, permitindo visualizar o ecrã da box.

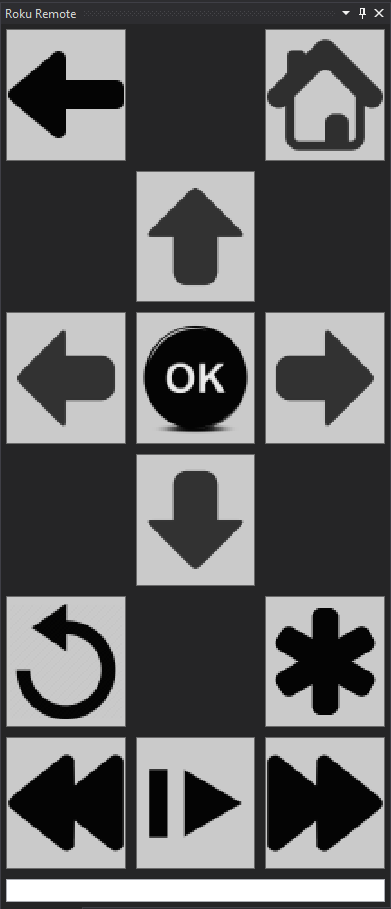


Figure 24 - Remote Window

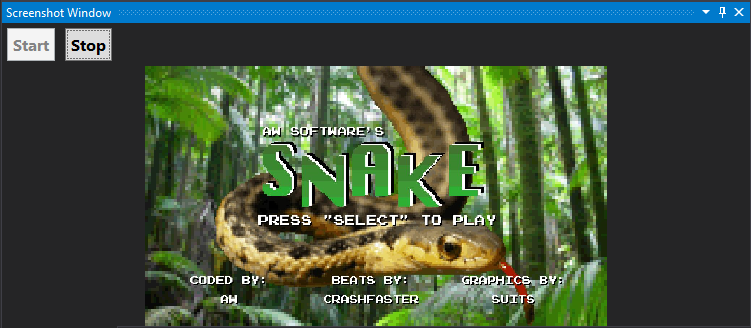


Figure 25 - Screenshot Window

# Conclusão

A extensão do Visual Studio suporta todas a features propostas, edição de codigo, syntax highlighting, validação de erros, intellisense, build e deploy da aplicação e debug integrado.

A extensão pode ser utilizada em projetos simples, mas precisa de alguns melhoramentos para utilizar em projetos mais complexos.

O Parser não suporta todas as Statments. Tais como if eles encadeados e o uso de palavras reservadas como identificadores.

O Debug Engine precisa de mais performance e de permitir a visualização de variáveis no stack.

O intellisense só usa o input do ficheiro corrente. O compilador precisa de gerar uma base de dados com todas as ASTs, para servir de input para o Intellisense.

Como trabalho futuro identificamos as seguintes tarefas: uma ferramenta para converter a base de código existente e adicionar suporte para a nova framework Scene Graph.