# 1
# Introdução

O Common Language Runtime (CLR) é uma plataforma criada com o objetivo de facilitar a interoperabilidade entre diferentes linguagens de programação, através de uma linguagem intermediária (a Common Intermediate Language, ou CIL) e um sistema de tipos comum (o Common Type System, ou CTS) [12, 13]. A especificação do CLR é um padrão da International Standards Organization (ISO) e da Ecma International [14], e implementações existem para os sistemas operacionais Microsoft Windows, Linux, FreeBSD e MacOS X [15, 17]. Diversas linguagens têm compiladores para o CLR já prontos ou em desenvolvimento; uma relação destas linguagens pode ser encontrada na Internet [18].

A Common Language Specification (CLS) é um subconjunto da especificação do CLR. A CLS define as regras que os compiladores devem seguir para garantir a interoperabilidade entre as linguagens [14, CLI Partição I Seção 7.2.2]. Os compiladores que geram código capaz de usar bibliotecas que seguem as regras da CLS são chamados *consumidores CLS*. Os que podem produzir novas bibliotecas, ou estender bibliotecas existentes, são chamados *extensores CLS*.

O código gerado por um consumidor da CLS deve poder chamar qualquer método ou *delegate* que siga as regras da CLS, mesmo métodos com nomes inválidos na linguagem; chamar métodos com a mesma assinatura mas de interfaces diferentes; instanciar qualquer tipo que siga as regras da CLS, inclusive tipos aninhados; ler e atribuir a qualquer propriedade que siga as regras do CLS; registrar e remover tratadores para qualquer evento que siga as regras do CLS.

Linguagens de script são frequentemente usadas para conectar componentes escritos em outras linguagens. Também são usadas na construção de protótipos, e em arquivos de configuração. A natureza dinâmica das linguagens de script permite o uso de componentes sem a declaração prévia de tipos, e sem a necessidade de compilação. Ainda assim, elas checam a corretude de todas as operações em tempo de execução, fornecendo informações

PUC-Rio - Certificação Digital Nº 0210489/CA

detalhadas em caso de erros. Os recursos das linguagens de script podem aumentar a produtividade dos programadores em um fator de dois ou mais [3].

Lua [19, 1, 2] é uma linguagem de script flexível e com um interpretador pequeno, eficiente e fácil de embutir em outros programas. Lua tem uma sintaxe simples, é interpretada, dinamicamente tipada, e possui recursos reflexivos. Lua também possui funções como valores de primeira classe, escopo léxico, e co-rotinas.

Esta dissertação apresenta a integração entre a linguagem Lua e o CLR. O objetivo da integração foi oferecer todos os recursos de um consumidor completo da CLS, ou seja, permitir que scripts Lua instanciem e usem objetos do CLR, seguindo a sintaxe de Lua. Uma interface entre Lua e o CLR dá acesso a componentes escritos em qualquer linguagem que possa definir tipos para o CLR para os scripts Lua.

São usadas duas abordagens para obter a integração desejada. A primeira abordagem consistiu na criação de uma ponte entre o interpretador Lua e o CLR. O objetivo foi usar os recursos reflexivos da linguagem Lua e do CLR para construir esta ponte sem precisar fazer qualquer mudança no interpretador Lua. Sua operação não exige que os scripts sejam pré-processados ou que sejam criados *stubs* para os tipos do CLR que os scripts usam.

A segunda abordagem consistiu na compilação dos bytecodes da máquina virtual Lua para instruções da CIL. O objetivo foi construir um compilador de Lua para o CLR que oferecesse todos os recursos da linguagem, sem introduzir modificações na sua sintaxe ou no seu comportamento. A preocupação com o desempenho também esteve presente, em ambas as abordagens; seus desempenhos foram avaliados e comparados com os desempenhos de trabalhos similares.

A razão de se implementar a primeira abordagem é verificar se a integração entre Lua e o CLR é possível sem precisar de um compilador Lua, e definir como será a interface entre Lua e o CLR. O sucesso dessa abordagem mostra um caminho que outras linguagens de script podem usar para uma integração fácil com o CLR. Além da facilidade de implementação, em relação a um compilador, outra vantagem de uma ponte é a possibilidade de usar as bibliotecas já existentes para a linguagem Lua, em geral escritas em C. Isso também vale para outras linguagens de script, que têm bibliotecas escritas em C para melhor desempenho.

Implementar a segunda abordagem, por sua vez, permite verificar qual o grau de dificuldade de se implementar linguagens dinâmicas sobre o CLR, e quais empecilhos existem. A construção de compiladores de linguagens

PUC-Rio - Certificação Digital Nº 0210489/CA

de script para o CLR tem sido problemática. Os projetos de criação de compiladores das linguagens Perl e Python foram abandonados antes de serem concluídos [7, 8]. A companhia Smallscript Inc. vem trabalhando em um compilador da linguagem Smalltalk desde 1999, e nenhuma versão está publicamente disponível [9]. Um fator comum entre esses projetos é a sua ênfase em estender o CLR com a criação de novas classes *durante a compilação*. Essa ênfase dificulta a construção dos compiladores, pois criação e extensão dinâmica de tipos são recursos comuns das linguagens de script. A segunda abordagem deste trabalho pretende contornar as dificuldades encontradas na construção dos outros compiladores, enfatizando primeiro a implementação dos recursos da linguagem e segundo a utilização de tipos do CLR já existentes, abandonando a criação de novas classes durante a compilação.

O restante deste capítulo apresenta mais detalhadamente a linguagem Lua e o Common Language Runtime. O Capítulo 2 apresenta a primeira abordagem de integração e sua implementação, a biblioteca LuaInterface. O Capítulo 3 apresenta a segunda abordagem de integração e o compilador Lua2IL. O Capítulo 4 descreve outros trabalhos similares e uma avaliação do desempenho de ambas as abordagens. Finalmente, o Capítulo 5 resume as conclusões deste trabalho e sugestões para melhorias e trabalhos futuros.

PUC-Rio - Certificação Digital Nº 0210489/CA

## 1.1 A Linguagem Lua

Lua é uma linguagem de script projetada para ser facilmente embutida em aplicações, e usada como linguagem de configuração e extensão. Lua é usada atualmente em projetos no mundo todo, em especial jogos de computador; uma lista de alguns projetos que usam Lua está na Internet [24]. Lua tem uma sintaxe simples, e combina características de linguagens procedurais (estruturas de repetição e testes, atribuição, definição de funções com parâmetros e variáveis locais) com recursos mais avançados como arrays associativos, funções como valores de primeira classe, escopo léxico e co-rotinas.

O interpretador de Lua é implementado em ANSI C, como uma biblioteca. Os scripts Lua são primeiro compilados para uma representação intermediária (*bytecodes*) e em seguida executada por uma máquina virtual. A Seção 3.1 descreve a máquina virtual de Lua em mais detalhes. A biblioteca possui uma API que oferece funções para iniciar e controlar instâncias do interpretador, descrita mais detalhadamente na Seção 2.2.1.

Lua é uma linguagem dinamicamente tipada, e possui sete tipos: `nil`, *number*, *string*, *boolean*, *table*, *function* e *userdata*. O tipo `nil` tem apenas um valor, `nil`, e representa uma referência não inicializada (ou inválida); valores do tipo *number* são números de ponto flutuante com precisão dupla (o tipo `double` da linguagem C); valores do tipo *string* são cadeias de caracteres (o tipo `char` da linguagem C); o tipo *boolean* tem dois valores, `true` e `false`; valores do tipo *table* são tabelas, ou arrays associativos; funções pertencem ao tipo *function*; e o tipo *userdata* é para dados arbitrários da aplicação que está embutindo o interpretador.

As tabelas podem ser indexadas por qualquer valor, de qualquer tipo; o interpretador otimiza o acesso a índices inteiros, usando um vetor, e uma tabela hash armazena os valores para outros tipos de índices. Lua também oferece "açúcar sintático" para usar tabelas como registros (ou objetos): a expressão `tab.campo` é equivalente a `tab["campo"]`.

Funções são valores de primeira classe, e portanto podem ser passadas como argumentos para outras funções, atribuídas a variáveis, guardadas em tabelas e retornadas por funções. Lua também permite definir funções anônimas. Tabelas e funções de primeira classe permitem a criação de objetos em Lua; as funções armazenadas em uma tabela são os seus métodos. Lua também oferece açúcar sintático para definir métodos. Se `tab` é uma tabela, a declaração

```
function tab:metodo(param1,param2)
  -- código do método
end
```

é equivalente a

```
tab["metodo"]=function (self,param1,param2)
  -- código do método
end
```

e uma expressão como `tab:metodo(arg1,arg2)` é equivalente a `tab["metodo"](tab,arg1,arg2)`, ou seja, o objeto de destino da chamada vira o primeiro argumento da função.

Funções podem ser aninhadas arbitrariamente (o próprio corpo do script é uma função anônima), e as funções internas têm livre acesso às variáveis locais das funções externas (escopo léxico). Por exemplo, o trecho de código

```
function faz_contador()
  local cont=0
```

PUC-Rio - Certificação Digital Nº 0210489/CA

```
    return function ()
             cont=cont+1
             return cont
           end
  end
```

define uma função `faz_contador` que, a cada vez que é chamada, retorna uma função que incrementa e retorna o valor da variável `cont`. Como essa variável é local a uma função externa (a função `faz_contador`) o valor é "lembrado" entre diferentes invocações: a primeira chamada à função retornada por `faz_contador` retorna 1; a segunda chamada à mesma função retorna 2, e assim por diante.

A linguagem Lua também permite a extensão e modificação do comportamento de tabelas e userdata, em operações como indexação e comparação. As Seções 2.2.1 e 2.2.5 descrevem este recurso e como ele pode ser utilizado para estender o comportamento do interpretador.

Finalmente, as co-rotinas são linhas independentes de execução dentro de um script Lua, mas sem um escalonador; uma co-rotina tem que suspender explicitamente sua execução chamando uma função. Apenas uma co-rotina é executada por vez, mesmo que a máquina seja capaz de execução paralela. A Seção 3.4.1 mostra como as co-rotinas são criadas e usadas.

PUC-Rio - Certificação Digital Nº 0210489/CA

## 1.2 O Common Language Runtime

Como já dito no início deste capítulo, o CLR é uma plataforma criada para facilitar a interoperabilidade entre diferentes linguagens de programação. Ele oferece uma linguagem intermediária e um ambiente de execução comuns, com recursos como um *heap* gerenciado por um coletor de lixo, suporte a criação e sincronização de linhas de execução concorrentes (*threads*), segurança e autenticação de código. Além disso o CLR oferece um sistema de tipos comum, com recursos como reflexão e incorporação de meta-dados aos tipos. Uma extensa biblioteca de classes fornece uma API para os recursos da plataforma.

Os tipos do Common Type System (CTS), o sistema de tipos do CLR, se dividem em duas categorias: os *value-types*, ou valores, e os *reference types*, ou referências. A atribuição entre referências cria outra referência para os mesmos dados apontados pela referência que foi copiada; a atribuição entre valores cria uma cópia distinta do valor original. A Tabela 1.1 resume os tipos do CTS.

| Valores | Referências |
|---|---|
| tipos primitivos | classes |
| estruturas | interfaces |
| enumerações | arrays |
| | delegates |
| | ponteiros |

Tabela 1.1: Tipos do CTS

As referências vêm em diversos tipos, o principal deles sendo as *classes*. O CLR oferece herança simples para as classes, formando uma hierarquia que começa pela classe `System.Object`. Os objetos são instâncias das classes, alocadas no heap. As classes do CTS podem definir campos, propriedades, métodos e eventos. Propriedades são definidas por um par de métodos, um método para ler e outro para atribuir à propriedade, mas sintaticamente elas são como campos. Eventos são uma maneira de se registrar *callbacks* com um objeto, e são definidos por um par de métodos, um para adicionar e outro para remover um callback, e o tipo do callback (sua assinatura).

Outro tipo de referência são os *arrays*, vetores ou matrizes alocados no heap que contêm valores (ou referências) de um determinado tipo. As *interfaces* são tipos que só podem definir métodos e propriedades abstratos; outras classes podem implementar interfaces, e as interfaces podem herdar de outras interfaces. Os *delegates* são tipos que definem uma assinatura de um método; suas instâncias encapsulam métodos com essa assinatura, que podem ser chamados e combinados. Finalmente, os *ponteiros* são usados para passagem de valores por referência, em chamadas a métodos; eles apontam para um valor no heap ou na pilha de execução.

Os valores do CTS podem pertencer aos tipos primitivos, os escalares, ou a tipos definidos pelo usuário, as enumerações e estruturas. Os tipos primitivos do CLR estão resumidos na Tabela 1.2. Os tipos marcados com um asterisco (*) também fazem parte da Common Language Specification.

As enumerações são uma maneira de se associar nomes a valores inteiros, ou a bits em uma máscara de bits. As estruturas são um tipo composto, como as classes, podendo inclusive ter métodos e propriedades, mas que seguem a semântica de atribuição dos valores (e não possuem herança). A cada tipo de valor corresponde também um tipo de referência, usado para *boxing*, ou empacotamento dos valores no heap.

O sistema de execução do CLR é uma máquina virtual baseada em pilha, com cerca de 220 instruções (a Common Intermediate Language, ou

PUC-Rio - Certificação Digital Nº 0210489/CA

| Nome do Tipo | Valores |
|---|---|
| System.Boolean* | Booleanos (`true` ou `false`) |
| System.Byte* | Naturais de 8 bits |
| System.SByte | Inteiros de 8 bits |
| System.Char* | Caracter unicode de 16 bits |
| System.Double* | Ponto flutuante com precisão dupla |
| System.Single* | Ponto flutuante com precisão simples |
| System.Int32* | Inteiros de 32 bits |
| System.UInt32 | Naturais de 32 bits |
| System.Int64* | Inteiros de 64 bits |
| System.UInt64 | Naturais de 64 bits |
| System.Int16* | Inteiros de 16 bits |
| System.UInt16 | Naturais de 16 bits |

Tabela 1.2: Tipos primitivos

CIL). A unidade básica de execução são os métodos; cada método possui um registro de ativação, mantido na pilha de execução do CLR. O registro de ativação contém um vetor com as variáveis locais do método, um vetor com seus argumentos, informação sobre o método em execução, uma referência para o registro do método que chamou o método atual, e a pilha de avaliação do método. As instruções da CIL cobrem operações como transferência de valores de e para a pilha de avaliação, criação de valores e referências, chamadas a métodos, operações aritméticas, operações em vetores, etc.

PUC-Rio - Certificação Digital Nº 0210489/CA