Gerenciamento de Agentes Remotos no Projeto Subverse via *Scripts* embutidos em Linguagem LUA

Saulo Popov Zambiasi[1]

[1]Sistemas de Informação / Ciência da Computação
UNISUL – Universidade do Sul de Santa Catarina – Florianópolis – SC – Brasil
{saulopz@gmail.com}

Abstract. *Subverse Project is a learning platform in constantly evolution. It is focused to Computer Science courses and it is to work with a set of theories presented in university disciplines. The Subverse is presented as a virtual world to implementation of agents based in paradigm of games multi-players. However, when it is necessary some update in behavior of an agent, a recompilation and a new connection with Subverse virtual world is obligate. An alternative way is the utilization of scripts that can be used by the agents. These scripts has the advantage that can be updated in execution time. A script can be read by an agent and executed in conformity with its specifications. In that context, this paper present a propose for management of behaviors of remote agents in the Subverse Project via scripts, more specifically in the programming language Lua.*

Resumo. *O Projeto Subverse é uma plataforma de aprendizado em constante evolução, para que alunos de cursos de computação possam trabalhar na prática diversas teorias apresentadas nas disciplinas. Este se apresenta como um mundo virtual para a implementação de agentes baseados no paradigma de jogos multijogador. Contudo, quando há a necessidade de alteração da forma como um Agente se comporta no mundo virtual, é necessário desconectá-lo do mundo virtual, de uma recompilação e uma nova conexão ao mundo virtual. Uma forma alternativa é a utilização de* scripts *que possam ser alterados em tempo de execução, lidos pelo Agente e executados em conformidade com suas especificações. Neste contexto, este artigo apresenta uma proposta de gerenciamento dos comportamentos dos Agentes remotos no Projeto Subverse via scripts, mais especificamente escritos em linguagem de programação Lua.*

1. Introdução

O Projeto Subverse, ou simplesmente Subverse, é um mundo virtual para a implementação de agentes baseado no paradigma de jogos multijogador. Este tem como objetivo servir como plataforma para o ensino universitário em cursos de computação [Benin, 2007], [Silva, 2008]. Por ser um ambiente que permite a multidisciplinaridade, este ambiente de aprendizado se apropria de várias disciplinas, tais como: computação gráfica, para a visualização do mundo virtual e dos agentes inseridos neste; técnicas de desenvolvimentos de jogos de computador; sistemas distribuídos com múltiplos agentes

conectados ao ambiente remotamente; Inteligência Artificial com a implementação de algoritmos inteligentes nos agentes que devem interagir no mundo virtual; entre outras [Zambiasi at al, 2012].

A participação no processo para o aprendizado proporciona ao aluno desafios reais. Este passa a não apenas adquirir conhecimentos teóricos e de utilização de softwares prontos, mas também passa ter o entendimento da estrutura, organização e demais elementos pelo processo da prática de desenvolvimento e da evolução do sistema em si. Assim, o Projeto Subverse se mostra como uma maneira de apresentar ao aluno esta possibilidade de participarem de todo o processo do desenvolvimento do sistema. Além disso, o sistema pronto objetiva oferecer também uma plataforma de desenvolvimento de algoritmos para aprendizado de diversas disciplinas.

Um Agente no mundo virtual do Subverse pode executar seus comportamentos como forma de atingir seus objetivos. Contudo, se algo deve ser alterado nesse comportamento, para que o agente possa ser melhorado ou modificado, é necessário parar sua execução, executar uma nova compilação no código fonte e então novamente carregá-lo no mundo virtual. Uma forma que poderia ser interessante para melhorar esse procedimento, seria a modificação da forma como o Agente age em tempo de execução. A utilização de *scripts*, conforme citado por Celes et al. (2011), poderia ser uma forma de se resolver este problema, alterando os *scripts* sem a necessidade de parar a execução do agente ou de nova recompilação.

Dessa forma, este artigo propõe uma forma alternativa de gerenciamento dos agentes, conectados remotamente ao Subverse, via utilização de *scripts*. no caso específico dessa proposta, é utilizada a linguagem de programação Lua.

O presente artigo está organizado da seguinte forma: Na seção 2 é apresentado o Projeto Subverse e os assuntos envolvidos em sua estrutura de execução atual. A seção 3 apresenta a proposta do gerenciamento via *scripts*. Na seção 4 são apresentadas as considerações finais.

2. Projeto Subverse

O Projeto Subverse se caracteriza por um conjunto de implementações que se mantêm em constante evolução. Módulos não desenvolvidos se mostram como fonte de pesquisa para a implementação de tecnologias atuais e módulos já prontos se mostram como estruturas que podem ser estudadas, melhoradas e adaptadas. Ainda, na forma de uma implementação de um mundo virtual que segue a filosofia dos jogos multijogadores, o Projeto Subverse fornece uma estrutura para conexões de múltiplos agentes para interagirem entre eles. Estes agentes são, por si, estruturas algorítmicas que podem ser implementadas como forma de aplicação de teorias de diversas disciplinas da computação. Contudo, para tal, o Projeto Subverse segue a filosofia dos sistemas *opensource*, tão conhecidos e disseminados na Internet hoje em dia. Ainda, para a implementação, sugere-se que sejam utilizadas também tecnologias *opensource*, gratuitas e atuais disponíveis.

No Subverse, diversos agentes podem ser implementados utilizando teorias de inteligência artificial, sistemas multiagentes, redes neurais artificiais, lógica, lógica difusa, sistemas especialistas e outras, a fim de estudo e aperfeiçoamento dos comportamentos destes [Benin, 2007]. Os agentes, especificamente, se caracterizam como algo que, provido de sensores, recebe informações do ambiente ao seu redor e toma ações, interagindo com este ambiente por meio de mecanismos denominados de atuadores.

Além disso, os agentes devem poder se comunicar com outros para poderem alcançar seus objetivos [Russel e Norvig, 2004]. No caso dos agentes do Subverse, estes são agentes de software e seus sensores e atuadores estão no contexto de um mundo virtual criado para a interação entre os agentes.

Um mundo virtual pode ser visto como uma arquitetura que suporta interações em tempo real em um mundo virtual. Esta arquitetura deve ser flexível o suficiente para suportar as tarefas que se deseja executar [Calvin et al., 1993]. Uma outra perspectiva de um mundo virtual é a de um ambiente artificial criado em um computador que contém especificidades mínimas de um mundo real. Estes sistemas são multiusuários, permitindo que muitas pessoas interajam simultaneamente. Os mundos virtuais são usados hoje em dia para diversas aplicações, tais como salas de conversa na Internet, vídeo conferência, jogos multijogadores, etc. Essas aplicações requerem que a interação seja feita em tempo real e intentam ser o mais realista possível [Pulo e Houle, 2001].

O Subverse pode ser comparado, em um certo ponto de vista, a jogos multijogadores. Estes geralmente utilizam uma arquitetura de comunicação cliente-servidor. Os clientes enviam mensagens para um servidor responsável pelo processamento do jogo e recebem mensagens processadas para mostrar em uma interface gráfica ao usuário. Para isso, um *middleware* pode promover certa modularidade de programação, facilitando o desenvolvimento de aplicações distribuídas, uma vez que o desenvolvedor não precisa se preocupar em escrever o código para gerenciar interações entre os processos. Entretanto, a comunicação entre o *middleware* e processos incorre em custos adicionais se comparado com a comunicação direta entre os processos [Dickey et al., 2004].

Em tempo, existem mundos virtuais *opensource* na internet, incluindo jogos multijogadores que possibilitam a conexão de agentes, ou *bots*. Porém, a documentação é em geral fraca e o tempo de aprendizado do sistema seria ampliado devido a necessidade da leitura de diversas linhas de código prontas. Isso, na maioria das vezes desestimula os novos desenvolvedores a se unirem ao projeto ou mesmo a alunos que estão em fase de aprendizado se arriscarem nesses sistemas. Existem também trabalhos, como o de Adobbati e Marshall (2001), já citado, que apresentam uma proposta bastante correlata com a apresentada aqui. Neste, é apresentada uma estrutura para que agentes possam se conectar à um mundo virtual 3d de um jogo multijogador chamado *Unreal Tournament* para interagirem na forma de um sistema multiagente. Porém, como já explicado, o objetivo do Projeto Subverse é possibilitar aos estudantes a participação em todo o processo de desenvolvimento e evolução do sistema como um todo.

2.1. Arquitetura

Algumas características são necessárias para que os elementos ativos, agentes, do mundo virtual possam interagir com os outros elementos e com o mundo virtual, propriamente dito. Essas compõem o modelo da arquitetura, composto por sistemas e módulos organizados conforme o paradigma cliente-servidor, conforme apresentada na Figura 1.

Os elementos da arquitetura são o Navegador Web, a Interface Visual e os Agentes, do lado do Cliente, e do lado do Servidor estão localizados um Banco de Dados, o Servidor Web, o Mundo Virtual e o Avatar [Zambiasi at al, 2009].

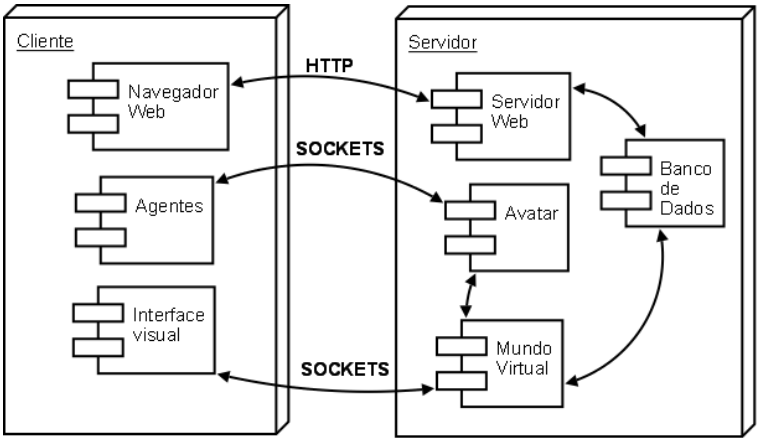

Figura 1: Arquitetura do Subverse.

O **Servidor Web** torna possível o acesso dos usuários às configurações e criação dos agentes que interagem no mundo virtual por meio de uma aplicação Web de forma bastante interativa. É necessário uma interface para permitir que o usuário acesse de qualquer lugar e a qualquer momento essas informações, tal como uma interface Web.

O **Banco de Dados** se apresenta como uma área para o armazenamento das informações de cada agente de forma persistente. Isto permite que o agente não perca suas informações, características e histórico quando ele está desconectado do mundo virtual ou adormecido.

O **Mundo Virtual**, propriamente dito, é a base de execução dos elementos ativos no mundo virtual. Este é responsável pela a leitura das informações persistentes quando esses são instanciados, pela alocação de um avatar para cada agente que se conecta e pela interação entre os elementos do mundo virtual. Este é caracterizado como o núcleo de processamento e comunicação dos agentes.

O **Avatar** é o *proxy* do agente no mundo virtual, ou seja, um módulo de *software* instanciado pelo mundo virtual que serve para representar cada agente neste. Este faz basicamente o trabalho de comunicação entre o agente e o mundo virtual, representando as ações do agente no mundo virtual e enviando os retornos dessas ações ao agente.

O **Navegador Web** provê um meio de acesso para a criação de novos agentes e para a configuração das informações dos agentes via Web. Por definição deste projeto, apenas agentes cadastrados podem acessar o mundo virtual, como forma de segurança, maneira de manter as informações persistentes desses no sistema e para o registro de ações (*logs*) dos agentes para possíveis estatísticas e análises de execução, tanto para os desenvolvedores do sistema, como para os usuários criadores e implementadores de agentes.

A **Interface Visual** é um ambiente gráfico baseado em jogos de computadores para fornecer aos usuários uma forma de observar visualmente os agentes em execução no mundo virtual. Neste projeto, uma primeira versão deste módulo segue as seguintes formas de visualização: modo Avatar (é criado um agente para representar o usuário no mundo virtual, este pode interagir com os outros agentes como se fosse um agente); modo Deus (apenas navega no sistema, com as setas direcionais, podendo visualizar o ambiente enquanto este se encontra em funcionamento) e modo Agente (acompanhando um agente específico enquanto este interage, por sua própria programação, com o

mundo virtual e outros agentes).

Os **Agentes**, por fim, são os elementos de *software* que, via um meio de comunicação remoto com o seu representante no mundo virtual, o Avatar, interagem com o mundo virtual e com outros agentes. A programação das técnicas de Inteligência Artificial podem ser implementadas neste módulo. Deve ser possível que cada usuário do mundo virtual possa criar diversos agentes, mas para cada agente é necessário um cadastro próprio e um avatar para representá-lo no mundo virtual.

2.2. *Middleware* de Comunicação

O Projeto Subverse sugere uma estrutura distribuída para múltiplas conexões de agentes remotos para interação entre eles no mundo virtual do Subverse [Zambiasi et al. 2009], tal como um MMOG (*Massive Multiplayer Online Game*). Contudo, no lugar de jogadores controlando avatares remotamente, os elementos que devem controlar os avatares são agentes desenvolvidos pelos usuários. Os usuários iniciam seus agentes e visualizam seu ciclo de vida no jogo por meio de uma interface gráfica que também se conecta remotamente ao jogo.

Em jogos MMOG, os clientes são utilizados pelos jogadores para se conectar ao servidor do jogo. Dessa forma, o jogador pode enviar informações da ação que deseja realizar e recebe informações atualizadas das atividades que estão ocorrendo à volta do seu personagem (avatar) no jogo. O servidor possui o estado geral do mundo do jogo e coordena a atividade dos jogadores [Balan et al. 2005]. Muitos desses jogos trabalham baseados no paradigma cliente-servidor. Neste estilo os clientes trocam mensagens com um servidor, que é a parte responsável pelo processamento do jogo [Dickey et al. 2004]. O servidor, em execução, abre um soquete de comunicação e fica aguardando clientes se conectarem. Um cliente estabelece uma conexão com o servidor por meio de um endereço IP (*Internet Protocol*) e uma porta, solicitando a conexão [Silberschatz et al. 2001].

A utilização de soquetes é uma ótima escolha para se trabalhar com a comunicação de processos por ser um método rápido e simples. Contudo, se por um lado, a sua utilização agiliza o processo da comunicação entre os processos, por outro, dificulta pela necessidade de se tratar as mensagens em um baixo nível. Sendo assim, para facilitar a utilização de soquetes, é interessante a utilização de um *middleware* de comunicação, a fim de promover a modularidade e maior facilidade na programação de sistemas distribuídos. Por meio desse recurso, "o desenvolvedor não precisa se preocupar em escrever o código para gerenciar interações entre os processos" [Silva 2008].

Além de facilitar ao desenvolvedor, os *middlewares* de comunicação buscam fornecer portabilidade, transparência, interoperabilidade aos sistemas distribuídos e ocultam detalhes de implementações [Silva 2008]. Para Coulouris et al. (2001), tal camada busca mascarar a heterogeneidade dos sistemas, fornecendo um conjunto de funções para a troca de mensagens, dispensando-o da preocupação de como esta vai ser enviada ou recebida.

Pelo motivo dos agentes no Subverse trabalharem remotamente, há então a necessidade de um controle de troca de mensagens e uma forma de permitir múltiplas conexões. Desse modo, para facilitar o desenvolvimento desses agentes, sem que o desenvolvedor precise se preocupar com os detalhes da comunicação, este artigo apresenta uma proposta de *middleware* de comunicação para o Subverse. A proposta é

baseada no paradigma cliente-servidor, e conexões via soquetes, com troca de mensagens. As mensagens são as representações das chamadas de métodos com uma estrutura de variáveis, com os nomes, tipos e valores, além do nome do método que está sendo chamado.

De forma a facilitar o desenvolvimento do Subverse na comunicação entre o cliente e o servidor via soquetes, surgiu a necessidade do desenvolvimento de um *middleware* de comunicação. Assim, tendo como inspiração a forma de comunicação do RPC (*Remote Procuedure Call*), visto em Coulouris et all. (2001), foi feita uma implementação de um *middleware* que fornece ao desenvolvedor um conjunto de métodos que, para ele, é como se estivesse acessando diretamente os métodos no próprio servidor. Esses métodos, são implementados na forma de criar uma mensagem com as informações do tipo do método que está sendo chamado e o conjunto de variáveis que estão sendo enviadas.

Conforme visto na Figura 2, o agente acessa o MAPA (Métodos de Acesso ao *Proxy* do Agente). Por sua vez, o *Proxy* do Agente se utiliza do MAA (Métodos de Acesso ao Agente) para se comunicar com o Agente. Da mesma forma se dá no lado do Visualizador com o MAPV (Métodos de Acesso ao *Proxy* do Visualizador) e o MAV (Métodos de Acesso ao Visualizador).

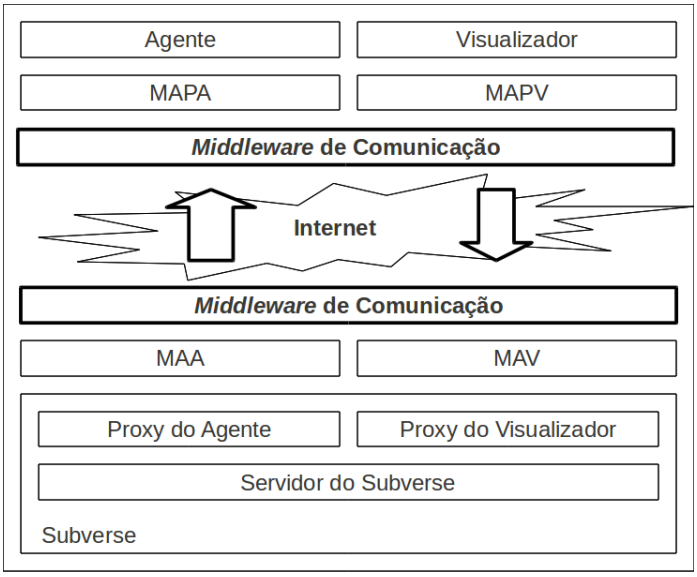

Figura 2: Middleware de comunicação

O conjunto de métodos MAPA, MAPV, MAA e MAV, utilizam o *middleware* de comunicação do Subverse diretamente, criando mensagens com o nome do método a ser chamado e as variáveis envolvidas na chamada do método, tanto de entrada como de saída (retorno).

As chamadas dos métodos, no Subverse, se dão na forma de comunicação assíncrona, ou seja, um método é chamado para efetuar uma determinada operação e outros métodos são utilizados para saber o status da chamada desse método. Por exemplo, se um agente quiser fazer uma movimentação para frente, um método "mover" é chamado com o valor "frente" para a variável "direção". O desenvolvedor deve então utilizar o método "moveu" que retorna null, false ou true para saber se a operação foi executada com sucesso. Caso o retorno for null, então ainda não foi retornada uma resposta, true caso conseguiu mover e false caso a operação não foi possível.

Por sua vez, o *middleware* de comunicação do Subverse (Figura 3) é composto de alguns módulos principais. Os Métodos de Criação e Extração de Variáveis são responsáveis por criar uma variável com nome, tipo e valor, e métodos correspondentes para recuperar os nomes, tipos e valores das variáveis. Uma mensagem, que é a informação trocada entre o cliente e o servidor, é formada por uma ou mais variáveis, formando uma estrutura de variáveis. Quando o usuário termina a criação de uma estrutura, ele utiliza o método de empacotamento dessa estrutura, criando uma mensagem e enviando-a para a Fila de Mensagens para Envio. O Transmissor fica responsável por enviar essa mensagem ao servidor, quando possível. Cada mensagem possui um código, de forma a poder interpretar de qual requisição um determinado retorno se refere.

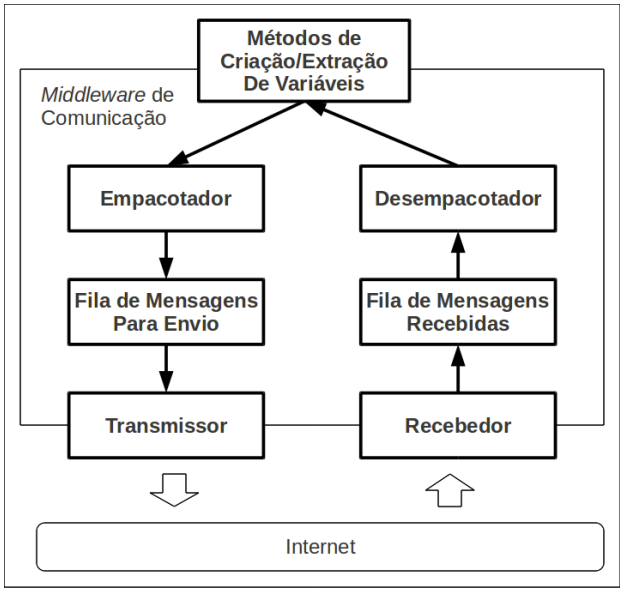

Figura 3: middleware de comunicação

O recebimento de uma mensagem inicia o processo contrário. Quando o recebedor pega uma mensagem, a coloca em uma Fila de Mensagens Recebidas, que pode ser desempacotada e interpretada, para efetuar a chamada da qual a mensagem é responsável para executar. Todo o processo de troca de mensagens funciona na forma de multitarefa, deixando tanto o cliente quanto o servidor livres para efetuarem as tarefas de processamento das quais são responsáveis, não necessitando o usuário desenvolvedor ficar preocupado com o processo de troca de mensagem. Além disso, o *middleware* do servidor aceita conexões de múltiplos clientes.

2.3. Implementação Embutida

Segundo Celes et al. (2011), a maioria dos jogos se utilizam do auxílio de alguma linguagem de script embutida no software. Essas linguagens normalmente são interpretadas, com tipagem dinâmica e gerenciamento automático de memória. A construção de estruturas de informações são bastante dinâmicas, fornecendo aos usuários uma forma interessante de ampliar as possibilidades do software em tempo de execução e sem a necessidade de uma nova compilação do código do software implementado.

O funcionamento dessa ideia se dá por meio da implementação de um script em um programa hospedeiro, desenvolvido em uma linguagem de programação não

necessariamente igual ao programa em si (por exemplo, o *script* em linguagem LUA e o programa em C++). Uma outra característica, que somada com a anterior torna a implementação na forma de *scripts* uma excelente ferramenta de desenvolvimento, é que essas linguagens também não devem poder acessar execuções não autorizadas no programa hospedeiro (Celes et al., 2011). Várias vantagens podem ser identificadas pela utilização de uma linguagem de *script* acoplada em um software (normalmente utilizadas em jogos de computador). O *script* pode ser utilizado para definir configurações, objetos e seus comportamentos, gerenciar algoritmos de inteligencia artificial e tratar os eventos de entrada (Celes et al., 2011).

Em Celes et al. (2011), a linguagem Lua é apresentada como uma linguagem de *script* bastante atual e como uma das mais utilizadas no desenvolvimento de jogos. Isso devido "ao seu pequeno tamanho, bom desempenho, portabilidade e facilidade de integração". Essa linguagem é extensível e projetada para oferecer metamecanismos que facilitam a adequação da linguagem às necessidades da aplicação.

3. Proposta

A proposta desse trabalho se firma na forma como ocorre a execução dos comportamentos do Agente. Em sua base, deseja-se evitar a recompilação do agente caso seja necessário a alteração do comportamento do mesmo, i.e. da maneira como este deve agir no mundo virtual. Em tempo, também é interessante que esses mesmos comportamentos possam ser modificados em tempo de execução do agente.

Um *Script* desenvolvido em linguagem de programação Lua se caracteriza como um arquivo texto contendo um conjunto de especificações algorítmicas para serem seguidas por um interpretador. No caso da proposta, cada um desses *scripts* é visto como um comportamento para o Agente. O agente possui um repositório desses *scripts* que podem ser acessados por um "Seletor de Comportamentos" que fica localizado no programa do Agente. Quando um comportamento é selecionado, este entra em execução para satisfazer as necessidades do Agente e para alcançar seus objetivos. Na Figura 4 pode ser vista uma representação gráfica da utilização de *scripts* em linguagem de programação Lua pelo Agente.

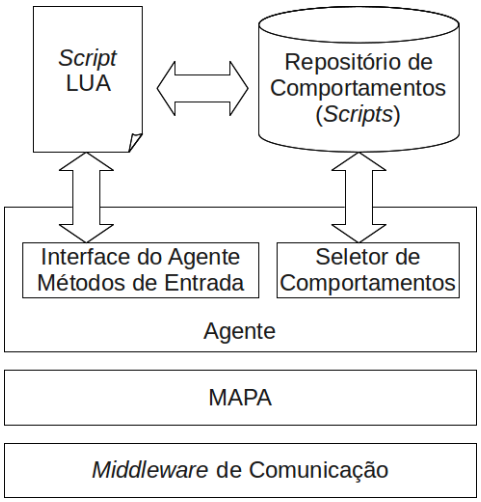

Figura 4: Comportamentos do agente via scritps.

O módulo de "Interface do Agente/Métodos de Entrada" da Figura 4 é classe do Agente com um conjunto de métodos públicos que o programador pode utilizar para

compor o comportamento do seu Agente. Contudo, essa classe está localizada no dispositivo computacional do cliente e o comportamento do Agente torna-se efetivo no mundo virtual apenas do lado do servidor. Essa classe, do lado do cliente, apenas faz o mapeamento dos métodos para um conjunto de mensagens que são enviadas para o servidor. Do lado do servidor, essas mensagens são recebidas e transformadas em chamada de métodos na classe do Avatar (sessão 2.1.), executadas e, se for o caso, um retorno é enviado de volta ao cliente. Todo esse processo é feito pelo MAPA e pelo *Middleware* de Comunicação visto na sessão 2.2..

Ainda, do lado do cliente, é feito o mapeamento para a linguagem Lua por meio dos recursos de biblioteca. Como exemplo, toma-se a classe de Agente da Figura 5 com parte dos métodos do Agente.

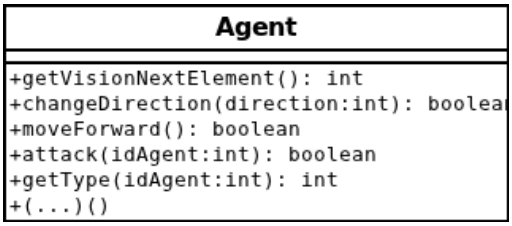

Figura 5: Representação da classe do Agente.

Para a execução do *script* criado pelo usuário, uma função (neste caso em linguagem de programação C++) é chamada passando como parâmetro o *script* na forma de uma *string* – pode ser um arquivo carregado em uma *string* ou informações texto de um banco de dados. Essa função (Figura 6) efetua o mapeamento dos métodos da classe do Agente para a linguagem Lua e executa o *script* do Agente criado pelo usuário.

```
1 #include "Agent.h"
2
3 void Lua_run(string script) {
4     lua_State *L = lua_open();
5     luaL_openlibs(L);
6     open(L);
7     module(L)[
8         class_<Agent>("Agent")
9             .def(constructor<>())
10            .def("getVisionNextElement", &Agent::getVisionNextElement)
11            .def("changeDirection", &Agent::changeDirection)
12            .def("moveForward", &Agent::moveForward)
13            .def("attack", &Agent::attack)
14            .def("getType", &Agent::getType)
15            // other methods of Agent
16    ];
17    if (luaL_dofile(L, script.c_str())) {
18        cerr << lua_tostring(L, -1) << endl;
19    }
20    lua_close(L);
21 }
```

Figura 6: Mapenamento da classe do Agente para o script Lua.

Na Figura 7 é possível visualizar um exemplo de um comportamento simples de um agente escrito em *script* Lua e baseado na estrutura parcial do Agente aqui apresentada. Este instancia um objeto no programa do Agente em execução com o nome "Agent" e pode efetuar chamada aos seus métodos para fazer o Agente agir no mundo virtual do Subverse.

```
1  -- Simple Agent Lua
2  require "globals"
3
4  a = Agent()
5  idElement = a:getVisionNextElement()
6  if idElement ~= 0 then
7      if a:getType(idElement) == MONSTER then
8          a:attack(idElement)
9      end
10 end
11 a:changeDirection(LEFT)
12 a:moveForward()
```

Figura 7: Exemplo de um comportamento simples.

No caso do exemplo de *script* (Figura 7), um arquivo de nome *globals.lua* é incluído. Este contem algumas constantes que são utilizadas pelo *script* (por exemplo: *MONSTER*, *LEFT*, *RIGHT*, etc.). Na linha 4, a classe do Agente, já instanciada no programa em C++ em execução é acessada por meio de uma variável chamada *a*. Na linha 5, o *script* utiliza um método da classe do Agente que retorna o próximo elemento que está na visão do Agente. Esse método retorna 0 se não houver mais nenhum elemento ou um inteiro positivo com o *id* do elemento visualizado. Seguindo o algoritmo do *script*, se o *id* do elemento do elemento seja diferente de 0, verifica se o elemento é do tipo *MONSTER*. Caso positivo, efetua um ataque a este elemento. Nas linhas 11 e 12 o Agente vira para a esquerda e dá um passo a frente.

A cada ciclo de execução de um *script*, o usuário pode executar apenas um conjunto de atividades. Por exemplo, um Agente não pode efetuar dois passos para frente num mesmo ciclo de execução. Comparativamente, é como pessoas em um jogo de cartas. Cada pessoa pode, em sua vez, efetuar apenas um conjunto de ações. A vez é passada para os próximos jogadores até que a pessoa possa jogar novamente.

4. Considerações Finais

Este artigo apresentou uma proposta para o gerenciamento dos Agentes que estão conectados ao mundo virtual do Projeto Subverse na forma de *scripts*. A linguagem de programação no contexto de *scripts* sugerida foi a linguagem Lua e a proposta se manteve nas bases de toda a estrutura e arquitetura do Projeto Subverse já existente.

Um protótipo foi implementado apenas para os testes da utilização de *scripts* em objetos/agentes. As chamadas dos métodos se deram de maneira correta com o que era esperado. Também foi possível alterar o comportamento do agente em tempo de execução pela alteração do arquivo de *script* escrito em linguagem de programação Lua. Contudo, ainda não foi implementada essa forma de gerenciamento dos comportamentos dos agentes no Projeto Subverse especificamente.

Por fim, este trabalho veio como uma contribuição na estrutura do Projeto Subverse, trazendo mais uma sugestão de implementação dos agentes e gerenciamento dos mesmos de forma alternativa, facilitando o controle, evolução e alteração do comportamento dos agentes em tempo de execução. Os próximos passos se dá em utilizar as especificações utilizadas para a criação do protótipo de testes dessa proposta como base de implementação dos agentes remotos no sistema do Projeto Subverse.

5. Agradecimentos

Este trabalho é parcialmente financiado pela PUIP – Programa Unisul de Incentivo à Pesquisa (http://www.unisul.br).

Referências

Adobbati, R., Marshall, A. N., at all. ***Gamebots: a 3d virtual world test-bed for multi-agent research***. Agentes'01. Montreal – Canada. May-June, 2001.

Balan, R., Ebling, M., Castro, P. and Misra, A.. ***Matrix: adaptative middleware for distributed multiplayer games***. *Journal Middleware. Springer*. 390-400. 2005.

Benin, M.. **Evolução de NPC's e adversários em jogos de computador usando algoritmos genéticos**. Monografia de graduação. Faculdades Barddal. Florianópolis, SC. 2007.

Calvin, J., Dickens, A., Gaines, B. ***The simnet virtual world architecture***. IEEE, 1993.

Celes, W. de Figueiredo, L.H. and Ierusalimschy, R.. **A Linguagem Lua e suas Aplicações em Jogos**. Rio de Janeiro, 2004. Disponível em: <http://www.tecgraf.puc-rio.br/~lhf/ftp/doc/wjogos04.pdf> Acessado em Ago/2011.

Coulouris, G., Dollimore, J. and Kindberg, T.. *D**istributed system: concepts and design***. Addison-Wesley. 2001.

Dickey, C.G, Zappala, D. and Lo, V., A ***Distributed Architecture for massively multiplayer online games***. ACM NetGames Workshop. 2004.

Pulo, K., Houle M. E., ***Evolution of Virtual World System***. IEEE, 2001.

Russel, S., Norvig, P., **Inteligência Artificial**. Tradução da segunda edição. Rio de Janeiro: Elsevier, 2004.

Silberschatz, A., Galvin, P., Gagne, G.. **Sistemas operacionais: conceitos e aplicações**. Editora Campus. 6a.ed. 2001.

Silva, F.C., 2008. **Middelware de comunicação para jogos multiplayer: um estudo de caso no Projeto Subverse**. Monografia de graduação. Faculdades Barddal. Florianópolis, SC.

Zambiasi, S.P., Benin, M. and Silva, F.C.. **Projeto Subverse, um mundo virtual baseado em jogos multijogadores como ferramenta de ensino multidisciplinar em cursos de tecnologia da informação**. SCGames. 2009.

Zambiasi, S.P.; Oberderfer, L.P.Z.B.; Benin, M.R. ***Asynchronous Remote Management of Agents for Subverse Project***. In IEEE *Latin America Transactions*, vol.10, no.1, pg.1289-1294. Jan, 2012.