NCLua SOAP: Acesso a *Web Services* a partir de aplicações de TV Digital Interativa

Manoel Campos da Silva Filho[1], **Paulo Roberto de Lira Gondim**[2]

[1]Coordenação de Informática
Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO)
AE 310 Sul, Av LO 05 S/N, Plano Diretor Sul, 77.021-090, Palmas–TO, Brazil

[2]Departamento de Engenharia Elétrica
Universidade de Brasília (UnB)
Campus Universitário Darcy Ribeiro, 70.910-900, Brasília–DF, Brazil

mcampos@ifto.edu.br, pgondim@unb.br

Abstract. *This paper describes the implementation of a module, written entirely in Lua language, to access Web Services in interactive Digital TV applications. Such applications can be developed using NCL and Lua languages for the Ginga-NCL subsystem of the Ginga middleware. The module allows the convergence between Digital TV applications and the Web, the use of existent services in a Digital TV graphical interface, besides the details encapsulation of the HTTP and SOAP protocols.*

Resumo. *O presente artigo descreve a implementação de um módulo, escrito inteiramente em linguagem Lua, para acesso a Web Services em aplicações de TV Digital interativa (TVDi). Tais aplicações podem ser desenvolvidas utilizando as linguagens NCL e Lua para o sub-sistema Ginga-NCL do middleware Ginga. O módulo permite a convergência entre aplicações de TV Digital e a Web, o uso de serviços já existentes a partir de uma interface gráfica de TV Digital, além do encapsulamento de detalhes dos protocolos HTTP e SOAP.*

1. Introdução

Um dos grandes atrativos da TV Digital (TVD) é com certeza a interatividade. Existem diversas categorias de aplicações interativas como jogos, informações (notícias, horóscopo, previsão do tempo, etc), educação (*T-Learning*), governo eletrônico (*T-Government*), comércio eletrônico (*T-Commerce*), saúde (*T-Health*), bancárias (*T-Banking*) e outras, como apresentado em [**?**], [**?**] e [**?**].

O nível de interatividade das aplicações vai desde a chamada interatividade local (onde os usuários/telespectadores podem acessar apenas os dados enviados pela emissora, por radiodifusão) até a interatividade plena (onde os usuários/telespectadores dispõem de um canal de retorno, permitindo enviar e receber dados em uma rede como a *Internet*) [**?**].

Tais aplicações que permitem interatividade plena, precisam utilizar protocolos de comunicação padrões e consolidados na *Internet* (como TCP, HTTP e SOAP) para garantir a interoperabilidade com outros sistemas.

Utilizando os protocolos citados, é possível garantir a convergência entre *Web* e TV. Com isto, as aplicações de TVDi podem ser enriquecidas com conteúdo proveniente

da *Internet*, como é o caso da aplicação que busca conteúdo na Wikipédia, baseada nas *tags* do programa em exibição, obtidos a partir do Guia Eletrônico de Programação (cujas informações são enviadas pela emissora), como apresentado em [**?**].

No entanto, a norma do sub-sistema Ginga-NCL do *middleware* Ginga define apenas a obrigatoriedade do protocolo TCP. Quaisquer protocolos acima da camada de transporte precisam ser implementados pelo desenvolvedor de aplicações.

Tendo em vista o cenário supracitado, são apresentados neste artigo os projetos NCLua HTTP e NCLua SOAP: implementações dos protocolos HTTP e SOAP, respectivamente, para o sub-sistema Ginga-NCL, utilizando linguagem Lua, que permitem a convergência entre aplicações de TVDi e a *Web*.

A escolha de implementação de HTTP e SOAP partiu da inexistência de versões livres e de código aberto de tais protocolos para o Ginga-NCL. Até onde sabe-se, o NCLua HTTP e NCLua SOAP são as primeiras implementações livremente disponibilizadas.

2. Delimitação do Problema

Apesar de Lua ser uma linguagem extensível [**?**], principalmente pela capacidade de utilizar módulos construídos em linguagem C, e de existirem vários destes módulos para as mais diversas finalidades, tais módulos binários não podem ser utilizados em aplicações interativas de TV Digital enviadas via *broadcast*, devido a questões de segurança, uma vez que um módulo escrito em linguagem C pode ter acesso a qualquer funcionalidade do sistema operacional (embarcado juntamente com o *middleware* no receptor de TV Digital).

Em [**?**] são citadas algumas ameaças de segurança em sistemas de TVDi, como transação fraudulenta (perda/roubo de dados), pirataria de conteúdo, falsificação, violação ou corrupção das aplicações e uso ilegítimo de serviços do provedor.

Tais ameaças podem ser potencializadas com o uso de módulos binários. Além do mais, a compilação de módulos em C gera código nativo dependente de plataforma, o que não garante que a aplicação executará em qualquer receptor [**?**]. Somente aplicações residentes podem ser desenvolvidas em linguagens compiladas como C.

Com isto, para haver, em aplicações NCLua de TVDi, algumas das funcionalidades dos módulos binários citados, é preciso implementar módulos inteiramente em linguagem Lua, cujo ciclo de vida é controlado pelo *middleware* Ginga[**?**].

3. Trabalhos Relacionados

Nesta seção são apresentadas as tecnologias envolvidas no desenvolvimento da solução apresentada e os trabalhos relacionados.

3.1. Tecnologias de *Web Services*

SOAP é um protocolo padrão da *World Wide Web Consortium* (W3C) que permite a aplicações oferecerem seus serviços na *Internet*, em uma arquitetura distribuída no modelo cliente/servidor. O protocolo permite troca de mensagens e chamadas de procedimentos remotos. Tais serviços podem ser providos e consumidos por aplicações desenvolvidas em diferentes linguagens e plataformas. Isto permite a interoperabilidade entre diferentes

aplicações, por meio da troca de documentos XML, usando algum protocolo de transporte como o HTTP [**?**] [**?**] [**?**].

Um dos grandes benefícios do protocolo SOAP para a integração de aplicações é a sua linguagem para descrição dos serviços disponibilizados, a *Web Service Description Language* (WSDL). De forma padronizada, manual ou automatizadamente, uma aplicação cliente pode conhecer os serviços disponibilizados por um *Web Service* lendo o documento WSDL do serviço (também em formato XML)[**?**].

Os *Web Services* permitem a construção de aplicações distribuídas pela *Internet*, garantindo a centralização de regras de negócios em servidores de aplicações, encarregados de toda a carga de processamento de tais regras. Considerando-se isto, *Web Services* são ideais para aplicações clientes executando em sistemas com restritos recursos de *hardware*, como celulares e *Set-top Boxes*, estes últimos sendo o foco do presente trabalho. Além de desonerar os clientes da carga de processamento, alterações nas regras de negócio não requerem a atualização das aplicações clientes.

3.2. Lua e os *scripts* NCLua

Lua é a linguagem imperativa utilizada pelo sub-sistema Ginga-NCL para o desenvolvimento de aplicações procedurais. Ela tem como grandes vantagens sua simplicidade, eficiência e portabilidade. Tais características são extremamente importantes em uma linguagem a ser utilizada em dispositivos com recursos de hardware restritos, como os conversores de TV Digital (*Set-top Boxes*). Além de tudo, Lua é livre de *royalties*, o que permite que a mesma seja embarcada em *Set-top Boxes* sem onerar o custo dos equipamentos. Este é um requisito muito importante, considerando as características sócio-econômicas do Brasil, a intenção do governo de utilização da TV como meio para inclusão digital e sua presença na grande maioria dos lares brasileiros.

Como a linguagem NCL é apenas declarativa, a inclusão de características imperativas a uma aplicação de TVDi para o Ginga-NCL é possibilitada por meio dos chamados NCLua, *scripts* Lua funcionando como nós de mídia dentro de um documento NCL [**?**] [**?**].

Tais *scripts* aumentam o poder das aplicações de TVDi, possibilitando a construção de aplicações sofisticadas, seguindo paradigmas como Programação Estruturada e Programação Orientada a Objetos, com definição de regras de negócio na aplicação, interoperabilidade com sistemas na *Internet* por meio de protocolos como HTTP e SOAP, entre outros recursos.

O que diferencia os *scripts* NCLua de *scripts* Lua convencionais é a possibilidade de comunicação bidirecional entre este e o documento NCL. Tal comunicação acontece por meio de eventos, como definido na norma do Ginga-NCL em [**?**]. Segundo [**?**] "essa integração deve seguir critérios que não afetem os princípios da linguagem declarativa, mantendo uma separação bem definida entre os dois ambientes". Para isto, nenhuma alteração nas linguagens NCL ou Lua foi necessária, o que garante a evolução independente das linguagens, como ressalta [**?**]. Tal integração foi feita para ser minimamente intrusiva, garantindo uma separação entre a forma declarativa e a procedural de desenvolver uma aplicação para o Ginga-NCL. Assim, o código NCLua deve ser escrito obrigatoriamente em um arquivo separado do NCL. Isto permite uma clara divisão de tarefas entre profissionais da área de *design* e da área de programação [**?**]. A integração entre outras

linguagens como HTML e *JavaScript* é bastante intrusiva, onde, muitas vezes, código *JavaScript* é intercalado com código HTML.

O tratamento de eventos e as particularidades associadas a aplicações de TVDi, desenvolvidas em linguagem Lua, são implementadas por módulos adicionais à Linguagem, definidos na norma do Ginga-NCL [**?**]. Isto permite que a linguagem se mantenha inalterada para o contexto de TV Digital. Os módulos disponíveis em NCLua, utilizados no presente artigo são: ***event***, que permite a comunicação bidirecional entre um NCL e um NCLua; e ***canvas***, que disponibiliza uma API para desenhar imagens e primitivas gráficas.

3.3. Protocolo TCP no Ginga-NCL

A norma do Ginga-NCL [**?**] define a disponibilidade do protocolo TCP para ser utilizado pelas aplicações interativas. Como os objetos NCLua têm a característica de poderem ser orientados a eventos, o módulo *event* permite a captura e tratamento de eventos, possibilitando a comunicação assíncrona entre o formatador NCL e um objeto NCLua.

A implementação do protocolo TCP deve ser disponibilizada por meio do módulo *event*. A norma especifica uma classe de eventos denominada *tcp*. Assim, por meio das funções do módulo *event*, um objeto NCLua pode enviar requisições e receber respostas usando tal protocolo.

Devido à característica assíncrona do módulo *event*, o tratamento de requisições TCP em NCLua não é trivial. Para facilitar o uso de tal protocolo, pode-se recorrer ao recurso de co-rotinas da linguagem Lua. Segundo [**?**], uma co-rotina é similar a um *thread* (no sentido de *multithreading*). No entanto, co-rotinas são colaborativas, sendo executada apenas uma por vez.

A documentação de NCLua disponível em [**?**], apresenta um módulo que utiliza co-rotinas para facilitar o tratamento das requisições assíncronas da classe de eventos *tcp*. No entanto, mesmo tal módulo ainda não permite encapsular todos os detalhes do envio da requisição em apenas uma função, para simplificar o uso para os desenvolvedores de aplicações interativas.

A Figura 1 apresenta um gráfico de máquinas de estados da realização de uma conexão TCP no Ginga-NCL, utilizando o módulo *tcp.lua*. Em um processo convencional, a aplicação estabelece uma conexão a um servidor. Após estabelecida a conexão ela envia uma requisição e fica aguardando o retorno. A função *tcp.receive* é executada até que não hajam mais dados a serem recebidos, realizando a desconexão.

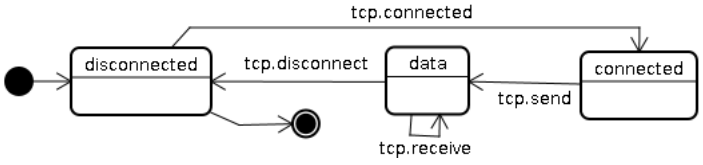

Figura 1. Diagrama de Máquinas de Estados do Módulo *tcp.lua*

A Figura 2 apresenta um gráfico de máquinas de estados do mesmo módulo, porém, do ponto de vista das corotinas em execução (que são semelhantes a *threads*, como já discutido anteriormente). O processo de uso do módulo é iniciado internamente com a criação de uma corotina (que é criada suspensa, não automaticamente iniciando sua

execução). A função *coroutine.resume* inicia a execução da co-rotina. Quando é solicitada uma tentativa de conexão, a co-rotina é suspensa, ficando aguardando até que a conexão seja estabelecida, ocorrendo tudo de forma assíncrona. Após a conexão, a co-rotina é novamente resumida (continuando a execução), permitindo que seja enviada uma requisição ao servidor (*tcp.send*). Tal função retorna imediatamente. Para a obtenção da resposta da requisição (que também será feita de forma assíncrona), é preciso usar a função *tcp.receive*, que faz com que a co-rotina seja suspensa novamente, até que algum dado da resposta seja obtido. A função *tcp.receive* pode ser chamada iterativamente até que não haja mais nenhum dado a ser retornado para a aplicação.

Todo este processo apresentado nos gráficos de máquinas de estado, mesmo utilizando as facilidades providas pelo módulo *tcp.lua*, não são encapsulados para facilitar o uso. O módulo citado apenas facilita o gerenciamento das chamadas assíncronas. A implementação do NCLua HTTP e NCLua SOAP encapsulam todas essas chamadas de funções, tornando o processo bem mais simples para o desenvolvedor, disponibilizando uma simples função para realização de uma requisição.

É importante lembrar que tal abordagem é útil em cenários de aplicações *stateless*, que não mantém estado entre uma requisição e outra, como o caso do HTTP/1.0 e das chamadas SOAP. Aplicações que precisem manter a conexão aberta, como mensageiros instantâneos, precisam utilizar diretamente as funções do módulo tcp.lua.

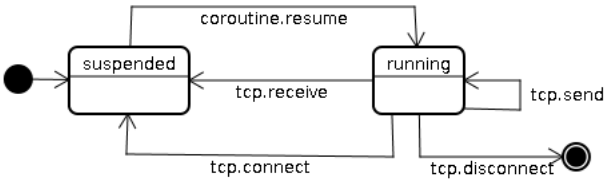

Figura 2. Diagrama de Máquinas de Estados do Módulo *tcp.lua* (corotinas em execução)

3.4. Implementações de SOAP

Existem diversos *toolkits* para provimento e consumo de *Web Services*, disponíveis em diferentes linguagens e plataformas. Nas sub-seções seguintes são apresentados alguns destes.

3.4.1. LuaSOAP

Para acesso a *Web Services* SOAP a partir de aplicações Lua, pode-se utilizar o módulo LuaSOAP [**?**].

O protocolo SOAP envolve a troca de mensagens em formato XML, permitindo a interoperabilidade entre sistemas desenvolvidos em diferentes linguagens. No entanto, para fazer o *parse* de arquivos XML, o LuaSOAP depende da biblioteca *Expat* [**?**][**?**], que é um *parser* XML desenvolvido em linguagem C, cujos problemas de uso em aplicações de TVDi foram apresentados na Seção 2. O projeto também depende da biblioteca *LuaSocket*, que também usa módulos em C.

O LuaSOAP não permite a geração automática de *stubs* para realizar a chamada

aos métodos remotos do *Web Service*. Desta forma, o desenvolvedor precisa ler o documento WSDL e obter as informações sobre o método que deseja invocar. A última atualização do projeto foi em 2004, o que mostra que o mesmo não está acompanhando as novas versões de Lua, como a 5.1 utilizada na implementação de referência do Ginga.

Uma das vantagens do projeto é que as chamadas aos métodos remotos no *Web Service* são síncronas, o que facilita bastante o uso.

3.4.2. gSOAP

O gSOAP é classificado como um *Software Development Kit* (SDK) para permitir que aplicações legadas, sistemas embarcados e de tempo real, desenvolvidos em linguagens C, C++ e Fortran, possam consumir *Web Services* [**?**]. O projeto possui um utilitário capaz de gerar *stubs* em C/C++, a partir do documento WSDL do serviço. Em tal *stub* são incluídos métodos *proxies* para realizar chamadas aos métodos remotos do serviço, tornando-as transparentes para a aplicação cliente.

As chamadas realizadas com gSOAP também são síncronas, o que facilita muito o desenvolvimento das aplicações clientes, pois o envio da requisição e tratamento da resposta pode ser todo feito em uma única rotina.

Pelo fato do projeto ser desenvolvido em C, o mesmo só poderia ser utilizado em aplicações de TVD residentes no conversor digital, como já comentado na Seção 2. A utilização de linguagens compiladas como C/C++ permite que o *(un)marshalling* seja feito em tempo de compilação, o que garante que a aplicação terá maior velocidade na execução. No entanto, a necessidade de tal processo de compilação elimina a grande vantagem do dinamismo existente em linguagens interpretadas como Lua.

3.4.3. Apache Axis

Em [**?**] são apresentados alguns outros *toolkits* para provimento e consumo de *Web Services*. Um dos projetos citados é o Apache Axis, uma implementação de SOAP para Java e C, que tem como uma das vantagem, o uso do *parser* XML SAX, o qual é completo e bastante eficiente. Ele também possui a vantagem de gerar *stubs* Java ou C, a partir do documento WSDL.

Mesmo o Ginga permitindo o uso de Java, o Apache Axis pode não ser uma solução ideal para aplicações de TVD, uma vez que inclui o SAX como *parser* XML. Considerando a capacidade de hardware restrita dos conversores de TV Digital, tal *parser* pode não ser ideal em tais equipamentos, além de poder não estar em conformidade com as normas do Sistema Brasileiro de TV Digital (SBTVD). *Parsers* mais simples como o NanoXML[1], que demandam menos capacidade de processamento, podem ser mais adequados neste cenário. Por fim, o Apache Axis não atende a um dos requisitos desejados: ser implementado em Lua para uso direto por aplicações NCLua.

[1] http://nanoxml.sourceforge.net

4. Proposta

O Ginga-NCL provê protocolos de rede até o TCP, assim, qualquer protocolo acima da camada de transporte do modelo OSI precisa ser implementado. Como o envio de envelopes SOAP é normalmente feito por HTTP, tal protocolo precisou ser implementado como um módulo NCLua, ao qual denominou-se NCLua HTTP, que será utilizado pelo NCLua SOAP.

4.1. NCLua HTTP

O NCLua HTTP[2] implementa alguns dos principais recursos do protocolo HTTP/1.0. Ele é um módulo escrito inteiramente em linguagem Lua para ser utilizado em *scripts* NCLua. O mesmo utiliza o protocolo TCP da forma como especificado na norma do Ginga-NCL em [**?**], por meio da classe de eventos *tcp* de NCLua. Pela simplicidade do protocolo HTTP, o módulo possui apenas algumas funções que permitem a geração de requisições e tratamento de respostas. Atualmente existem os seguintes recursos implementados:

- suporte à autenticação básica, uso de portas específicas e download de arquivos;
- suporte à requisições *GET* e *POST*;
- passagem de parâmetros em requisições *POST*;
- suporte à passagem de cabeçalhos HTTP e definição de *User-Agent*;
- suporte à separação automática dos dados do cabeçalho e do corpo da resposta de uma requisição.

4.2. NCLua SOAP

O NCLua SOAP[3] implementa as principais funcionalidades do protocolo SOAP 1.1 e 1.2. Ele também é um módulo inteiramente escrito em Lua, que faz o *parse* de arquivos XML, realizando o *marshalling* e *unmarshalling* de/para tabelas Lua, permitindo que o desenvolvedor Lua trabalhe com a estrutura de dados principal da linguagem: o tipo *table*.

O módulo utiliza o NCLua HTTP para transportar as mensagens SOAP. Assim, todos os detalhes do protocolo HTTP são encapsulados pelo respectivo módulo. Com isto, a implementação do SOAP fica bastante simplificada, tornando o código fácil de ser mantido. O NCLua SOAP encarrega-se apenas de gerar o XML da requisição SOAP, utilizando o NCLua HTTP para enviar tal XML no corpo da mensagem. O *parse* e *unmarshalling* do XML para uma tabela Lua é todo encapsulado pelo módulo LuaXML[4].

Um importante recurso, não disponível em implementações como o LuaSOAP (citada na Seção 3), e que facilita bastante a utilização do módulo, é a simplificação do XML retornando como resposta, que é convertido (*unmarshalling*) automaticamente para uma tabela Lua. Para demonstrar este recurso, utilizar-se-á o *Web Service* de consulta de endereço a partir de um CEP, disponível em http://www.bronzebusiness.com.br/webservices/wscep.asmx. Tal WS possui um método chamado "cep", que recebe um determinado CEP e retorna o endereço referente ao mesmo. O XML do retorno do método "cep", convertido para uma tabela Lua, é semelhante ao mostrado na Listagem 1.

[2] http://ncluahttp.manoelcampos.com

[3] http://ncluasoap.manoelcampos.com

[4] *Parser* XML escrito inteiramente em Lua, adaptado para funcionar com Lua 5

A estrutura da tabela reflete o código XML retornado. Como pode ser visto, o elemento da tabela que contém de fato os dados do endereço retornado (tbCEP) está envolvido em várias outras tabelas que não contém dado algum, sendo estruturas completamente desnecessárias para a aplicação NCLua. Com isto, para o desenvolvedor poder acessar, por exemplo, a cidade do CEP indicado, precisará conhecer toda a estrutura retornada, utilizando uma instrução como *result.cepResult.diffgr.NewDataSet.tbCEP.cidade*.

Para esconder estes detalhes do desenvolvedor, o NCLua SOAP simplifica qualquer resultado que contenha estruturas desnecessárias, como o mostrado na Listagem 1.

Listagem 1. Exemplo de tabela Lua gerada a partir de um XML de resposta a uma requisição SOAP

```
1 { cepResult = {  diffgr = { NewDataSet = {  tbCEP = {
2    nome="Cln 407", bairro="Asa Norte", UF="DF", cidade="Brasilia"
3 } } } } }
```

Desta forma, para o exemplo citado, a tabela Lua (gerada a partir do XML de retorno da requisição) ficará como apresentado na Listagem 2, o que simplifica o acesso aos elementos da estrutura retornada, permitindo, por exemplo, que o campo cidade seja acessado utilizando-se apenas a instrução *result.cidade*.

Listagem 2. Exemplo de simplificação de retorno de resposta a uma requisição SOAP feita pelo NCLua SOAP

```
1 {    nome="Cln 407", bairro="Asa Norte", UF="DF", cidade="Brasilia"    }
```

O módulo ainda conta com um *script* (*wsdlparser.lua*), em fase inicial de implementação, que realiza o *parse* de um documento WSDL e obtém algumas das informações que precisa-se passar ao NCLua SOAP para que ele realize a requisição (como o *namespace* do serviço, o nome do método desejado e a lista de parâmetros de entrada). Atualmente o *script* apenas lê o WSDL e exibe algumas das informações citadas, cabendo ao desenvolvedor copiá-las e passá-las ao método *call* do NCLua SOAP para realizar a chamada a um determinado método remoto. No entanto, a extração de tais informações já ajuda de alguma forma, principalmente os usuários menos experientes na tecnologia de *Web Services* e no NCLua SOAP.

Para resumir, as características principais do NCLua SOAP são:

- suporte a SOAP 1.1 e 1.2;
- suporte a parâmetros de entrada e saída do tipo *struct* e *array* (sendo feito *marshalling* e *unmarshalling* de/para tabelas Lua automaticamente);
- facilidade para manipulação de chamadas assíncronas, característica do protocolo TCP disponível no Ginga-NCL;
- simplicidade na obtenção do retorno de uma requisição a um método remoto;
- suporte a SOAP *Fault* para captura de erros SOAP;
- suporte a SOAP *Header*[**?**] possibilitando a passagem de parâmetros específicos da aplicação (como informações sobre autenticação, pagamento, etc);
- realização de testes com *Web Services* desenvolvidos em diferentes linguagens.

A Figura 3 apresenta um diagrama de componentes dos módulos implementados. O módulo *event* faz parte do Ginga-NCL e é responsável por tratar eventos, como requisições e obtenção de respostas por meio do protocolo TCP. Ele é a base para toda a

implementação. O módulo *tcp.lua* facilita o gerenciamento das requisições TCP assíncronas, geradas por meio do módulo *event*. O módulo *ncluahttp.lua* implementa o protocolo HTTP, utilizando o TCP como camada de transporte. O módulo *ncluasoap.lua* implementa o protocolo SOAP, utilizando o *ncluahttp.lua* para enviar os envelopes SOAP por HTTP. Uma aplicação cliente, que queira utilizar o protocolo HTTP (*NCLua HTTP Client App*), pode fazer uso direto das funções do módulo *ncluahttp.lua*, abstraindo todos os outros módulos. Uma aplicação cliente, que queira consumir *Web Services* SOAP (*NCLua SOAP Client App*), pode fazer uso direto das funções do módulo ncluasoap.lua, também abstraindo todos os outros módulos.

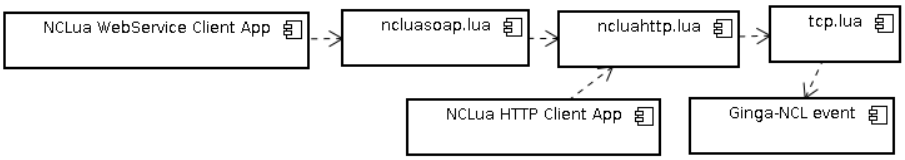

Figura 3. Diagrama de Componentes do NCLua SOAP e NCLua HTTP

5. Resultados

Nesta seção são apresentados os resultados obtidos com a implementação dos módulos NCLua HTTP e NCLua SOAP, assim como exemplos de utilização e aplicações desenvolvidas utilizando tais módulos.

5.1. Exemplos de uso do NCLua HTTP

A utilização básica do módulo NCLua HTTP é por meio de uma chamada *GET* a uma determinada URI, utilizando o método *request* do módulo, como exemplificado no *script* NCLua da Listagem 3. Tal exemplo envia uma requisição HTTP *GET* para a página em http://manoelcampos.com/votacao/votacao2.php, enviando um parâmetro "voto"com valor igual a "sim", obtendo o resultado (um código HTML neste caso) e exibindo no terminal. Não existe interface gráfica neste exemplo, desta forma, o *script* contém apenas detalhes da requisição HTTP, mas é inteiramente funcional.

Na Listagem 3, a linha 1 adiciona o módulo NCLua HTTP. A linha 8 chama a função *http.request* que envia a requisição HTTP. Devido à particularidade assíncrona do protocolo TCP no Ginga-NCL (que é utilizado para transportar as mensagens HTTP), como explicado na Seção 3.3, para facilitar o envio de requisições e recebimento de respostas, o módulo NCLua HTTP utiliza co-rotinas de Lua. Por este motivo, a obtenção do retorno deve ser feita dentro de uma função, como a definida na linha 6, contendo os parâmetros *header* e *body* (comentados na definição da mesma). Tal função deve ser passada como parâmetro à função *http.request*, para que seja chamada por esta quando a resposta for obtida.

Listagem 3. Exemplo de envio de requisição GET (contendo parâmetros) com NCLua HTTP

```
1 require "http"
2 --Funcao de callback executada de forma assincrona pelo modulo,
3 --quando a resposta da requisicao eh obtida.
4 --@param header Headers HTTP enviados na resposta
5 --@param body Corpo da mensagem HTTP
6 function getResponse(header, body) print("Resposta obtida\n", body) end
```

```
7
8 http.request("http://manoelcampos.com/voto.php?voto=sim", getResponse)
```

O envio de requisições *POST* é bastante semelhante ao exemplo apresentado anteriormente. Neste caso, os parâmetros devem ser passados à função *http.request* por meio de uma tabela Lua. A formatação destes valores, como exigido pelo protocolo HTTP, é feita automaticamente pelo NCLua HTTP.

Algumas aplicações foram desenvolvidas como prova de conceito do uso do módulo NCLua HTTP, as quais são apresentadas a seguir:

- **Enquete TVD**: enquete com registro de voto e apuração a partir de servidor *Web*;
- **NCLua RSS Reader**: leitor de notícias RSS de um provedor de conteúdo na *Web*;
- **NCLua Tweet**: envio e recebimento de mensagens pelo micro *blog Twitter*.

5.2. Exemplos de uso do NCLua SOAP

A seguir são apresentados exemplos de uso do NCLua SOAP para consumo de alguns *Web Services* disponíveis na *Internet*.

A Listagem 4 apresenta um exemplo de aplicação para exibir a previsão do tempo de uma cidade. A mesma não possui interface gráfica, mostrando o resultado no terminal, para simplificar o código. A linha 10 realiza a chamada ao método remoto. Os dados para realização da requisição (incluindo o endereço do serviço, nome do método remoto e parâmetros de entrada) devem ser informados em uma tabela Lua, como mostrado entre as linhas 4 a 8. A obtenção do retorno do método remoto não é direta, devido à característica assíncrona do protocolo TCP no Ginga-NCL, como já explanado nas Seções 3.3 e 4.1. Desta forma, é necessária a definição de uma função que deve receber apenas um parâmetro (normalmente nomeado de *result*), como a função *getResponse* exibida na linha 2. Tal função deve ser passada como parâmetro ao método *call* do módulo ncluasoap, como mostrado na linha 10. O envio da requisição será executado dentro de uma co-rotina Lua, criada pelo NCLua SOAP. A função *getResponse* será executada automaticamente quando o retorno for obtido. Desta forma, todo o controle das chamadas assíncronas é feito pelo NCLua SOAP, como já amplamente discutido na Seção 4.1.

Como o serviço consumido neste exemplo retorna apenas uma *string* com a previsão do tempo (chuvoso, nublado, ensolarado, etc), para exibir tal resultado basta imprimir o parâmetro *result* da função *getResponse*, como apresentado na linha 6.

Listagem 4. Exemplo de consumo de *Web Service* de previsão do tempo

```
1  require "ncluasoap"
2  function getResponse(result) print("Previsao do Tempo: ", result) end
3
4  local msg = {
5    address = "http://www.deeptraining.com/webservices/weather.asmx",
6    namespace = "http://litwinconsulting.com/webservices/",
7    operationName = "GetWeather", params = {  City = "New York"  }
8  }
9
10 ncluasoap.call(msg, getResponse)
```

O exemplo apresentado na Listagem 5 consume um serviço para obtenção de um endereço a partir do CEP. Observe que o que muda entre as linhas 6 e 10, em relação

ao exemplo anterior, são apenas os valores da tabela "msg", que contém os dados para geração da requisição SOAP. Na função *getResponse*, entre as linhas 2 e 4, note que agora, o resultado retornado é um tipo composto, que é acessado como uma tabela Lua. Tal tabela conterá campos com os valores armazenados no XML enviado pelo *Web Service*.

Listagem 5. Exemplo de consumo de WS de consulta de endereço a partir do CEP

```
1  require "ncluasoap"
2  function getResponse(result)
3     print(result.nome, result.bairro, result.cidade, result.UF)
4  end
5
6  local msg = {
7    address = "http://www.bronzebusiness.com.br/webservices/wscep.asmx",
8    namespace = "http://tempuri.org/",
9    operationName = "cep", params = {  strcep = "70855530"  }
10 }
11
12 ncluasoap.call(msg, getResponse)
```

Algumas aplicações foram desenvolvidas como prova de conceito de utilização do módulo. Entre elas estão o rastreamento de encomendas postadas pelos Correios, consulta de cotação do Dólar, previsão do tempo, consulta de endereço a partir do CEP e outras.

5.2.1. Testes de Interoperabilidade

Os diversos serviços consumidos pelas aplicações apresentadas na Seção 5.2 são desenvolvidos em diferentes linguagens e plataformas. Procurou-se realizar testes com estes diferentes serviços para verificar a interoperabilidade do NCLua SOAP (um dos requisitos fundamentais, se não o mais importante, para qualquer implementação SOAP). Com os testes realizados pôde-se comprovar a eficiência do módulo, uma vez que para todos os serviços testados, obteve-se o retorno esperado, tratando-o de forma padronizada.

No início, alguns usuários do fórum do projeto relataram problemas ao utilizar serviços desenvolvidos em linguagem Java. Os problemas encontrados foram devido aos *Web Services* criados com a API JAX-WS, padrão da plataforma Java, especificarem os tipos de dados utilizados pelo serviço em um arquivo XSD (*XML Schema Definition*) externo, no lugar de especificar diretamente dentro do documento WSDL. Tal característica requer pequenas mudanças no formato da mensagem SOAP a ser enviada para consumir tais *Web Services*. Assim, foi preciso adequar o módulo para entrar em conformidade com o padrão SOAP.

Particularidades encontradas em *Web Services* PHP também foram relatadas e o módulo foi adequado para permitir a interoperabilidade com tais serviços. É importante ressaltar que, no caso de tais serviços PHP, estes foram desenvolvidos utilizando o *tookit* nuSOAP[5]. Tal *tookit* não leva em conta os nomes dos parâmetros de entrada e sim a ordem dos mesmos, não estando em conformidade com o padrão SOAP. Assim, o problema é específico da implementação do nuSOAP, mas que foi contornado pelo NCLua SOAP para evitar quaisquer transtornos ao consumir *Web Services* desenvolvidos com tal *tookit*.

[5]http://sourceforge.net/projects/nusoap

5.3. Comparativo entre os módulos implementados e o uso direto do módulo tcp.lua

Nesta seção é feito um comparativo entre aplicações desenvolvidas utilizando os módulos implementados e o uso direto do módulo *tcp.lua*[**?**].

O NCLua SOAP, juntamente como o NCLua HTTP, encapsulam toda a complexidade de envio de requisições SOAP por meio do protocolo HTTP, facilitando a utilização de tais protocolos em aplicações de TVDi. A quantidade de código necessária para a realização de requisições HTTP ou SOAP é bastante reduzida com os módulos implementados, permitindo uma maior agilidade no desenvolvimento de aplicações, além de minimizar a possibilidade de erros e reduzir a duplicação de código.

A Tabela 1 apresenta um comparativo do total de linhas existentes em aplicações utilizando os módulos desenvolvidos e outras que não os utilizam, mostrando como o código das aplicações é reduzido com o uso dos módulos implementados. A utilização dos módulos no desenvolvimento de aplicações, além de reduzir o código das mesmas, libera o desenvolvedor da necessidade de conhecer detalhes de implementação dos protocolos HTTP e SOAP. Sem a utilização do módulo *tcp.lua*, com o uso direto da classe de eventos *tcp* do Ginga-NCL, o código das aplicações aumentaria substancialmente, em torno de 100 linhas de código.

Aplicação/Protocolo	**Sem módulos implementados**	**Com módulos implementados**
Enquete/HTTP	14 linhas de código 5 funções utilizadas diretamente	5 linhas de código = 35% do anterior 1 função usada diretamente
Cotação do dólar/SOAP	64 linhas de código 5 funções utilizadas diretamente	15 linhas de código = 23% do anterior (9 são parâmetros do WS) 1 função usada diretamente

Tabela 1. Comparativo entre aplicações de TVD com e sem os módulos implementados

6. Decisões de Projeto

Para o desenvolvimento do projeto, optou-se por seguir o padrão de módulos, amplamente utilizado na atual versão da linguagem Lua. Um módulo encapsula funções com objetivos correlatos e tal padrão é bastante conhecido dos programadores Lua.

Tal modelo segue o paradigma de programação estruturada, assim, um módulo consiste de um conjunto de funções que são chamadas pelo desenvolvedor que for utilizá-lo.

Na geração das requisições, optou-se por fazer o *(un)marshalling* de XML para tabelas Lua por estas serem as estruturas de dados padrões disponibilizadas pela linguagem e por serem estruturas bastante flexíveis e fáceis de manipular, devendo ser de conhecimento de qualquer programador Lua.

Como o conteúdo das requisições HTTP e SOAP é apenas texto, formatado segundo o padrão de cada protocolo, a geração e formatação deste conteúdo foi bastante simples e direta. Devido a *strings* em Lua serem imutáveis[**?**], para economizar memória na concatenação das *strings* que compõe cada requisição, as tabelas de Lua foram utilizadas como um *buffer* de *strings* para otimizar o uso de memória RAM.

Na obtenção dos resultados, procurou-se facilitar ao máximo tal tarefa para o desenvolvedor, permitindo que ele tenha acesso direto aos dados retornados, sem precisar conhecer o caminho dentro do XML de retorno onde a resposta está armazenada, assim como fazem outros *toolkits* SOAP como a API JAX-WS.

Quanto ao *parser* XML escolhido, não haviam muitas opções implementadas inteiramente em Lua. Todas as implementações testadas foram encontradas em http://lua-users.org e em [**?**]. Optou-se pelo uso do LuaXML[6], pois foi a implementação que fazia o *unmarshalling* de XML para tabelas Lua com a estrutura mais simples e fácil de manipular. Além do mais, as outras implementações encontradas não funcionaram adequadamente para XML's mais complexos retornados por alguns *Web Services*.

7. Ambiente de desenvolvimento

Para o desenvolvimento do projeto foi utilizado:

- sistema operacional GNU/Linux Ubuntu 10.10 como sistema desktop para realização das tarefas de desenvolvimento:
  - ferramentas *telnet* e *curl* para teste de envio de requisições HTTP/SOAP geradas com os módulos implementados.
- *soapUI* 3.5.1 para testes de consumo de *Web Services* SOAP;
- *Astah Community* 6.1 (antigo *Jude*) para modelagem UML;
- IDE Eclipse 3.6:
  - *plugin* NCLEclipse 1.5.1;
  - *plugin* LuaEclipse 1.3.1.
- ferramenta LuaDoc[7] para geração de documentação;
- implementação de referência do Ginga-NCL (Ginga *Virtual Set-top Box* 0.11.2):
  - módulo *tcp.lua*[8] para envio de requisições TCP;
  - módulo LuaXML para tratamento de XML;
  - módulo *LuaProfiler*[9] para realização das análises de desempenho, descritas na Seção 8.
- interpretador Lua para execução do *script wsdlparser.lua* fora do Ginga *Virtual Set-top Box*.

8. Análises de Desempenho do NCLua SOAP

Utilizando-se a ferramenta *LuaProfiler* (que precisou ter seu processo de compilação alterado para funcionar em aplicações NCLua[10]) foram feitas algumas análises do desempenho do NCLua SOAP para avaliar o tempo gasto para geração da requisição SOAP. Por meio da função *collectgarbage* da biblioteca padrão de Lua, pôde-se verificar o total de memória consumida pelo módulo.

O percentual de uso da CPU foi obtido por meio da ferramenta *top*, existente no sistema operacional do Ginga *Virtual Set-top Box*. Para a obtenção deste percentual, executou-se a aplicação NCL (que não inicia o *script* Lua automaticamente) e verificou-se o uso da CPU pelo processo de nome "ginga"(responsável pela execução das aplicações interativas) no momento em que este se tornava estável. Em seguida, por meio de um comando na aplicação NCL, o *script* Lua é então executado, gerando e enviando a requisição SOAP. Após isto, verificou-se o pico de uso do processador, considerando que os

[6]http://lua-users.org/wiki/LuaXml
[7]http://luadoc.luaforge.net
[8]http://www.lua.inf.puc-rio.br/ francisco/nclua/
[9]http://luaprofiler.luaforge.net
[10]http://goo.gl/42CQA

scripts Lua testados somente enviam a requisição SOAP e exibem o retorno no terminal. Com a diferença entre o pico de uso do processador depois da execução do *script* e o valor observado antes do início do mesmo, obtém-se o percentual de uso do processador pelo NCLua SOAP.

A Tabela 2 apresenta os resultados obtidos para algumas das aplicações testadas.

***Web Service* consumido**	**Parâmetros de entrada**	**Tempo de geração da requisição: chamada ao método *call* (em segundos)**	**Uso de Memória RAM (em KB)**	**% de Uso da CPU**
Situação do tempo http://www.deeptraining.com /webservices/weather.asmx	*City* = "Brasília"	0.13	121.78	0.3
Conversão de Moeda http://www.webservicex.net /CurrencyConvertor.asmx	*FromCurrency* = "USD" *ToCurrency* = "BRL"	0.13	121.65	0.3
Consulta de endereço a partir do CEP http://www.maniezo.com.br /webservice/soap-server.php	cep = "77021682"	0.13	133.17	0.3

Tabela 2. Análise de Desempenho do NCLua SOAP

Como pode ser observado na Tabela 2, o tempo para geração da requisição, ou seja, a chamada ao método *call* do módulo, na implementação de referência do Ginga (Ginga *Virtual Set-top Box*, versão 0.11.2) está em torno de 0.13 segundo. O método *call* inclui a conversão dos dados passados em uma tabela Lua para o formato XML para serem enviados ao *Web Service*. Tal método não aguarda a conexão ao servidor *Web* e nem o envio e resposta da requisição, pois tais operações são assíncronas no Ginga-NCL. Assim, o tempo de execução do método *call* reflete apenas o tempo de geração da requisição SOAP. Os tempos obtidos nos testes não variaram muito, e a variação vai depender apenas do total de parâmetros passados.

O uso de memória RAM também ficou baixo, em torno de 120KB, que varia de acordo com os parâmetros de entrada e saída do método remoto.

Em todos os exemplos executados, pôde-se observar que o percentual de uso da CPU se manteve estável em 0.3%, sendo um valor consideravelmente baixo.

Foram feitas análises do uso direto do módulo *tcp.lua* para verificar o *overhead* causado pelo NCLua SOAP, que é uma camada sobre o *tcp.lua*. Nos três exemplos apresentados na Tabela 2, o tempo de uso da CPU se manteve também estável e igual aos valores obtidos com o NCLua SOAP, 0.3%. Não foram feitas medidas quanto ao tempo gasto (em segundos) para geração da requisição, pois, para uso direto do *tcp.lua*, o desenvolvedor precisa passar ao módulo uma *string* já com a requisição SOAP formatada. Assim, não há um *delay* para geração da requisição pois o *tcp.lua* recebe a mesma pronta para envio (o que é bastante complicado para o desenvolvedor gerar manualmente tal requisição HTTP/SOAP, além de ser totalmente inflexível).

Quanto ao consumo de memória RAM, o uso direto do *tcp.lua* permite uma otimização neste sentido, uma vez que a quantidade de varáveis e estruturas de dados declaradas é muito menor, considerando que o módulo deve receber a requisição pré-formatada. Para o primeiro exemplo da Tabela 2, a aplicação consumiu 76.02 KB de memória RAM, para o segundo consumiu 75.59 KB e para o terceiro exemplo consumiu 66.28 KB. Tais valores mostram que o consumo de memória está em torno de 50% do consumido pelo NCLua SOAP.

Foram feitas aplicações para consumir os mesmos *Web Services* apresentados na Tabela 2 utilizando o módulo LuaSOAP. No entanto, a aplicação funcionou apenas para o *Web Service* de obtenção de endereço a partir do CEP, que implementa o SOAP 1.1. Para os outros *Web Services* testados, que implementam SOAP 1.2, o LuaSOAP não enviou a requisição no formato esperado e os *Web Services* retornaram mensagens de erro. O LuaSOAP é um projeto que não tem atualizações desde 2004, assim, tais problemas eram esperados. Com o *Web Service* de busca de endereço, a aplicação consumiu 139.20KB de memória RAM, cerca de 6KB a mais que com o NCLua SOAP. O tempo de geração da requisição foi de apenas 0.01 segundo, bem inferior ao do NCLua SOAP, que levou 0.13 segundo. Presume-se que este tempo reduzido gasto pelo LuaSOAP é devido ao mesmo utilizar módulos escritos em C, que por serem compilados, têm este ganho de desempenho.

Quanto ao uso de CPU, a aplicação com o LuaSOAP teve pico de 1.0%, sendo que com o NCLua SOAP este percentual foi de apenas 0.3. Presume-se que tal valor elevado é devido ao fato de uso do *parser* XML *Expat*, que é uma implementação mais completa que o LuaXML usado no NCLua SOAP.

Com tais valores apresentados anteriormente, considera-se que o NCLua SOAP é bastante eficiente em questões de tempo de processamento e uso de memória. Presume-se que o tempo de processamento pode ser reduzido em um hardware dedicado como o dos *Set-top Boxes*, pois os testes foram realizados em uma sistema operacional Linux com várias aplicações em execução e em uma máquina virtual (cujo desempenho é inferior a de uma máquina real com finalidades específicas).

9. Limitações do módulo NCLua SOAP

A atual versão do NCLua SOAP possui algumas limitações, que entendemos não interferirem no uso do mesmo:

- necessidade de o desenvolvedor saber a versão do protocolo SOAP do serviço que ele deseja consumir;
- necessidade de extrair manualmente alguns dados como o *namespace* do serviço;
- falta de tratamento de erros retornados pelo protocolo HTTP, o que causará comportamentos inesperados na aplicação;
- necessidade de saber se o serviço utiliza um arquivo *XML Schema Definition* (XSD) externo, para poder indicar isto no momento da chamada ao método remoto;
- não permitir o envio de anexos em formato *Multipurpose Internet Mail Extensions* (MIME)[**?**] dentro de uma mensagem SOAP.

10. Comparação do NCLua SOAP com outras implementações

A Tabela 3 apresenta um comparativo entre alguns *toolkits* SOAP e o NCLua SOAP.

Características	**Axis**	**Axis2**	**PHP**	**gSOAP**	**NCLua SOAP**
Linguagem	Java	Java	PHP 5	C++	NCLua
SOAP 1.1	Sim	Sim	Sim	Sim	Sim
SOAP 1.2	Sim	Sim	Sim	Sim	Sim
SOAP com Anexos	Sim	Sim	Sim	Sim	Ainda não
Geração de código cliente a partir do WSDL	Sim	Sim	Sim	Sim	Ainda não
Suporte para formato *document/literal*	Bom	Bom	Médio	Bom	Bom
Requisitos de *runtime*	JVM	JVM	PHP engine	Nenhum	Ginga-NCL
Documentação	Boa	Pequena	Média	Boa	Média

Tabela 3. Comparação entre NCLua SOAP e outros *toolkits* SOAP. Adaptada de [?]

11. Conclusão

Os módulos NCLua HTTP e NCLua SOAP facilitam a convergência entre *Web* e TV, escondendo os detalhes de implementação dos protocolos HTTP e SOAP do desenvolvedor de aplicações de TVDi, permitindo o surgimento de novas aplicações interativas que fazem uso de conteúdo da *Internet*. O NCLua SOAP permitiu o desenvolvimento de aplicações de *T-Government* como apresentado em [**?**], sistema de recomendação[**?**] entre outras. Na página do projeto, em http://ncluasoap.manoelcampos.com, existe um *link* para o fórum de discussão do módulo, onde alguns usuários relatam a utilização do mesmo, por exemplo, em aplicações de *T-Learning*.

Atualmente, o processo de obtenção dos dados para realizar a chamada a um método remoto em um *Web Service* ainda é praticamente todo manual, no entanto, após o desenvolvedor obter tais dados para o método remoto desejado, a realização da requisição é bastante simplificada, principalmente pelo fato de Lua ser uma linguagem de tipagem dinâmica [**?**], não obrigando a declaração de variáveis com seus respectivos tipos. O *feedback* dos usuários tem mostrado que o módulo está sendo bastante útil para a comunidade, além de permitir a evolução do mesmo.

12. Trabalhos Futuros

Como trabalhos futuros, pretende-se concluir a implementação do *parse* automático do documento WSDL e geração de *stubs* Lua, contendo funções *proxies* para realizar a chamada aos métodos remotos (semelhante a ferramentas como o *wsdl2java*[11], pretende-se transformar o *script wsdlparser* em um *wsdl2nclua*) e incluir tratamento de exceções para permitir que as aplicações de TVDi possam emitir mensagens amigáveis ao usuário quando uma requisição HTTP falhar.

O módulo NCLua HTTP permite que seja utilizado qualquer método HTTP em uma requisição, no entanto, apenas os método *GET* e *POST* foram testados. Desta forma, pretende-se realizar testes de conformidade utilizando-se os métodos HTTP *OPTIONS, HEAD, PUT* e *DELETE*. Pretende-se também implementar mais funcionalidades no módulo, como realizar redirecionamentos automaticamente a partir de respostas HTTP como 301 e 302, além de implementar alguns recursos do HTTP/1.1, como conexões persistentes.

[11] http://ws.apache.org/axis/java/user-guide.html