APLICAÇÕES MÓVEIS MASHUP: UTILIZAÇÃO DE SERVIÇOS *WEB* ATRAVÉS DE UM APLICATIVO MÓVEL DE MENSAGENS

Raphael Soares Moreira[1]
Rogério Nogueira Tostes[2]
Romualdo Monteiro de Resende Costa[3]

RESUMO

Este trabalho apresenta uma proposta de publicação, em dispositivos móveis, de informações obtidas a partir de serviços Web já existentes. Essas informações podem ser combinadas, tornando possível construir novos serviços a partir de serviços já existentes. Nessa proposta, os serviços são acessados através de aplicações Lua, uma linguagem de script projetada para facilitar o desenvolvimento de aplicações. Complementarmente, a especificação da combinação dos serviços é realizada de forma declarativa, utilizando a linguagem NCL (*Nested Context Model*), que é a linguagem padronizada para o Sistema Brasileiro de TV Digital (SBTVD) e padrão ITU-T para IPTV. Nesta proposta não é necessária a instalação de aplicativos específicos no dispositivo móvel, uma vez que as informações são recebidas através de mensagens.

Palavras-chave: Serviços Web. Mensagens. Aplicativo móvel. Lua. NCL.

1 INTRODUÇÃO

No próximo ano, a *World Wide Web (Web)* completa 30 anos. Ao longo de todo esse tempo, os serviços disponíveis para essa plataforma evoluiram de mecanismos para interligação de informações científicas e acadêmicas para vídeo sobre demanda (VoD), redes sociais e comércio eletrônico. Mais

[1] Discente do Curso de Engenharia de Software, do Centro de Ensino Superior de Juiz de Fora. E-mail: <faelsoaresjf@gmail.com>
[2] Discente do Curso de Engenharia de Software, do Centro de Ensino Superior de Juiz de Fora. E-mail: <rogertostes@yahoo.com.br>
[3] Doutor em Informática pela Pontifícia Universidade Católica do Rio de Janeiro. Docente do Curso de Engenharia de Software do Centro de Ensino Superior de Juiz de Fora. E-mail:<romualdocosta@cesjf.br>

recentemente, o sucesso da computação móvel, além de contribuir para a popularização da Web, trouxe também um grande número de novas aplicações que, muitas vezes, oferecem serviços já disponíveis na própria Web.

O desenvolvimento de novas aplicações para dispositivos móveis é, por vezes, necessário, uma vez que as características desses dispositivos, como telas reduzidas e ausência de interfaces como o teclado e o mouse, podem comprometer a usabilidade das aplicações já existentes. No entanto, quando for possível, pode ser interessante reutilizar os serviços já existentes na Web no ambiente da computação móvel, evitando o retrabalho no desenvolvimento de novas aplicações e, até mesmo, permitindo que o usuário não seja obrigado a instalar inúmeros aplicativos no seu dispositivo, muitas vezes um para cada serviço desejado.

Entre os aplicativos disponíveis para os dispositivos móveis, os aplicativos de mensagens instantâneas têm se tornado extremamente populares pois, além de permitirem aos usuários enviar e receber mensagens individualmente, ou em grupos, são uma alternativa prática para enviar e receber serviços (JIN, 2016). Nessa estratégia, conhecida como plataformização, o foco do negócio está na disponibilização de facilidades para acesso a produtos e serviços e não no produto ou serviço, propriamente dito. Através dessa estratégia, serviços como e-mail, jogos, comércio eletrônico, entre outros, podem ser implementados através de um aplicativo de mensagens (M-Apps), sem a necessidade, portanto, da instalação de aplicativos específicos.

Num cenário mais abrangente, uma plataforma, como é o caso dos M-Apps, pode oferecer não apenas um serviço, mas um conjunto de serviços que, agregados, trazem vantagens aos clientes. Essa possibilidade se torna mais provável a medida que os serviços se tornam mais numerosos e, consequentemente, mais especializados. Assim, a plataforma pode oferecer, na verdade, uma composição de serviços, conhecido como *mashup* (GRAMMEL, 2010).

Alguns M-Apps focam quase que exclusivamente no serviço de mensagens. Esse é o caso, por exemplo do Whatsapp[4]. Outros, no entanto, como o Telegram[5], permitem que uma variedade de serviços sejam disponibilizados. Nessa plataforma, robôs podem ser especificados e armazenados para interagir com os usuários, informando ou mesmo solicitando serviços disponíveis na Web.

A especificação dos serviços a serem oferecidos na plataforma requer, inicialmente, a comunicação com os provedores do serviço. Na Web, essa comunicação envolve utilizar os protocolos HTTP (IETF, 2014) e o HTTPS (IETF, 2000), bem como a sintaxe de comandos que, por ventura, o serviço ofereça para facilitar o acesso. Adicionalmente, considerando a possibilidade de que mais de um serviço seja empregado, faz-se necessário especificar os relacionamentos entre os serviços desejados, a fim de definir, por exemplo, a ordem entre eles, as condições para a utilização de mais de um serviço ou outras situações que sejam necessárias ao correto oferecimento dos serviços aos usuários.

Nesse contexto, este artigo apresenta uma proposta, onde aplicativos móveis de mensagens são utilizados como plataforma para o oferecimento de serviços disponíveis na Web, através do acesso de aplicações Lua (Brandão, 2010), uma linguagem amplamente utilizada em aplicações (como os softwares da Adobe[6]), sistemas embarcados (*middlewares* diversos) e, principalmente, jogos (como, por exemplo, *World of Warcraft*[7] e *Andry Birds*[8]).

Nessa proposta, os serviços gerenciados por aplicações Lua são posteriormente relacionados através de documentos NCL (*Nested Context Language*) (ABNT, 2009), para que aplicações *mashup* possam ser construídas. NCL é o padrão ABNT para especificação de aplicações para TV digital (ABNT, 2008), bem como recomendação ITU-T para serviços de IPTV

[4] www.whatsapp.com
[5] www.telegram.org
[6] www.adobe.com
[7] worldofwarcraft.com
[8] www.angrybirds.com

(ITU-T, 2009). Para testes da proposição, uma aplicação demonstrativa é apresentada, utilizando o Telegram como M-App.

O restante deste artigo está organizado da seguinte forma. A próxima seção discute os trabalhos relacionados em relação às tecnologias que oferecem suporte à proposta apresentada. Na terceira seção segue uma apresentação sobre a construção de uma aplicação que explora as peculiaridades da proposta deste artigo. A quarta seção apresenta os resultados alcançados com discussões correlatas e é seguida pela última seção, onde são apresentadas as conclusões finais.

2 REFERENCIAL TEÓRICO

2.1 NCL – *NESTED CONTEXT LANGUAGE*

NCL é uma linguagem declarativa baseada em eventos e são justamente essas características que a habilitam para a especificação de relacionamentos entre serviços. NCL segue uma abordagem modular, onde cada módulo agrupa elementos e atributos XML semanticamente relacionados. Esses módulos, por sua vez, podem ser agrupados em diferentes perfis de linguagem, construídos para atender a uma demanda específica.

Na NCL, os tipos básicos de objetos de mídia são definidos no módulo *Media*. Para a especificação dos objetos, esse módulo define o elemento <*media*>, contendo um conjunto de atributos para, entre outras informações, identificar cada objeto (atributo *id*) e o seu conteúdo (atributo *src*).

Também na NCL, sobre cada objeto de mídia podem ser especificadas âncoras de conteúdo, correspondentes a uma porção das unidades de informação que compõem o conteúdo de um objeto. Essas âncoras são definidas no módulo *MediaContentAnchor*, através do elemento <*area*>. Diferentes propriedades também podem ser especificadas sobre um objeto de

mídia. A especificação de cada propriedade é realizada através do elemento <*property*>, definido no módulo *PropertyAnchor*.

Na verdade, NCL não restringe os tipos dos objetos que podem ser mídias, como imagens, vídeos e áudios, ou mesmo outras aplicações, como é o caso dos objetos Lua. A Figura 1, por exemplo, apresenta três objetos, definidos pelo elemento <*media*>, que contém aplicações Lua, cujo arquivo contendo o código da aplicação é definido através do atributo *src*.

Figura 1. Exemplo de Objetos de Mídia em uma Aplicação NCL.

```
44. ...
45. <body id="corpo">
46.     <port id="entryPoint" component="lua1"/>
47.
49.         <media id="lua1"  src="1.lua">
50.                 <property name="cidade"/>
51.                 <property name="data"/>
52.                 <property name="min"/>
53.                 <property name="max"/>
54.                 <property name="previsaodotempo"/>
55.         </media>
56.         <media id="lua2"  src="2.lua">
57.                 <property name="data"/>
58.                 <property name="min"/>
59.                 <property name="max"/>
60.         </media>

61.         <media id="lua3" src="3.lua">
62.                     <property name="data"/>
63.         </media>
64. ...
```

Fonte: do autor

Na Figura 1, o objeto **lua1** (linhas 49 a 55) possui as propriedades cidade, *data*, *min*, *max* e *previsaodotempo*. Os valores dessas propriedades são acessados pela aplicação Lua e podem, portanto, modificar o

comportamento dessa aplicação. A aplicação Lua pode, inclusive, modificar os valores dessas propriedades, gerando eventos na aplicação NCL. Esses eventos, por sua vez, podem estar relacionados a ações executadas sobre outros objetos, como é o caso dos objetos **lua2** e **lua3** também especificados na Figura 1. A especificação desses relacionamentos, por sua vez, deve ser realizada através do uso de elos (elemento <*link*>), cujo exemplo é apresentado na Figura 2.

Figura 2. Exemplo de Elos em uma Aplicação NCL.

```
60. ...
61.     <link xconnector="onBeginSet_var">
62.         <bind role="onBegin" component="lua1"/>
63.         <bind role="set" component="lua1"
interface="previsaodotempo">
64.                 <bindParam name="var" value="455875"/>
65.         </bind>
66.     </link>
67.
68.     <link xconnector="onEndSet_var">
69.         <bind role="onEnd" component="lua1"/>
70.         <bind role="get" component="lua1" interface="data"/>
71.         <bind role="set" component="lua2" interface="data">
72.                 <bindParam name="var" value="$get"/>
73.         </bind>
74.     </link>
75. ...
```

Fonte: do autor

Nas aplicações NCL, os relacionamentos são especificados através do elemento <*link*>, definido no módulo *Linking*. Esse elemento agrupa elementos <*bind*>, também definidos no módulo *Linking*, que especificam os participantes de uma relação. Na Figura 2, dois relacionamentos são especificados, no primeiro (linhas 61 a 66), o objeto **lua1** é o único participante do

relacionamento. Já no segundo, o relacionamento é especificado entre o objeto **lua1** e o objeto **lua2**.

Embora o relacionamento seja especificado através dos elos, É o conector, definido no módulo *ConnectorBase*, que especifica a semântica da relação de um elo. Por exemplo, em uma relação causal, o conector especifica condições que devem ser satisfeitas para que ações possam ser disparadas. As condições dessas relações podem simples ou compostas, isto é, podem ser formadas por mais de uma condição, mas têm somente uma única condição que deve ser satisfeita em um instante de tempo infinitesimal.

Conectores podem ser especificados na própria aplicação NCL, através do elemento <*causalConnector*>. Cada elemento <*causalConnector*> é especificado como um filho de um elemento <*connectorBase*>, que agrupa (define uma base de) conectores. Nos conectores, condições e ações definem um papel (atributo *role*), que deve ser único na especificação do conector. São esses papéis que serão assumidos pelos participantes de um relacionamento, associados através dos <*binds*> de um elo.

Para facilitar a especificação das aplicações, dois conjuntos de papéis são pré-definidos na NCL. Esses papéis associam transições das máquinas de estado dos eventos, que podem representar tanto condições (primeiro conjunto, descrito na Tabela 1) quanto ações (segundo conjunto, descrito na Tabela 2) em um relacionamento.

Tabela 1. Valores reservados para papéis como condição

Papel (role)	Ação que executa a transição	Evento
onBegin	*start*	*presentation*
onEnd	*stop*	*presentation*
onAbort	*abort*	*presentation*
onPause	*pause*	*presentation*
onResume	*resume*	*presentation*
onSelection	*start*	*selection*

onBeginAttribution	*start*	*attribution*
onEndAttribution	*start*	*attribution*

Fonte: do autor

Tabela 2. Valores reservados para papéis como ação

Papel (role)	Ação que executa a transição	Evento
start	*start*	*presentation*
stop	*stop*	*presentation*
abort	*abort*	*presentation*
pause	*pause*	*presentation*
resume	*resume*	*presentation*
set	*start*	*attribution*

Fonte: do autor

Voltando ao exemplo da Figura 2, o primeiro elo (linhas 61 a 66) tem como condição que o evento de apresentação sobre um objeto seja iniciado ("*onBegin*") para que a atribuição ("set") de um valor a um propriedade desse mesmo objeto também seja iniciada. O outro elo tem como condição que o evento de apresentação sobre um objeto seja encerrado ("*onEnd*") para que a atribuição de um valor a uma propriedade de outro objeto, também seja iniciada. Os elos apresentados na Figura 2 utilizam os conectores definidos na Figura 3.

Figura 3. Exemplo de Conectores em uma Aplicação NCL.

```
...
    <causalConnector id="onBeginSet_var">
        <connectorParam name="var"/>
        <simpleCondition role="onBegin"/>
        <simpleAction role="set" value="$var"/>
     </causalConnector>

    <causalConnector id="onEndSet_var">
        <connectorParam name="var"/>
        <simpleCondition role="onEnd"/>
        <simpleAction role="set" value="$var"/>
    </causalConnector>

...
```

Fonte: do autor

2.2 LUA

Lua é considerada uma linguagem de fácil aprendizado, que combina sintaxe procedural com declarativa. Desde o início do seu desenvolvimento, essa linguagem foi projetada para ser utilizada juntamente com outras linguagens. O próprio nome da linguagem remete ao fato da linguagem ser um satélite a outra. Assim, é esperado que a aplicação Lua esteja junto a outra aplicação, no caso deste trabalho, a NCL.

Para comunicação com aplicações NCL, a linguagem LUA foi estendida com novas funcionalidades (Brandão, 2010). Entre as funcionalidades adicionadas, o módulo *event*, que permite que as aplicações Lua se comuniquem com o documento NCL, é o mais relevante para a execução deste trabalho. Através desse módulo, a aplicação Lua e o documento NCL podem trocar eventos, permitindo uma comunicação com acoplamento fraco, onde cada uma dessas entidades preserva sua autonomia.

Através do módulo *event*, a aplicação Lua pode registrar funções de tratamento aos eventos gerados na aplicação NCL sobre o objeto Lua. Para esse registro, a função *event.register*, apresentada na Figura 4, é utilizada.

Figura 4. Registro de Evento em LUA com o módulo event .

```
...                          -- código de inicialização
function tratador (evt)
    ...                      -- código do tratador de eventos
end
event.register(tratador)     -- registro do tratador de eventos
```

Fonte: (SANTANA, 2009)

O código de inicialização das aplicações Lua é a parte do código que é executada antes que o documento NCL sinalize qualquer evento a essa aplicação. Após essa inicialização, apenas o código do tratador de eventos é executado, toda a vez que ocorre um evento a partir do documento NCL. Assim o código de inicialização é utilizado para criar objetos e funções auxiliares empregadas pelos tratadores.

Complementarmente, para enviar eventos à aplicação NCL, a função *event.post* é utilizada. Para a execução deste trabalho, essa função recebe uma classe NCL, apresentada na Figura 5. Essa classe define o tipo do evento e a ação correspondente que será enviada ao documento NCL em execução.

Figura 5. Postagem de Evento em Lua com o módulo *event*

```
event.post {
     class = 'ncl',
     type = 'presentation',
     action = 'stop'
}
```

Fonte: (SANTANA, 2009)

3 METODOLOGIA

Para execução das aplicações NCL e Lua foi utilizado o middleware Ginga[9] (ABNT, 2007), que consiste em uma camada de software intermediária entre o sistema operacional e as aplicações para TV digital. O Ginga é um software livre, sem restrições para uso não comercial e com seu código fonte disponível para download[10]. Além do código fonte, é possível utilizar o Ginga diretamente em equipamentos, como TVs e conversores digitais (set-top box) ou ainda através de máquinas virtuais[11].

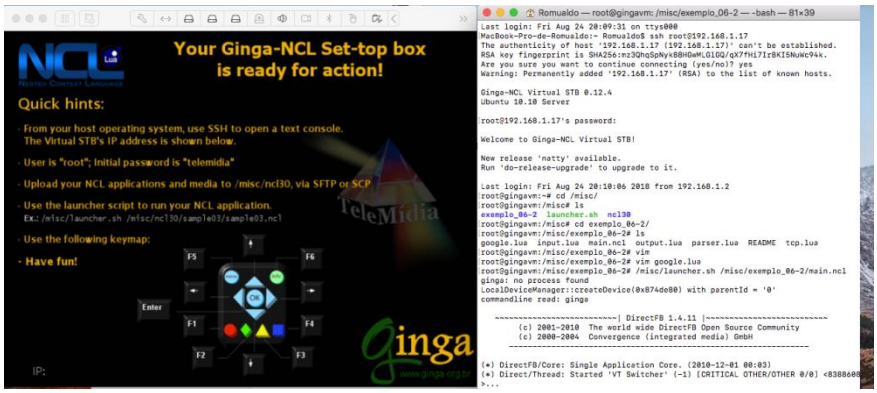
Para acesso a serviços na Internet, o interpretador Lua no middleware Ginga dispõe apenas do módulo TCP, que é integrado ao módulo *event*, através de classe homônima (BRANDÃO, 2010). Foram realizados testes de acesso a serviços Web através desse módulo, incluindo o acesso a buscas no Google[12]. Embora tenha sido possível realizar a comunicação, com os módulos citados, além da dificuldade inerente a especificação dessa comunicação, os módulos citados não disponibilizam requisições a serviços através do HTTPS o que limita, substancialmente, os serviços que atualmente podem ser obtidos.

A fim de habilitar a requisição a serviços Web acessíveis através do HTTPS, módulos adicionais ao interpretador Lua foram testados, sendo escolhido o módulo LuaSockets[13] que disponibiliza requisições HTTP e, mediante a instalação adicional do OpenSSL[14], também requisições HTTPS. A grande dificuldade na instalação desses módulos adicionais está em habilitar no Ginga a sua utilização. Equipamentos, como TVs e conversores digitais, usualmente oferecem bloqueios a esse tipo de modificação que pode, inclusive, provocar problemas no funcionamento do equipamento. Da mesma forma, as

[9] www.ginga.org.br
[10] softwarepublico.gov.br
[11] gingancl.org.br
[12] www.telemidia.puc-rio.br/~francisco/nclua/tutorial/exemplo_06.html
[13] w3.impa.br/~diego/software/luasocket
[14] www.openssl.org

modificações para inclusão do LuaSockets que foram realizadas no código fonte do Ginga, bem como nas versões mais recentes das máquinas virtuais não funcionaram. Por outro lado, nas versões da máquina virtual anteriores à 10.10 esse módulo foi instalado com sucesso, sendo então essa versão, cuja execução da máquina virtual é apresentada na Figura 6, utilizada nos testes realizados.

Figura 6. Acesso à máquina de apresentação Ginga, versão 10.10.



Fonte: do autor

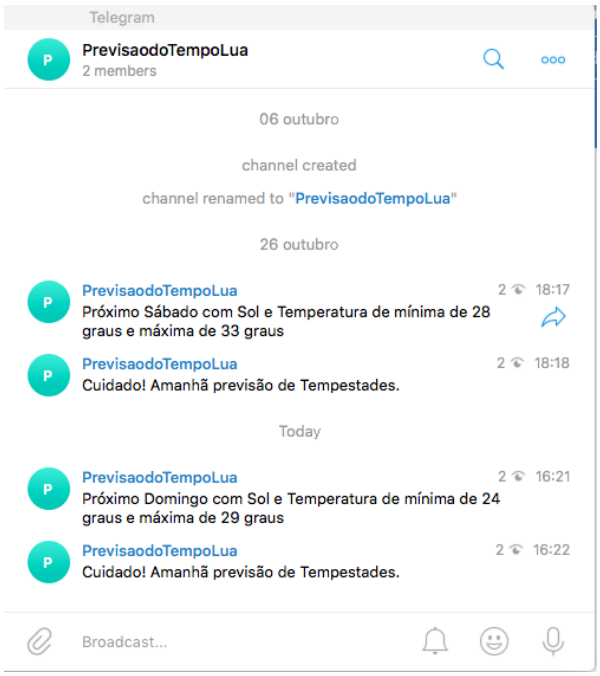
As instruções para acesso à máquina virtual são apresentadas na tela inicial após o seu carregamento, conforme apresentado na Figura 6. O acesso é realizado através do protocolo SSH (IETF, 2006) tanto para a configuração, quanto para a execução das aplicações. Para verificar o funcionamento da máquina virtual foram testadas diversas aplicações, algumas encontradas no Clube NCL[15], com destaque para as aplicações com objetos Lua. Também foram realizados testes com requisições a serviços na Web através do módulo LuaSockets, incluindo, por exemplo, serviços como o Catho[16] e o Linkedin[17].

O outro ambiente necessário a execução deste trabalho que precisa ser configurado, é o Telegram, que se destaca entre os aplicativos de mensagens, por permitir que os usuários possam interagir com outras aplicações. Neste

[15] clube.ncl.org.br
[16] www.catho.com.br
[17] www.linkedin.com

projeto, um Telegram bot[18] conhecido como "Previsão do Tempo Lua" foi construído. Essa aplicação é construída através de um *wizard*, conhecido como BotFather[19], e fica armazenada na própria infraestrutura do Telegram. Dentro desse aplicativo, o próprio é, na verdade, um robô que, ao receber o comando /newBot solicita o nome do novo robô a ser construído (no caso, Previsão do Tempo Lua) e, não havendo coincidência de nomes com outros existentes, fornece uma chave de acesso para enviar e receber mensagens através dessa aplicação. A Figura 7 apresenta a interface do Telegram com as mensagens recebidas pela aplicação.

Figura 7. Bot Previsão do Tempo Lua no Telegram



Fonte: do autor

Na Figura 1, as informações sobre a previsão do tempo são obtidas através do serviço oferecido pelo HGBrasil[20], que oferece a previsão do tempo para os próximos 10 dias. Essas informações são obtidas e processadas pela

[18] core.telegram.org/bots
[19] telgram.me/BotFather
[20] www.hgbrasil.com

aplicação Lua apresentada na Figura 8. Se existe a previsão de algum dia do final de semana ser ensolarado (linhas 16 a 50) ou se está prevista tempestade para o próximo dia (linhas 51 a 60) eventos com essas informações são gerados. Esse é o caso do **evtData** (linha 23)**,** que contém a descrição do dia do final de semana (sábado ou domingo), do **evtMin** e **evtMax** (linhas 29 e 35 da Figura 1), que contém os valores previstos das temperaturas mínima e máxima, respectivamente.

Figura 7. Aplicação Lua para solicitação de serviços.

```
01. function handler (evt)
02.         if evt.type ~= 'attribution' then return end
03.         if evt.name == 'previsaodotempo' then
04.                 local ok, res = pcall(previsaotempo(evt.value))
05.                 if not ok then
06.                         print(res)
07.                 end
08.         end
09. end
10.
11. function previsaotempo(cidade)
12.    http = require "socket.http"
13.    json = require "json"
14.   local
result=http.request("https://api.hgbrasil.com/weather/?format=json&woeid="..cidade)
15.    local res = json.decode(result)
16.    for i=1,10 do
17.          local data = res["results"]["forecast"][i]["weekday"]
18.          local description = res["results"]["forecast"][i]["description”]
19.          local max = res["results"]["forecast"][i]["max”]
20.          local min = res["results"]["forecast"][i]["min”]
21.          if ((data == “Dom”) or (data == “Sáb”)) then
22.             if (description == “Ensolarado”) then
23.              local evtData = {
24.               class = 'ncl',
25.               type = 'attribution',
26.               name = 'data',
27.               value = data,
28.              }
29.              local evtMin = {
30.               class = 'ncl',
31.               type = 'attribution',
32.               name = 'min',
```

```
33.              value = min,
34.             }
35.             local evtMax = {
36.              class = 'ncl',
37.              type = 'attribution',
38.              name = 'max',
39.              value = max,
40.             }
41.             evtData.action='start'
42.             event.post(evtData)
43.             evtMin.action = 'start'
44.             event.post(evtMin)
45.             evtMax.action = 'start'
46.             event.post(evtMin)
47.             break;
48.           end
49.         end
50.   end
51.   description = res["results"]["forecast"][2]["description”]
52.   if (description == “Tempestades”) then
53.        local evtAlerta = {
54.          class = 'ncl',
55.          type = 'attribution',
56.          name = 'previsao',
57.          value = description,
58.        }
59.        evtAlerta.action='start'
60.        event.post(evtAlerta)
61.   end
62.
63.   local evt ={
64.       class ='ncl',
65.       type='presentation',
66.       label='lua1',
67.       action='stop',
68.   }
69.   event.post(evtData)
70.   event.post(evt)
71. end
```

Fonte: do autor

Os eventos da aplicação Lua são enviados para uma outra aplicação, especificada em NCL onde esses eventos, ou mesmo seus valores, representam condições em relacionamentos, conforme apresentado na Figura

2. O último evento que pode ser gerado na aplicação Lua, representada na Figura 7, é o **evtAlerta** (linha 53), que indica à aplicação a possibilidade de ocorrência de tempestades no dia seguinte.

Enquanto a aplicação Lua apresentada na Figura 7 recebe as informações do serviço de previsão do tempo, a aplicação Lua apresentada na Figura 8 realiza a postagem das informações selecionadas no Telegram. A função **chatbot**, apresentada na Figura 8 (linha 12) é quem recebe o valor a ser publicado. Para publicação efetiva, é usado o módulo LUA *socket.http* que realiza a requisição ao Telegram.

Figura 8. Aplicação LUA para publicação no Telegram.

```
01. function handler (evt)
02.         if evt.type ~= 'attribution' then return end
03.         print(evt.name)
04.         if evt.name == 'previsao' then
05.                 local ok, res = pcall(chatbot(evt.value))
06.                 if not ok then
07.                         print(res)
08.                 end
09.         end
10. end
11.
12. function chatbot(value)
13.    http = require "socket.http"
14.    local result = http.request("https://api.telegram.org/bot484847545:
          AAECIYmuwIYe5wYBPjgW-mi-u1VpW1yt8SQ/
          sendMessage?chat_id=@previsaodotempolua&text=Cuidado! Amanhã previsão de
"..value)    15.    local evt = {
16.         class ='ncl',
17.         type='presentation',
18.         label='lua2',
19.         action='stop',
20.       }
21.       event.post(evt)
22. end
23.
24. event.register(handler)
```

Fonte: do autor

4 RESULTADOS E DISCUSSÃO

Em relação ao acesso aos serviços, a requisição e grande parte do processamento das informações foi realizada nas aplicações Lua, sendo a NCL utilizada apenas como ligação entre essas aplicações Lua. Assim, os eventos foram reportados com os valores já processados. Uma outra linha de ação, poderia ser realizar o processamento das informações nos documentos NCL. No entanto, dada a falta de expressividade da NCL para esse fim, essa tarefa seria demasiadamente complexa. Evidentemente, ao colocar grande parte do processamento na aplicação Lua, essas aplicações se tornam mais específicas, diminuindo a possibilidade de reuso o que pode fazer com que seja necessário criar um grande número de aplicações.

Infelizmente, ao longo do projeto, o ambiente de execução, isto é, o middleware Ginga disponível, não favoreceu a construção das aplicações, seja pela ausência de ferramentas de edição e depuração do código mas, principalmente, pela dificuldade em atualizar os componentes Lua. Como exemplo, a instalação do módulo *Socket*, necessário à execução de requisições com o HTTPS, só foi plenamente realizada na versão 10.10 da máquina virtual. Assim, não foi possível realizar a instalação de outras opções de módulos Lua, seja para acesso a serviços utilizando o HTTPS ou mesmo para a realização de outras requisições. O próprio telegram oferece um módulo Lua para acesso aos serviços cuja instalação no middleware não foi possível. Atualmente, o middleware não tem sido atualizado constantemente e as poucas atualizações realizadas não tem favorecido a utilização de novos módulos Lua, fundamentais para a execução das tarefas.

Como trabalho futuro, estão sendo avaliadas ferramentas de automatização de tarefas que poderiam, eventualmente, permitir a interconexão de serviços Web, como o desejado neste trabalho. Entre essas ferramentas

pode ser citado o Microsoft Flow[21], que permite a integração de serviços. Como essa ferramenta permite que, além dos serviços previamente cadastrados, sejam utilizados qualquer serviço acessível via HTTP, ela, eventualmente, poderia ser utilizada como ambiente de trabalho ao invés do middleware Ginga.

5 CONSIDERAÇÕES FINAIS

Neste projeto foi avaliada a possibilidade de reuso de serviços já existentes na Web permitindo a sua agregação e, assim, a construção de novos serviços. Para acesso aos serviços, foi utilizada a linguagem Lua, executada no middleware Ginga. A ligação entre o retorno desses serviços, isto é, entre as aplicações Lua foi realizada utilizando aplicações NCL. Através dessas ferramentas foi possível, inclusive, acessar os serviços através de um dispositivo móvel, utilizando para isso um aplicativo de mensagens.

MASHUP MOBILE APPLICATIONS:
USE OF WEB SERVICES THROUGH A MOBILE MESSAGE APPLICATION

ABSTRACT

This paper presents a publication proposal, in mobile devices, of information obtained from existing Web Services. This information can be combined, making it possible to build new services from existing ones. In this proposal, services are accessed through Lua applications, a scripting language designed to facilitate the development of applications. In addition, the specification of the service combination is performed declaratively using the NCL (Nested Context Model), which is the standardized language for the Brazilian Digital TV System (SBTVD) and ITU-T standard for IPTV. This proposal does not require the installation of specific applications on the mobile device, since the information is received through messages.

Keywords: Web Services. Messages. Mobile Applications. Lua. NCL.

[21] flow.microsoft.com

REFERÊNCIAS

ABNT – Associação Brasileira de Normas e Técnicas. **Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações.** ABNT, NBR 15606-2:2007. Versão corrigida 3. 2009.

ABNT – Associação Brasileira de Normas e Técnicas. **Televisão digital terrestre – Sistema de Transmissão.** ABNT, NBR 15601:2007. Versão corrigida. 2008.

BRANDÃO, M.; ROSSI, R.; SOARES, L.F.G. **Extended Features for the Ginga-NCL Environment: Introduction the LuaTV API**. International Conference on Computer Communication Networks, 2010.

GRAMMEL L., TREUDE, C., STOREY, M.A. **Mashup Environments in Software Engineering.** Proceedings of the 1st Workshop on Web 2.0 for Software. Cape Town, África do Sul. 2010.

IETF – Internet EngineeringTask Force. **Upgrading o TLS Within HTTP/1.1**. Request for Comments 2817. Maio 2000.

IETF – Internet EngineeringTask Force. **The Secure Shell (SSH) Transport Layer Protocol**. Request for Comments 4253. Janeiro 2006.

IETF – Internet EngineeringTask Force. **Hypertext TransferProtocol HTTP/1.1: MessageSyntaxandRouting**. Request For Comments 7230. Junho 2014.

ITU-T – International Telecommunication Union - **Recommendation H.761 - Nested Context Language (NCL) and Ginga-NCL for IPTV Services**. Geneva, Suiça. Abril, 2009.

JIN, L. Y., PARK, J. **Efects of Platformization Strategy on Continuance Intention of Mobile Messaging APPs**. Int. J. Mobile Communications, 3, v. 14, 2016.

SANTANA, F. SOARES NETO, C. S., BARBOSA, S. D. J., SOARES, L. F. G. **Aplicações Declarativas NCL com Objetos NCLua Imperativos Embutidos**. Monografias em Ciência da Computação, 17/09, PUC-RJ, Rio de Janeiro, 2009.