Uma Proposta de Suíte de Testes de Conformidade para o Ginga-NCL

Cláudio F. Pinheiro, Eddie B. L. Filho, Rodrigo R. Oliveira, Anastácio A. M. Cavalcante, Volnei S. Klehm, Daniel P. Pereira e Hallyson L. M. Melo

***Resumo*— Este artigo apresenta uma nova metodologia para a realização de testes de conformidade em implementações Ginga-NCL, baseada em dois elementos principais e complementares: um núcleo de controle e execução, compatível com a especificação Ginga-NCL para a TV digital brasileira e portanto completamente portável para outras plataformas, e servidores de eventos, dependentes do sistema sob investigação, que permitem a execução de testes cuja avaliação necessitaria de informações não disponíveis ao Ginga. Com o objetivo de validar da metodologia proposta, uma suíte de testes baseada na mesma foi implementada. Como resultado, testes de conformidade podem ser executados isoladamente ou agrupados em módulos, com um alto grau de automatização.**

***Palavras-Chave*— Linguagem de Contextos Aninhados, Ginga-NCL, Suíte de Testes de Conformidade, NCLua, Servidores de Eventos.**

***Abstract*— This work presents a novel conformance testing methodology for Ginga-NCL middlewares, which is based on two complimentary elements: an execution and control core, compliant with the brazilian digital TV specification and consequently portable across different platforms, and event servers, dependent on the system under analysis, that allow tests whose validation demands information not available to Ginga. In order to verify the proposed methodology, a test suite based on that was implemented. As a result, conformance tests can now be performed in a stand-alone mode or embedded into a test module, with a high degree of automation.**

***Keywords*— Nested Context Language, Ginga-NCL, Compliance Test Suite, NCLua, Event Servers.**

I. INTRODUÇÃO

Atualmente, no Brasil, a TV digital (TVD) está em processo de consolidação, com vários canais digitais já disponíveis, em diversas capitais, e alguns outros em implantação. Além da imagem em alta definição e do áudio digital, os outros diferenciais encontrados são o guia eletrônico de programação (*Electronic Program Guide* - EPG) e a transmissão de aplicativos interativos, presentes em uma pequena parcela dos canais.

Entretanto, para que os usuários utilizem aplicativos interativos, é necessário que o receptor possua o *middleware* brasileiro instalado, conhecido como Ginga [1], [2]. Este componente de *software* é o responsável pela execução de aplicativos declarativos, em linguagem de contextos aninhados (*Nested Context Language* - NCL), ou imperativos, em Java. A parte declarativa do *middleware* é chamada de Ginga-NCL [3] e a parte imperativa de Ginga-J [4]. O Ginga-NCL, devido a limitações inerentes a linguagens declarativas, como manipulação de dados dinâmicos e estruturas de controle, disponibiliza a possibilidade de inclusão de *scripts* Lua [5], que são considerados como mídias do documento NCL.

Quando o Ginga foi proposto, em 2006, já havia discussões sobre a necessidade de uma suíte de testes, de modo a se garantir a conformidade de dispositivos produzidos para o mercado brasileiro. Porém, não houve a normatização de um sistema de certificação, sendo então adotada a abordagem de auto-certificação, o que resultou em um legado de produtos que não foram submetidos a testes padronizados.

Devido ao que foi exposto, e também ao fato da NCL ter sido recomendada para serviços de TV sobre IP (IPTV) [6], [7], o grupo de estudos 16 da União Internacional de Telecomunicações (*International Telecommunications Union* - ITU) começou o desenvolvimento de um suíte de testes para o Ginga-NCL, baseado no que foi proposto por Araújo et al. (2011) [8]. Neste trabalho, por exemplo, os autores propõem uma execução semiautomática de testes, com a utilização de *scripts* Lua para o controle da suíte e a geração de relatórios.

Apesar disso, um grau de automatização maior geralmente requer o desenvolvimento de componentes com alguma dependência da plataforma. Por exemplo, a abordagem semi-automática apresentada por Araújo et al. (2011) [8] não conseguiria simular o pressionamento de uma tecla, ou mesmo verificar a exibição de mídias (resultado na tela), o que ainda requereria a intervenção de um testador humano.

O presente trabalho apresenta uma nova abordagem para testes de conformidade do Ginga-NCL, através do desenvolvimento de uma metodologia baseada em dois componentes principais: um núcleo NCL/Lua, para a execução e o controle dos testes, e módulos de *software* dependentes do sistema em verificação, conhecidos como servidores, que respondem a requisições do núcleo NCL/Lua para a geração de eventos e obtenção de informações. Os dados resultantes destas ações são então analisados pelos *scripts* de controle, com o objetivo de validar os testes executados. Como consequência, há um aumento no grau de automatização dos testes, mesmo em casos em que a saída seja uma mensagem do tipo *Short Message Service* (SMS) ou uma imagem exibida na tela.

O restante deste artigo está organizado conforme descrito a seguir. Os principais problemas enfrentados, por sistemas de testes, quando aplicados ao Ginga-NCL, são discutidos na seção II. Na seção III, as características da metodologia de testes proposta são apresentadas. Na seção IV, expõem-se alguns aspectos relacionados à especificação de casos de teste para as interfaces de programação (*Application Programming Interface* - API) NCL e Lua. A seção V aborda o esquema básico de funcionamento dos servidores. Finalmente, a seção VI introduz uma implementação da metodologia proposta, conhecida como suíte de testes do ginga (*Ginga Test Suite* - GTS), e apresenta alguns resultados obtidos em testes práticos. As conclusões são apresentadas na seção VII.

Cláudio F. Pinheiro ⋆, Eddie B. L. Filho ⋆†, Rodrigo R. Oliveira ⋆†, Anastácio A. M. Cavalcante ⋆, Volnei S. Klehm ⋆†, Daniel P. Pereira ‡ e Hallyson L. M. Melo ∗. ⋆Centro de Ciência, Tecnologia e Inovação do Pólo Industrial de Manaus, Rua Salvador, 391, Adrianópolis, Manaus-AM, 69057-040, Brasil. †Universidade Federal do Amazonas - UFAM, Av. Gen. Rodrigo Octávio Jordão Ramos, 3000, Manaus - AM, 69077-000, Brasil. ‡Empresa Brasileira de Aeronáutica - EMBRAER, Av. Brigadeiro Faria Lima, 2170, São José dos Campos - SP, 12227-901, Brasil. ∗Instituto Nokia de Tecnologia - INdT, Av. Torquato Tapajós, 7200, Manaus - AM, 69048-660, Brasil. E-mails: {claudio,eddie,rodrigo,anastacio,volnei}@ctpim.org.br, daniel.patrick@embraer.com.br, hallyson.melo@indt.org.br.

II. Os desafios encontrados na verificação de conformidade para o Ginga-NCL

A linguagem NCL, quando utilizada em conjunto com *scripts* Lua, possibilita a criação de uma estrutura de verificação extensível, que facilita a inserção de novos testes e a sua execução sequencial. É possível também criar arquivos de configuração para habilitar/desabilitar testes, ou mesmo calcular a porcentagem de erros encontrados. A criação de tal estrutura, utilizando-se propriedades, ligadores e eventos [9], pode ser relativamente complexa, quando comparada a outras linguagens, mas é completamente possível, como mostrado por Araújo et al. (2011) [8].

Uma vez que toda a estrutura de testes esteja implementada, é preciso então avaliar os recursos do Ginga-NCL que podem ser utilizados na verificação das suas próprias APIs (se possível, de forma automática). A seguir, alguns itens que apresentam dificuldade na execução de testes, devido a dependência de eventos externos ou mesmo à dificuldade de validação automática, são discutidos.

A. Eventos de usuário

A especificação atual do Ginga-NCL define eventos de tecla e de toque, que são provocados pelo usuário. É possível capturar tais eventos em documentos NCL, através de ligadores, e também em *scripts* Lua, através do módulo *event* [1]. Entretanto, as APIs apenas recebem eventos externos, não havendo uma forma de simular o envio dos mesmos, de tal modo que essa funcionalidade não poderia ser testada sem a intervenção do usuário.

B. Canal de retorno

As APIs para o canal de retorno estão disponíveis, na linguagem Lua, através das classes de evento SMS e *Transport Control Protocol* (TCP), do módulo *event*. A aplicação pode então enviar e receber dados, utilizando esses dois meios de comunicação. Entretanto, o canal de retorno por SMS apresenta um comportamento assimétrico, pois possibilita o envio de mensagens SMS padrão, idênticas às utilizadas em telefones móveis, mas permite recebimento apenas de mensagens com um número de porta específico (16000 a 16999), como nos protocolos TCP e *User Datagram Protocol* (UDP).

Entre outros usos, o valor da porta é importante para evitar que a mensagem seja encaminhada diretamente para a caixa de entrada do dispositivo, o que possibilita ao middleware recebê-la e encaminhá-la à aplicação interativa. Para se superar tal dificuldade e viabilizar o teste da API de SMS, é necessário criar uma entidade que responda automaticamente às mensagens enviadas pela aplicação, em uma porta específica.

C. Validação Gráfica

Testes envolvendo diversos elementos da linguagem NCL, como regiões, monomídias visuais (imagem, vídeos, textos, etc.) e atributos de mídias (barra de rolagem, transparência, bordas, foco, etc.), dentre outros itens meramente gráficos, necessitariam de uma verificação gráfica posterior.

No módulo *Canvas*, disponível na linguagem Lua, o método *Canvas::pixel()* pode ser utilizado para se recuperarem os *pixels* da região atribuída à mídia NCLua, o que é bastante útil para se testar a API gráfica da linguagem Lua. Entretanto, não seria possível fazer o mesmo para as regiões da tela onde há mídias NCL, pois a especificação do Ginga-NCL não disponibiliza uma forma de extrair informações gráficas de tais regiões. Sendo assim, surge a necessidade de criação de um mecanismo externo, que seja capaz de capturar a área da tela do dispositivo.

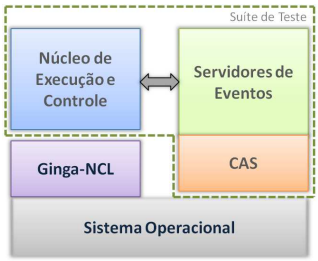

Fig. 1. O modelo de verificação de conformidade proposto.

III. Uma nova proposta de metodologia de testes

O objetivo do presente artigo é apresentar uma nova metodologia de testes para a verificação de aderência, de implementações Ginga-NCL, às normas do Sistema Brasileiro de TV Digital (SBTVD) [1], [2]. Tais testes, chamados de testes de conformidade, têm como finalidade básica analisar as funcionalidades providas pelas APIs envolvidas, para assegurar que uma aplicação carregada em diferentes dispositivos apresente o mesmo comportamento.

No caso de linguagens imperativas, como Java, que são bastante flexíveis, a verificação das APIs pode ser executada por módulos desenvolvidos na própria linguagem [10], sem a necessidade de módulos dependentes do sistema, e ainda cobrir a maior parte das funcionalidade existentes. Além disso, a própria especificação pode prover ferramentas que auxiliem a execução dos testes, como a API *org.dvb.test* presente no *Media Home Platform* (MHP) [12].

Por outro lado, o Ginga-NCL é baseado em um paradigma declarativo, devido à linguagem NCL, o que dificulta o desenvolvimento de um sistema de verificação. Além disso, há muita interdependência entre os elementos da linguagem, o que impossibilita a execução de testes unitários [8].

Outro ponto importante, que deve ser levado em consideração, é a automatização dos testes, o que também é comprometido pela natureza declarativa da NCL. Mesmo com a utilização de *scripts* Lua, não seria possível, por exemplo, comparar imagens exibidas por elos NCL [3] ou simular a interação com o usuário (*e.g.* pressionamento de teclas).

A metodologia proposta agrega componentes dependentes do sistema, que proporcionam um acesso maior aos eventos ocasionados pelos módulos de teste. Tal abordagem, mostrada na Figura 1, pode ser utilizada tanto para a API NCL quanto à Lua, permitindo uma automatização maior do ambiente de testes. A suíte de testes foi dividida em dois componentes principais: o núcleo, que utiliza a estrutura disponibilizada pelo Ginga-NCL e é responsável por todo o controle e a execução dos testes, e os servidores, que compõem a parte dependente da plataforma e permitem simular a interação do resto do sistema com o middleware.

O núcleo de execução e controle é composto por módulos NCL e Lua, cujo objetivo é assistir a execução dos testes. O arquivo NCL principal (main.ncl), que é diretamente executado, fornece toda a infraestrutura básica e fica encarregado de acionar os demais módulos. O início, a execução e o monitoramento da suíte são realizados por *scripts* Lua (ctrl), que determinam o teste a ser realizado e requisitam, quando necessário, eventos dos servidores, recolhendo dados providos

pelos mesmos. Ao final, a máquina de controle detecta o fim do teste e aciona o módulo de avaliação (result), cujo principal objetivo é analisar todos os dados resultantes do teste e gerar um relatório, que é baseado em um comportamento esperado. Esta metodologia está ilustrada na Figura 2.

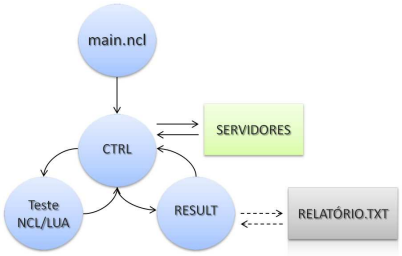

Fig. 2. O esquema utilizado nos testes das APIs NCL e Lua.

Os servidores são componentes de software dependentes da plataforma e podem ser desenvolvidos em linguagens como C/C++ e Java, sendo esta última dependente da existência de uma máquina virtual. Entretanto, para que o servidor seja facilmente portado, permitindo que diferentes plataformas sejam avaliadas pela suíte de testes, inclui-se uma Camada de Abstração de Sistema (CAS), cujo objetivo é traduzir/adequar as requisições do servidor para o sistema alvo.

Nas seções seguintes, o núcleo e os servidores serão detalhados, apresentando-se as suas principais características.

IV. A ESPECIFICAÇÃO DE TESTES PARA AS LINGUAGENS NCL E LUA

Após a exposição dos desafios encontrados (ver seção II), no desenvolvimento de um sistema integrado de testes para o Ginga-NCL, e da metodologia proposta (ver seção III), resta a definição de como a estrutura de testes proposta pode ser realizada. Tendo em vista as diferenças estruturais entre uma linguagem imperativa, no caso da linguagem Lua, e uma linguagem declarativa, no caso da NCL, dois sistemas serão abordados nas subseções seguintes, detalhando-se o processo de concepção de testes para o modelo aqui proposto.

A. Especificação de testes para a API Lua

Os testes relativos aos módulos da linguagem Lua, além de verificar o funcionamento das funções que compõem as APIs, também devem ser capazes de testar o funcionamento geral do sistema, através da combinação de vários métodos, com o objetivo de obter um resultado final.

Os testes, que validam diretamente as APIs, utilizam asserções, comparando o resultado obtido com o resultado esperado de cada função ou método da linguagem. Já nos testes mais complexos, que compreendem uma sequência de operações em linguagem Lua, o resultado final é verificado através do comportamento esperado, tal como a exibição de uma mídia na tela, a manipulação de variáveis de ambiente e até mesmo o envio ou recebimento de dados SMS ou TCP.

Cada teste é composto por um arquivo NCL, denominado *"Test-Caller"*, no qual é declarada uma mídia do tipo NCLua, que constitui o teste propriamente dito.

Sendo assim, todos os casos de teste podem ser realizados de forma integrada à suíte de testes, quando a aplicação NCL principal é executada, e também de forma independente (*stand-alone*), quando o seu *"Test-Caller"* é diretamente executado pelo *middleware*. Durante a execução de testes, de forma integrada, cada resultado é gravado em um arquivo, por módulo, através de uma estrutura de relatórios presente na suíte de testes. Tal abordagem é completamente possível, dado que o módulo *IO* provê facilidades de manipulação de arquivos. Se um determinado teste for realizado em modo *stand-alone*, o seu resultado será escrito diretamente no arquivo de saída padrão.

As asserções, para a verificação de resultados, podem ser feitas através de métodos como

```
gts_assertion:assert_not_nil('assertion_descriptor', value)
gts_assertion:assert_false('assertion_descriptor', value)
gts_assertion:assert_equal('assertion_descriptor', value1, value2).
```

B. Especificação de testes para a API NCL

Dado que supõe-se desejável a execução de testes de forma automatizada, algumas propostas têm surgido na comunidade acadêmica, que utilizam a linguagem Lua como tratador de eventos do formatador NCL. Deste modo, testa-se, de forma parcial, o funcionamento de alguns elos existentes no documento NCL [11]. Tal proposta apresenta, como ponto fraco, a limitação dos itens NCL passíveis de testes automatizados.

Este trabalho aborda os pontos fracos dos esquemas existentes, através da proposição de uma nova metodologia, na qual estejam contemplados tanto a padronização quanto automatização do processo de execução de testes. Para tal, são necessários componentes adicionais aos arquivos NCL nativos, que constituem duas entidades principais: injetores de eventos (*event injectors*) e *scripts* de asserção (*assertion scripts*). A Figura 3 ilustra uma visão estrutural da inter-relação sugerida entre essas entidades e o arquivo de teste principal.

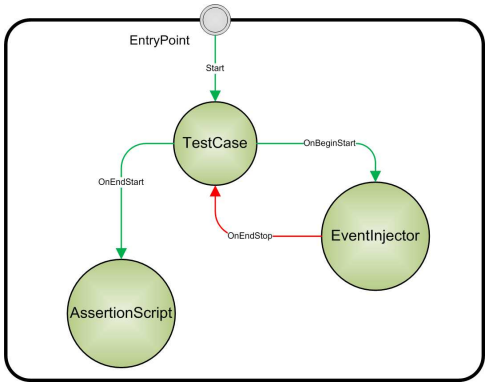

Fig. 3. Visão estrutural das entidades envolvidas em um teste NCL.

Os injetores são, na verdade, *scripts* Lua executados paralelamente ao documento NCL a ser testado, cuja função é comandar servidores de eventos externos (ver seção V). Esses componentes, dependentes da plataforma, interagem com o sistema em teste e fazem com que, por exemplo, eventos idênticos aos gerados pelo usuário sejam retornados ao *middleware*, ou ainda recuperam dados não diretamente acessíveis pela suíte de testes, como a tela em exibição. Um código sugerido para o injetor de eventos está ilustrado na Figura 4, na qual é possível perceber a existência de uma tabela contendo comandos e a sua base de tempo de execução. Essa sequência de ações é então executada pelo controle da suíte de testes, através dos servidores, gerando todos os estímulos necessários a um dado caso de teste.

```
require ("luaEventCtrl.lua")

Time_base = 1000          -- time base ms
-- command table
server_cmds = {
        { 3, 'KEY_A'},    -- {time, event/CAPTURE}
        { 9, 'CAPTURE'}, -- {time, event/CAPTURE}
}

--Starts Server
set_test_ID('TM000_TC00215')
start_evt_server()

--Process ComandTable
set_evt_timebase(Time_base)
set_evt_table(server_cmds)

--Stop Event Server
stop_evt_server()
```

Fig. 4. Código exemplo de um possível injetor de eventos.

Com as imagens capturadas pelos servidores, os resultados dos testes poderão então ser obtidos manualmente (teste semiautomático), através da inspeção visual dos arquivos gerados, ou automaticamente, através do uso de um *script* de verificação, que seja capaz de analisar a tela em exibição. Nesse último caso, para que a validação do teste seja simplificada, adota-se a seguinte estratégia: na parte superior da tela há uma referência para o resultado obtido, com todos os campos que devem ser gerados, sendo que a parte inferior contém os elementos criados pelo caso de teste. Esta abordagem está ilustrada na Figura 5.

Fig. 5. Da esquerda para direita: divisão da tela, teste correto e teste falho.

Com a abordagem aqui mostrada, é possível testar uma grande parte dos itens de um documento NCL. Outra vantagem importante é a possibilidade de execução sequencial e automatizada de testes.

V. Os servidores de eventos

O Ginga-NCL é capaz de se comunicar, com o meio exterior, através do mecanismo de eventos, que permite um acesso simplificado ao protocolo TCP, a mensagens SMS e à requisições do sistema, como o pressionamento de teclas. Para que essas funcionalidades sejam testadas, a metodologia proposta utiliza 3 servidores de eventos, que estão ilustrados na Figura 6: o de comunicação TCP, o de mensagens SMS e o de teclas e tela. O servidor TCP recebe e envia mensagens através deste protocolo, para que a suíte de testes valide as funcionalidades inerentes a esse tipo de comunicação. O servidor SMS, por sua vez, realiza a troca de mensagens SMS com os módulos de teste, permitindo assim uma avaliação direta do que foi enviado e recebido pelo Ginga-NCL. Finalmente, o servidor de teclas e tela é utilizado para validar a interação com o hardware e os resultados de elos NCL, sendo uma ferramenta importante na verificação da parte declarativa do Ginga-NCL.

As funcionalidades da comunicação TCP são testadas através do envio e da recepção de eventos de classe *'tcp'* [1], em *scripts* Lua. Com o objetivo de verificar tal comunicação,

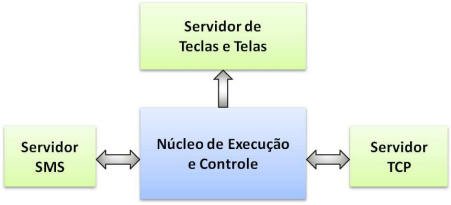

Fig. 6. Comunicação entre do Núcleo de Execução e Controle e os servidores.

o servidor TCP monitora um determinado conjunto de portas, nas quais a comunicação será realizada, respondendo às mensagens enviadas pelos módulos de teste. As respostas são então avaliadas pelo controle da suíte de testes, de modo a validar os caracteres enviados e toda a informação de controle associada.

Como mencionado na seção II, a API de mensagens SMS, presente na linguagem Lua, apresenta uma assimetria em sua concepção, o que dificulta a verificação da mesma. Devido a isso, não é possível, através da API de SMS, uma comunicação direta entre aplicativos Ginga-NCL, pois as mensagens enviadas não alcançariam os objetos imperativos NCLua. Sendo assim, para se preencher esta lacuna, surgiu a necessidade de um servidor específico, chamado de servidor SMS. O objetivo deste componente é receber tais mensagens e retorná-las ao remetente, permitindo a validação, na origem do envio, da cadeia de comunicação. Vale ressaltar que tanto o servidor TCP quanto o SMS não estão necessariamente localizados no mesmo receptor com o *middleware* sob avaliação, podendo ser agrupados em um "dispositivo servidor" e então acessados por várias implementações em teste.

O Ginga-NCL é baseado em uma arquitetura declarativa, na qual as mídias são declaradas e exibidas em regiões [1]. Para que a aplicação seja capaz de reagir a eventos externos, conectores são declarados no documento, cujo objetivo é definir como os elos devem ser acionados. Além disso, a navegação é implícita e o foco nas mídias muda de acordo com uma ordem pré-estabelecida, que pode ser dependente das declarações feitas pelo usuário.

Apesar da facilidade em se definir a estrutura básica da aplicação, não há ferramentas que permitam avaliar a efetividade das instruções executadas. Sendo assim, surgiu a necessidade de se criar um componente externo, que permitisse validar a estrutura criada e verificar se o que é exibido na tela está realmente de acordo com o especificado no documento NCL. Tal componente é, na verdade, um servidor de teclas e tela, cujo objetivo é copiar a tela atual e enviá-la ao controle da suíte de testes, que então valida a execução do NCL. Este servidor também inclui a capacidade de geração de teclas, às quais os elos NCL podem responder, permitindo a simulação completa do fluxo de execução.

VI. O GTS e seus resultados práticos

O *Ginga Test Suite* (GTS) surgiu devido à necessidade de se testar e validar, de forma automática, as APIs NCL e Lua de implementações Ginga-NCL. O GTS implementa a metodologia proposta nas seções III, IV e V, dividindo as APIs existentes em módulos, que testam um conjunto comum de funcionalidades especificadas na norma Ginga-NCL [1].

Há um conjunto de testes para cada um dos módulos adicionais da linguagem Lua, presentes no Ginga-NCL: *IO*, *Canvas*, *Persistent*, *Settings* e *Event* [1], além de um outro conjunto denominado *"Existing"*, para verificar os módulos

Lua que foram removidos da especificação Ginga-NCL. No caso da NCL, os módulos de teste representam diretamente os elementos da linguagem: *structure*, *layout*, *components*, etc.

Conforme mencionado, a abordagem proposta possibilita a portabilidade para outras plataformas, baseadas em diferentes sistemas operacionais, desde que o Ginga esteja presente. Nos testes que envolvem informações ou facilidades não acessíveis ao *middleware*, utilizam-se três servidores de eventos: o servidor TCP, o servidor SMS e o servidor teclas e tela. Os servidores SMS e de Teclas e Tela foram implementados em C/C++, com uma camada de portabilidade baseada em Qt, e o de TCP foi implementado em Java.

Visando ampliar o seu escopo de utilização, o GTS foi concebido de forma a permitir maior flexibilidade na execução dos testes. Através de arquivos de configuração, é possível selecionar os módulos a serem executados e os testes ativos em cada um deles.

Durante os testes de uma funcionalidade específica, não é interessante se aguardar a execução de todos os casos de teste, para só então verificar o ponto desejado. Prevendo situações como estas, o GTS disponibiliza a execução de casos de teste isolados, com resultados mostrados diretamente na tela. Neste modo, o teste é executado sem a necessidade de se alterar arquivos de configuração ou carregar a estrutura completa do GTS. Os casos de teste, carregados diretamente através de seu arquivo NCL principal, utilizam apenas uma pequena parte da estrutura do GTS, pois são autocontidos. Entretanto, as asserções não persistem em memória.

Em testes de módulos NCL puros, é necessário, antes de tudo, iniciar o servidor de teclas e tela, de modo a se avaliar a imagem gerada pelo caso de teste.

Para que a suíte de testes fosse validada, o GTS foi aplicado a uma implementação Ginga-NCL comercial para dispositivos portáteis, que foi desenvolvida com base nas normas ABNT NBR 15606-2/5. A execução do módulo Canvas, realizada durante essa avaliação, está mostrada na Figura 7.

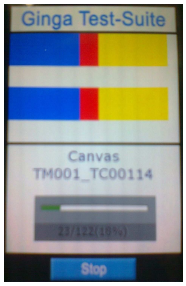

Fig. 7. Testes de conformidade no módulo *Canvas*.

Os servidores de teclas e tela e TCP foram instalados no mesmo dispositivo em teste, ao passo que o servidor SMS foi disponibilizado em outro aparelho. As capturas de tela foram armazenadas em arquivos PNG e disponibilizadas diretamente em um diretório acessível pelo GTS.

Após a execução da suíte de testes, composta de aproximadamente 200 casos de teste, e com base nos resultados obtidos, alguns pontos principais podem ser destacados. Primeiramente, a execução da suíte completa não levou mais que 30 minutos, sem qualquer intervenção ou assistência humana. Apenas, em alguns momentos, o dispositivo foi inspecionado, para se verificar se houve travamento e se os resultados estavam sendo exibidos na tela, para cada teste. Vale mencionar que os mesmos testes foram executados manualmente, o que resultou em um tempo de avaliação até 5 vezes maior. Dois modelos do dispositivo testado foram avaliados e exibiram os resultados finais. Entretanto, dispositivos mais simples chegaram a acusar indisponibilidade de memória. Sendo assim, é interessante se alocar bastante memória, quando da execução do GTS. Os relatórios gerados apontaram onde o comportamento do *middleware* não estava de acordo com a norma, o que resultou em uma média de, aproximadamente, 20% de não conformidades. Tal informação foi diretamente retornada aos desenvolvedores da implementação, que então começaram um trabalho de correção de código.

VII. Conclusões

Através de uma metodologia baseada em código NCL/Lua e servidores, que representa a contribuição deste trabalho, é possível avaliar uma grande parte da especificação Ginga-NCL, com um alto grau de automatização e também com um tempo de execução reduzido. Tal abordagem é mais interessante que a normalmente utilizada para exibidores de linguagens baseadas em XML, que é preponderantemente manual. Além disso, a metodologia proposta pode ser aplicada tanto à API NCL quanto à Lua, gerando uma solução abrangente e integrada, que pode ser incorporada a uma suíte de testes existente.

Agradecimentos

Os autores gostariam de agradecer à empresa Nokia, por financiar o projeto que resultou no desenvolvimento do GTS, em cooperação com o Instituto Nokia de Tecnologia - INdT.

Referências

[1] ABNT NBR 15606-2: Digital terrestrial television - Data coding and transmission specification for digital broadcasting Part 2: Ginga-NCL for fixed and mobile receivers - XML application language for application coding, Brazil: 2011.

[2] ABNT NBR 15606-5: Digital terrestrial television - Data coding and transmission specification for digital broadcasting Part 5: Ginga-NCL for portable receivers - XML application language for application coding, Brazil: 2008.

[3] L. F. G. Soares, R. F. Rodrigues and M. F. Moreno, "Ginga-NCL: the declarative environment of the Brazilian digital TV system", *Journal of the Brazilian Computer Society*, Vol. 12, No. 4, pp. 37-46, August 2007.

[4] G. L. S. Filho, L. E. C. Leite and C. E. C. F. Batista, "Ginga-J: The Procedural Middleware for the Brazilian Digital TV System", *Journal of the Brazilian Computer Society*, Vol. 12, No. 4, pp. 47-56, August 2007.

[5] F. Sant'Anna, R. Cerqueira and L. F. G. Soares, "NCLua - Objetos Imperativos Lua na Linguagem Declarativa NCL", in proceedings of the *14th Brazilian Symposium on Multimedia and the Web*, Vila Velha, Brazil, October 2008.

[6] ITU-T Recommendation H.761, Nested Context Language (NCL) and Ginga-NCL for IPTV Services, Geneva, April 2009.

[7] L. F. G. Soares, M. F. Moreno, C. S. S. Neto and M. F. Moreno, "Ginga-NCL: Declarative Middleware for Multimedia IPTV Services", *IEEE Communications Magazine*, Vol. 48, No. 6, pp. 74-81, June 2010.

[8] E. C. Araújo, L. R. Redlich, V. Lago, M. F. Moreno and L. F. G. Soares, "Suíte de teste de conformidade para o Ginga-NCL", in proceedings of the *17th Brazilian Symposium on Multimedia and the Web*, pp. 9-17, Florianópolis, Brazil, October 2011.

[9] L. F. G. Soares and S. D. J. Barbosa, Programando em Ncl 3.0, 1st ed, Brazil: Campus, 2009.

[10] A. Twigger, "Guidelines for the Development of Test Suites for DVB MHP Implementation Testing", UniSoft Limited, September 2001, retrieved March 8, 2012, from http://www.unisoft.com/mhp/MEG038r2.doc.

[11] E. C. Araújo, L. R. Redlich, V. Lago, M. F. Moreno, L. F. G. Soares, "Nested Context Language 3.0 Parte 14 - Suíte de Testes de Conformidade para o Ginga-NCL", *Monografias em Ciência da Computação*, No. 10/11, ISSN: 0103-9741, September 2011.

[12] C. S. Martín, L. García, J. M. Menéndez and G. Cisneros, "Access Services Based on MHP Interactive Applications", *IEEE Transactions on Consumer Electronics*, Vol. 56, No. 1, pp. 198-202, February 2010.