

Transmissão de Aplicações e Comandos de Edição ao Vivo em IPTV e DTV Terrestre

Marcio Ferreira Moreno; Marcelo Ferreira Moreno; Luiz Fernando Gomes Soares

Departamento de Informática – PUC-Rio

Rua Marquês de São Vicente, 225

Rio de Janeiro/RJ – 22453-900 - Brasil

{mfmoreno, moreno, lfgs}@inf.puc-rio.br

ABSTRACT

DTV applications (with their related media objects) and live editing commands (with their associated parameters) are transmitted embedded in data structures supported by asynchronous transport services that must be defined by each particular DTV system. This paper proposes several transport alternatives to Ginga-NCL middleware, stressing their data structures that allow for not only controlling the application life cycle and mapping the authoring syntax to the transfer syntax without the author's intervention and knowledge, but also for improving the transmission and processing performance of DTV applications. Although focusing on the Ginga-NCL middleware for terrestrial TV and IPTV systems, the proposed alternatives can be extended to other middlewares.

RESUMO

Aplicações de TV digital (com seus objetos de mídia relacionados) e comandos de edição de aplicações ao vivo (com seus parâmetros associados) são transportados em estruturas de dados de serviços assíncronos definidas em cada sistema de TV Digital específico. Este artigo propõe várias alternativas de transportes para o middleware Ginga-NCL, salientando as estruturas de dados que permitem não apenas o controle do ciclo de vida das aplicações e o mapeamento de referências do ambiente de autoria na sintaxe de transferência sem a intervenção, ou conhecimento, do autor, mas também um maior desempenho na transmissão e processamento de aplicações. Embora propostas para o middleware Ginga-NCL, focando tanto a TV digital terrestre quanto sistemas de IPTV, as opções podem ser estendidas a outros middlewares.

Categories and Subject Descriptors

I.7.2 [Document Preparation]: Languages and systems, Hypertext/hypermedia, Markup languages, Standards.

General Terms

Algorithm, Design, Standardization, Languages.

Keywords

Multimedia Synchronism, Declarative Middleware, Ginga-NCL, Interactive DTV, NCL, SBTVD-T.

1. INTRODUÇÃO

Em um sistema de TV digital, além dos fluxos de áudio principal (ou primário) e vídeo principal (ou primário), outros objetos de mídia podem ter suas exibições sincronizadas no tempo e no espaço, compondo o que se chama um *programa não-linear de TV*, ou uma *aplicação DTV*.

Como exemplo, usado em todo este artigo, suponha a aplicação “O Primeiro João”, bem simples, tendo como fluxo audiovisual principal um desenho

animado sobre os dribles do jogador “Garrincha” (objeto de mídia do tipo vídeo). Suponha ainda que em certo instante da animação, é apresentado um ícone de uma chuteira (objeto de mídia do tipo imagem) que, se selecionado por uma dada tecla do controle remoto, redimensiona a área de exibição do vídeo da animação, faz aparecer um novo objeto de vídeo com a propaganda da marca da chuteira, junto a um formulário (objeto de mídia declarativo HTML) para compra da chuteira, sendo todos esses objetos superpostos a uma imagem de fundo (objeto de mídia do tipo imagem), como ilustra a Figura 1.



Figura 1. Aplicação DTV “O Primeiro João”.

Em aplicações DTV, os fluxos de áudio e de vídeo principal podem ser transmitidos por *multicast*, como em sistemas IPTV, ou mesmo por difusão, como acontece nos sistemas de TV digital terrestre. Os demais objetos de mídia da aplicação, bem como o documento de especificação responsável por relacionar no tempo e no espaço os vários objetos de mídia, podem ser obtidos do mesmo serviço (ou da mesma rede) em que são transportados os fluxos de áudio e de vídeo principal, ou de outras redes, como por exemplo, o canal de retorno (ou de interatividade) em um sistema de TV digital terrestre.

Independente da forma de transmissão, a localização dos vários objetos de mídia de uma aplicação é definida pelo documento de especificação da aplicação, criado no que chamamos de *fase de autoria*. Como a localização conhecida pelo sistema de autoria é diferente da localização dos mesmos objetos quando transportados (*fase de transferência*) e entregues a um receptor para exibição (*fase de apresentação*), metadados são necessários para que a sintaxe de autoria de uma aplicação possa ser compreendida em seu destino final.

O controle do ciclo de vida das aplicações, principalmente as geradas ao vivo e aquelas relacionadas temporalmente com o fluxo audiovisual principal, também requer o uso de metadados para o transporte de comandos de controle.

As soluções, para os dois problemas mencionados acima, oferecidas pelos middlewares padrões para TV digital apresentam algumas limitações importantes. A exceção é o middleware Ginga-NCL, que propõe soluções alternativas, contornando as limitações presentes em outros sistemas.

O middleware Ginga-NCL e sua linguagem declarativa NCL (*Nested Context Language*) foram adotados pelo Sistema Brasileiro de TV Digital Terrestre (SBTVD) [1] em 2007. No início de 2009, NCL e o Ginga-NCL se tornaram parte dos padrões ISDB (o antigo padrão japonês, mas agora

¹ Transmission of Applications and Live Editing Commands for IPTV and Terrestrial DTV

incorporando as inovações brasileiras, renomeado como *International Standard for Digital Broadcasting*) e parte da recomendação ITU-R BT 1699 [2]. Ainda no início de 2009, NCL e Ginga-NCL tornaram-se a primeira tecnologia padronizada para aplicações multimídia para serviços IPTV, por meio da Recomendação ITU-T H.761 [3]. NCL e Ginga-NCL vêm sendo projetados no laboratório TeleMídia da PUC-Rio. O trabalho vem sendo coordenado pelos autores deste artigo, que também são os responsáveis pela edição da Recomendação ITU-T H.761 e pela coordenação do Grupo de Trabalho de Middleware do SBTVD.

Este artigo apresenta várias propostas dos autores como opções de transporte desenvolvidas para o middleware Ginga-NCL. Ao adotar o middleware, cada sistema de DTV deve definir sua opção. O artigo expõe e traz uma discussão dessas várias opções, exemplificando seus usos e discutindo suas particularidades. Inicialmente, os problemas mencionados nesta introdução são mais detalhados na Seção 2, como motivação. Na Seção 3 são apresentados alguns dos principais trabalhos relacionados ao tema, ligados aos principais sistemas de DTV existentes. A Seção 4 discute as estruturas de dados necessárias para que um sistema receptor possa interpretar e saber referenciar os dados de aplicações e parâmetros de comandos de edição enviados ao middleware Ginga-NCL. A Seção 5 discute como essas estruturas podem ser envelopadas e transmitidas via protocolos para dados obtidos sem solicitação (*pushed data*). A Seção 6 é reservada para as considerações finais do trabalho. Embora toda a discussão gire em torno do middleware Ginga, as soluções propostas neste artigo podem ser adaptadas a outros middlewares.

2. MOTIVAÇÃO

Mapeamento de referências (identificação de recursos)

Como exemplificado pela Figura 1, uma aplicação DTV é composta de um documento, escrito em alguma linguagem de programação (declarativa ou imperativa), especificando como os objetos de mídia, que também compõem a aplicação, se relacionam no tempo e no espaço. Ao desenvolver a especificação de uma aplicação, o autor faz referência a uma série de conteúdos (de seus objetos de mídia). Cada conteúdo, ou constitui um fluxo elementar contínuo (principalmente os conteúdos gerados ao vivo), ou é um arquivo em um sistema de arquivos de um servidor. Os identificadores de conteúdos presentes no documento de especificação da aplicação se referem, assim, à localização do recurso conhecida pelo *ambiente de autoria*.

Retomando o exemplo da Figura 1, a aplicação deve se referir ao fluxo audiovisual e também aos arquivos de conteúdo de mídia (sistema de arquivos ilustrado na Figura 2), com os respectivos localizadores (*locators*) definidos no ambiente de autoria. A aplicação, também em um arquivo (localizado na Figura 2 em `c:\nclRepository\applications\primeiroJoao.ncl`) deve ser iniciada assim que o serviço com o vídeo da animação for sintonizado. Note que esse exemplo apresenta um caso comum em que a base temporal para o início do documento é um objeto de mídia (o vídeo da animação) especificado no próprio documento.

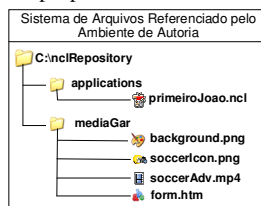


Figura 2. Sistemas de arquivos da aplicação TVD.

Para serem transportados, os fluxos e os sistemas de arquivos da aplicação devem ser colocados em uma estrutura de transporte. Nesse momento, pode surgir uma primeira *inconsistência*, visto que o documento da aplicação transportado referenciará os recursos (conteúdos) com localizadores que refletem o caminho, para esses dados, nos servidores e não o caminho no *sistema de transporte* que no momento os transporta. Dessa forma, no receptor cliente de destino, a aplicação referenciará dados da forma como seriam acessados pelo ambiente de autoria e não como estão codificados no fluxo recebido. Assim, deve haver algum mecanismo que mapeie a sintaxe concreta de transferência para a sintaxe abstrata utilizada no documento da aplicação. Se os dados forem recebidos sob demanda (*pulled data*), o próprio receptor, que faz a demanda, pode inferir sobre o mapeamento (sem metadados adicionais), uma vez que ele fez a solicitação, uma a uma. Entretanto, dados recebidos sem solicitação (*pushed data*) devem vir

acompanhados de algum *metadado* que permita tal mapeamento pelo receptor. Nos dados recebidos sem solicitação reside o problema, e sua forma de transmissão, descrita a seguir, tem de ser bem entendida.

Para *transmissões ao vivo*², uma vez que a sintonização de um serviço (canal) específico pode ser realizada em qualquer instante, dados recebidos sem solicitação que não tenham relações temporais especificadas por meio de selos de tempo devem ser enviados *ciclicamente*, da mesma forma que o próprio documento de especificação da aplicação. Se assim for feito, o recebimento desses dados será independente do instante de sintonização.

Os metadados, que permitem o mapeamento no receptor dos identificadores criados no ambiente de autoria para sua localização no sistema de transporte, também deverão ser enviados ciclicamente, para que o mapeamento sempre possa ser realizado, independente do momento de sintonização.

Controle do ciclo de vida das aplicações

Para controlar o ciclo de vida de uma aplicação, por exemplo, para iniciá-la, pausá-la, pará-la etc., comandos devem ser enviados ao receptor. Em geral, no entanto, *comandos de edição* permitem manipulações mais sofisticadas de aplicações, até mesmo sua geração ou modificação ao vivo. Tal é o caso de aplicações geradas na linguagem NCL [4] e com suporte no middleware Ginga-NCL.

Tal qual uma aplicação com seus objetos de mídia relacionados, comandos de edição podem ser recebidos pelo mesmo serviço (ou pela mesma rede, ou canal) onde são transportados os fluxos de áudio e de vídeo principal, ou por outras redes (ou canais). Mais ainda, comandos de edição podem estar embutidos nos próprios objetos com código imperativo que compõem uma aplicação, permitindo, assim, que a própria aplicação se altere, como um autêntico sistema reflexivo. É também possível receber comandos de edição diretamente do telespectador, fazendo uso de um aplicativo residente no receptor.

Comandos de edição são envelopados em estruturas de dados que carregam uma série de parâmetros adicionais, inclusive o tempo de ocorrência do comando. Eles são tipicamente recebidos sem a solicitação do receptor e, portanto, devem ter alguma forma de garantia de seu recebimento, ou, no mínimo ter reduzida a probabilidade de seu não recebimento, quando a garantia não for possível. De fato, o modelo de referência de um sistema de TV digital define não apenas essas estruturas de dados, mas como elas são transportadas, ou seja, os protocolos para transmissão.

3. TRABALHOS RELACIONADOS

Mapeamento de referências (identificação de recursos)

Os principais sistemas de TV digital interativa por difusão terrestre [5], o europeu DVB (Digital Video Broadcast), o americano ATSC (*Advanced Television System Committee*) e o japonês ARIB (*Association of Radio Industries and Businesses*), definem o suporte a execução de aplicações especificadas tanto em linguagens declarativas quanto imperativas. Mais especificamente, esses sistemas definem o suporte a execução de aplicações imperativas na linguagem Java; e a aplicações declarativas especificadas em uma linguagem baseada nos padrões XHTML, CSS e DOM, que podem utilizar ainda a expressividade da linguagem imperativa ECMAScript em objetos embutidos.

Como formato de transporte de uma aplicação (sua especificação e os conteúdos por ela referenciados), os sistemas mencionados fazem uso de carrosséis DSM-CC [6]. O padrão DSM-CC especifica dois tipos de carrosséis: o carrossel de dados, usado pelo ARIB, e o carrossel de objetos, utilizado pelos sistemas DVB e ATSC.

Uma questão importante sobre o carrossel de objetos é o fato que, embora ele preserve a estrutura de arquivos e diretórios nele enviada sem a necessidade de informações adicionais, ele não permite inferir o localizador (no sistema de autoria) da raiz da estrutura de arquivos recebida. Os três sistemas especificam tais informações adicionais por meio de descritores de localidades (*location descriptors*) [7], que informam qual é o localizador (no sistema de autoria) da raiz da estrutura de arquivos transportada no

² Deve-se aqui fazer uma diferença entre transmissão ao vivo e geração de conteúdo ao vivo. Transmissão ao vivo pode ser de um conteúdo previamente gerado. Ela simplesmente indica que os usuários não têm controle de seu início.

carrossel, o diretório base (no carrossel de objetos) associado à raiz, além do caminho do arquivo de especificação da aplicação DTV, relativo à raiz [7].

Os descritores de localidade, no entanto, apresentam um fator limitante: nas especificações das aplicações (feitas no ambiente de autoria) não é permitido o uso de referências absolutas para objetos, se eles forem transportados no carrossel. Por exemplo, o mecanismo não contempla a transmissão de aplicações armazenadas no ambiente do provedor de serviços que referenciem conteúdos dispostos em outros dispositivos de armazenamento compartilhados; ou mesmo recursos dispostos em discos ou partições diferentes de uma mesma máquina, mesmo que esses recursos estejam sendo transmitidos em outro carrossel de objetos a parte. Em outras palavras, o sistema de arquivos que compõe uma aplicação deve compor uma única árvore de diretórios. Uma discussão mais detalhada sobre os descritores de localidade pode ser encontrada em [7].

Outra questão que merece atenção é o *overhead* computacional e de transmissão do carrossel de objetos. O envelopamento de arquivos e diretórios é realizado em várias camadas e mensagens auxiliares são especificadas para informar detalhes do carrossel como, por exemplo, tamanho e versão das estruturas transportadas [6].

Ainda outro problema relacionado aos carrosséis é o controle de versão das estruturas transportadas. Uma incoerência pode ocorrer quando uma estrutura for atualizada ao mesmo tempo em que um cliente receptor está com uma montagem do carrossel em andamento. Se o cliente detectar que a versão de uma determinada estrutura não corresponde com a versão informada nas mensagens auxiliares, a montagem deve ser abortada [6]. Outras incoerências podem ocorrer, porém com maior grau de dificuldade de detecção, devido a referências distribuídas, possivelmente presentes em uma aplicação interativa, fazendo com que atualizações no carrossel de dados, bem como no carrossel de objetos, sejam consideradas tarefas com nível de dificuldade elevado.

O sistema ARIB define uma alternativa ao carrossel de objetos, especificando uma abstração própria para o carrossel de dados DSM-CC. Nessa abstração, os descritores de localidades são enviados em mensagens auxiliares do carrossel de dados, que transportam também informações sobre o sistema de arquivos a ser criado no receptor cliente. Porém, além de apresentar o mesmo ponto crítico anteriormente mencionado dos descritores de localidades do carrossel de objetos, e a dificuldade de realizar atualizações, herdada do carrossel de dados, existe a restrição de no máximo quatro níveis na hierarquia de diretórios na abstração especificada pelo sistema japonês [8].

Sistemas IPTV podem utilizar a multiplexação MPEG-2 para o fluxo de transporte, incluindo a codificação de dados DSM-CC. No entanto, redes de distribuição IPTV são tipicamente *multicast* e apresentam características diferenciadas, como o gerenciamento de qualidade de serviço, a possibilidade de ocorrência de congestionamentos e de erros de transmissão. Em sistemas IPTV é comum a adoção do protocolo de transporte RTP [9]. Nesse protocolo, fluxos de áudio e vídeo podem ser enviados em diferentes sessões RTP, cada qual possuindo suas marcações de tempo, usadas para sincronizar a apresentação no receptor.

Para o transporte de dados sob demanda em serviços assíncronos (a especificação da aplicação e seus conteúdos), pode-se adotar qualquer protocolo de aplicação baseado em IP, sendo o HTTP o caso mais comum. Na transmissão de dados sem solicitação (*pushed*) é necessário o uso de um protocolo cíclico, que pode ser o carrossel de objetos DSM-CC, ou outro protocolo mais direcionado às características da rede IPTV. O protocolo FLUTE [10] oferece transporte de arquivos em modo *push*, tanto em *multicast* quanto *unicast*, e é muito usado em sistemas IPTV, sendo também adotado pelo sistema DVB-H [11] e 3GPP MBMS [12].

FLUTE suporta correção de erros baseada em objetos FEC (*Forward Error Correction*) e controle de congestionamento baseado no transporte de objetos do protocolo ALC (*Asynchronous Layered Coding*). Em FLUTE, uma tabela denominada FDT (*File Description Table*), especificada normalmente por meio de um arquivo XML, provê toda a informação necessária para a identificação, localização e recuperação de arquivos. A localização se dá por meio de URIs, que podem ser usadas diretamente no receptor cliente para acesso ao sistema de arquivos montado. A FDT deve ser enviada antes dos arquivos, porém, suas informações não precisam estar completas e podem ser modificadas de forma incremental. Para prover repetições como um carrossel, a FDT (e os arquivos que refere) é enviada periodicamente em uma mesma sessão FLUTE, especificando os mesmos arquivos. Quando é necessária a atualização, modificação ou criação de um

novo carrossel, uma nova FDT é enviada pela mesma sessão FLUTE, com as características do novo carrossel. Apesar de não prover abstrações mais estruturadas que arquivos, FLUTE permite que hierarquias de sistemas de arquivos sejam montadas, ao definir que as referências a arquivos sejam feitas por URIs. Diferente dos sistemas de DTV terrestre anteriormente citados, diferentes raízes podem ser adotadas, com o uso de URIs absolutas. Resta mencionar que as estruturas de dados para o mapeamento de referências propostas neste artigo podem ser facilmente envelopadas em estruturas do FLUTE.

Controle do ciclo de vida das aplicações

Nos sistemas de DTV terrestre citados nesta seção, o controle do ciclo de vida das aplicações é realizado através de comandos enviados na estrutura denominada AIT (*Association Information Table*) [5]. Aplicações podem ser iniciadas, paradas ou abortadas, imediatamente ao recebimento do comando. Note que não há uma forma de agendar o início de uma aplicação para um certo instante do tempo a partir da sintonia de um serviço. No entanto, uma aplicação pode começar e esperar por um evento DSM-CC para sua continuação. Note também que uma aplicação não pode ser pausada por um comando da AIT.

Nos sistemas citados, o suporte à edição ao vivo de aplicações é baseado em eventos DSM-CC [6]. Os eventos DSM-CC são estruturas que contêm um identificador e um valor de referência temporal para sua ocorrência, que pode ser o do momento de sua recepção (eventos conhecidos como *do-it-now*).

Além do identificador e da referência temporal, a estrutura de um evento DSM-CC possui um campo para dados privados que podem ser usados de acordo com uma semântica definida pela aplicação tratadora do evento. Esse campo pode atingir o tamanho máximo de 255 bytes, restringindo assim a possibilidade da definição de semânticas mais elaboradas. Outro detalhe interessante do uso de eventos DSM-CC nesses sistemas está no fato do ambiente de recepção não garantir a recepção de todos os eventos recebidos [5] [6]. Para evitar que um evento DSM-CC seja perdido, ele é enviado diversas vezes pelo provedor de conteúdo, cabendo ao cliente receptor interpretar a possível duplicação de recepção.

Nas aplicações imperativas baseadas em Java, certo grau de edições ao vivo pode ser alcançado criando observadores para monitorar a chegada dos eventos DSM-CC de edição. As linguagens baseadas em XHTML são mais simples de serem utilizadas na autoria de uma aplicação. No entanto, para as tarefas de edição ao vivo, funções de scripts são necessárias (funções ECMAScript permitem a edição da árvore DOM de um documento XHTML [5]). Assim, em ambos os casos, procedimentos imperativos são necessários, exigindo programadores especialistas, conhecedores de bibliotecas e de técnicas de programação específicas, sendo, portanto, mais propensos a erros de programação, uma vez que todo o controle é passado ao programador. Nenhum suporte declarativo é oferecido para a edição de aplicações ao vivo, incluindo o seu início a partir de um ponto qualquer da sintonização de um serviço, ou seu início por diferentes interfaces (partes da aplicação).

FLUTE, usado em sistemas IPTV como já mencionado, não oferece a abstração de estruturas, como os eventos DSM-CC, que possa ser usada para o encapsulamento de comandos de edição. Obviamente, em sistemas IPTV, uma representação de estruturas auxiliares pode ser definida de forma proprietária, como conteúdo de um dos arquivos do carrossel FLUTE. Ginga-NCL estende as soluções presentes nos outros sistemas tanto com relação à identificação de recursos quanto com relação ao controle do ciclo de vida das aplicações, oferecendo várias opções, como discutido no restante deste artigo.

4. COMANDOS E APLICAÇÕES NO GINGA-NCL

O núcleo da máquina de apresentação Ginga-NCL é composto pelo Formador NCL e pelo módulo Gerenciador de Bases Privadas.

O Formador NCL é responsável por receber um documento NCL e controlar sua apresentação, tentando garantir que as relações especificadas entre os objetos de mídia sejam respeitadas. O formador lida com aplicações NCL que são coletadas dentro de uma estrutura de dados conhecida como base privada. Como exemplo, no Sistema Brasileiro de TV Digital Terrestre, o middleware Ginga-NCL associa uma base privada a um canal de televisão.

Os documentos NCL em uma base privada podem não só ser iniciados, parados e abortados, como também pausados e retomados, além de poderem referir-se uns aos outros.

O Gerenciador de Base Privada é responsável por receber comandos de edição de documentos NCL e pela execução desses comandos, incluindo a edição de documentos NCL ativos (documentos sendo apresentados), ou seja, edições ao vivo.

4.1 Descritor de Evento

Os Comandos de Edição NCL [13] [1] [3] definem eventos (ocorrências no tempo) envelopados em uma estrutura chamada *descritor de evento*: a primeira estrutura de interesse deste artigo.

Cada descritor de evento (de edição) tem uma estrutura composta basicamente por um *id*, uma referência de tempo e um campo de dados privados. A identificação define univocamente o evento como sendo de edição (e não cada tipo de comando). No middleware Ginga, outros tipos de evento além dos de edição NCL podem ser enviados. A referência de tempo (campo *eventNPT*) indica o exato momento de ocorrência (disparo) do evento. Um tempo de referência igual a zero informa que o evento de edição deve ser disparado imediatamente após ser recebido. Se diferente de zero, o tempo de disparo é no momento de ocorrência do valor NPT (*Normal Play Time*) [6] especificado. O campo de dados privados oferece suporte para a identificação do comando e a definição de parâmetros do evento de edição, como apresentado na Figura 3.

Diferente dos outros sistemas discutidos na Seção 2, qualquer alteração de comportamento de uma aplicação NCL é feita de forma declarativa, evitando erros e efeitos colaterais discutidos naquela seção. Assim, os comandos de edição NCL devem ter uma sintaxe padrão bem definida, como discutido em [13]. Na Figura 3, o campo *commandTag* identifica univocamente os tipos de comandos de edição. Os comandos são divididos em três grupos: o primeiro para operação da base privada (para abrir, ativar, desativar, fechar e salvar bases privadas); o segundo para manipulação de documentos (para adicionar, salvar e remover um documento em uma base privada e para iniciar, pausar, retomar e parar apresentações de documentos); e o terceiro para manipular entidades NCL de um documento. Para cada entidade NCL, foram definidos os comandos *add* e *remove*. Se uma entidade já existir, o comando *add* tem a semântica de atualização (alteração).

Sintaxe	Número de bits
EventDescriptor() {	
eventId	15
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 a 1928
FCS	8
}	

Figura 3. Descritor de evento para comandos de edição NCL.

Ainda na Figura 3, para permitir o envio de um comando completo, com tamanho superior a 255 bytes, em mais de um descritor de evento, todos os descritores de um mesmo comando devem ser numerados e enviados em seqüência (isto é, não pode ser multiplexado com outros comandos de edição com o mesmo *commandTag*), com o *finalFlag* igual a 0, exceto para o último descritor, que deve ter o campo *finalFlag* igual a 1. O campo *privateDataPayload* contém os parâmetros do comando de edição. Finalmente, o campo FCS contém um *checksum* de todo o campo *privateData*, inclusive o *privateDataLength*.

Os comandos *add* têm entidades NCL como seus argumentos (parâmetros de comando especificados em XML). A consistência do documento é mantida pelo formatador NCL, quer a entidade

especificada já exista ou não. As entidades são definidas utilizando uma notação sintática idêntica à usada pelos esquemas NCL [4] [1]. Se o parâmetro de comando baseado em XML for curto o suficiente, ele pode ser transportado diretamente no *payload* dos descritores de evento. Senão, o *privateDataPayload* transporta um conjunto de pares de referência {cUri, cId}, para a identificação dos recursos, diferente dos sistemas apresentados na Seção 2, como descrito a seguir.

No caso de arquivos recebidos pelo canal de difusão (documentos ou nós³ NCL enviados sem solicitação), cada par relaciona um caminho de arquivo ou diretório de arquivos identificado pelo *sistema de autoria* e sua respectiva localização no sistema de transporte. Não é necessária a inclusão de um par {cUri, cId} no comando para cada arquivo enviado por difusão. Mas é necessário que a partir dos pares {cUri, cId} incluídos no comando, todo arquivo recebido possa ter o seu caminho absoluto deduzido a partir de metadados também enviados ao receptor, conforme discutido na Seção 4.3. Isso equivale a dizer que, a partir dos metadados recebidos e dos pares {cUri, cId} dos comandos, será possível o sistema receptor mapear os conteúdos dos objetos de mídia referenciados no documento NCL na sua localização dentro da base privada que ele gerencia. No caso de arquivos recebidos sob demanda pelo canal de interatividade ou localizados no próprio receptor, nenhum par de referências necessita ser enviado, exceto se o arquivo for o da especificação do documento NCL ou da especificação XML do nó (objeto) NCL que deverá ser adicionado, segundo o comando de adição (*add*) correspondente. Nesse caso, o par {cUri, "null"} deve ser enviado, especificando o caminho do arquivo a ser buscado.

Retomando o exemplo da Figura 1, a adição da aplicação na base privada "TV GINGA" seria realizada pelo comando:

```
addDocument("TV GINGA",
            {"C:\nclRepository\applications", "0x1, 0x1, 0x2"})
```

Campos	Valor
eventId	Identificador de 15 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x05
sequenceNumber	0x00
finalFlag	1
privateDataPayload	"TV GINGA", {"C:\nclRepository\application", "0x1,0x1,0x2"}
FCS	8 bits de checksum

Figura 4. Descritor de evento para o comando *addDocument*.

Note que, nesse caso, o comando de edição faz referência apenas ao caminho do diretório onde está o documento NCL e onde ele será transportado (no caso, em um carrossel de objetos: Service Domain "0x1", module "0x1" e object "0x2", conforme discutido na Seção 5). Com o auxílio de metadados, também recebidos pelo sistema de transporte, todos os arquivos poderão ter seus caminhos absolutos resolvidos a partir desse relacionamento, conforme também discutido na Seção 5.3.2. A Figura 4 apresenta os campos do descritor de evento do comando. Note que o comando determina a adição imediata do documento na base.

4.2 Mapa-de-eventos

Três tipos de estrutura de dados são definidos para dar suporte à transmissão de parâmetros dos comandos de edição NCL, além da estrutura de descritor de evento já definida: mapa, metadados e arquivo de dados. As duas últimas são assuntos da próxima subseção.

³ Um nó (objeto) NCL pode ser um objeto de mídia que compõe a aplicação, ou um agrupamento de objetos (de mídia ou outros objetos, recursivamente).

Sintaxe	Número de bits
mappingStructure() {	
mappingType	8
for (i=1; i<N; i++) {	
eventId	16
eventNameLength	8
eventName	8 to 255
}	
}	

Figura 5. Lista de ids de eventos definidos pela estrutura de mapa

Para estruturas de *mapa* (*mappingStructure*), o campo *mappingType* identifica o tipo do mapa. Se o valor de *mappingType* for igual a “0x01” (“events”), um mapa-de-eventos é caracterizado. Nesse caso, depois do campo *mappingType*, vem uma lista de identificadores de eventos, como ilustrado na Figura 5. Outros valores para o campo *mappingType* podem ser definidos, mas não são relevantes para a discussão deste artigo.

Mapas-de-eventos são usados para mapear nomes de eventos em *eventIds* dos descritores de evento. Mapas-de-eventos são usados para indicar quais eventos devem ser recebidos. Nomes de eventos permitem especificar tipos de eventos, oferecendo maior nível de abstração às aplicações. Assim, quando for necessário enviar um comando de edição NCL, deve-se criar um mapa-de-eventos, mapeando a string “*nclEditingCommand*” em um *eventId*, previamente selecionado, de um descritor de evento. O Gerenciador de Bases Privadas de um sistema receptor deve se registrar como ouvinte de um evento “*nclEditingCommand*” para ser notificado da chegada desse tipo de evento.

4.3 Arquivos de Dados e Metadados

Cada estrutura *arquivo de dados* é de fato o conteúdo de um arquivo que compõe uma aplicação NCL ou um nó NCL. Isto é, um arquivo contendo a especificação da aplicação, ou um arquivo com a especificação XML de um nó NCL, ou um dos arquivos com conteúdo de mídia de um objeto de mídia qualquer (vídeo, áudio, texto, imagem, etc.).

Para permitir ao sistema receptor a completa independência entre a localização dos dados de uma aplicação em seu sistema de armazenamento local e a localização dos dados como referenciado pelo documento de especificação da aplicação, uma estrutura de *metadados* é definida, conforme o schema XML NCLMetadataFile especificado⁴. Para todo arquivo de dado entregue sem solicitação (*pushed file*), uma associação entre sua localização no sistema de transporte (identificação do sistema de transporte – atributo *component_tag* – e a identificação do arquivo no sistema de transporte – atributo *structureId*) e seu identificador de recurso universal (atributo *uri*), conforme especificado no documento da aplicação, é definida.

```
<metadata name="primeiroJoao" size="50kb">
  <baseData uri="file://c:/nclRepository/applications/">
    <pushedRoot component_tag="0x01.0x09" structureId="0x0A"
      uri="primeiroJoao.ncl" size="10KB"/>
    <pushedData component_tag="0x01.0x09" structureId="0x09"
      uri="./mediaGar/background.png" size="10KB"/>
    <pushedData component_tag="0x01.0x09" structureId="0x08"
      uri="./mediaGar/soccerIcon.png" size="10KB"/>
    <pushedData component_tag="0x01.0x09" structureId="0x07"
      uri="./mediaGar/soccerAdv.mp4" size="10KB"/>
    <pushedData component_tag="0x01.0x09" structureId="0x06"
      uri="./mediaGar/form.htm" size="10KB"/>
  </baseData>
</metadata>
```

Figura 6. Arquivo de metadados do exemplo primeiroJoao

Para cada arquivo de documento NCL ou arquivo de documento XML, usados nos comandos de edição *addDocument* e *addNode*, pelo menos um arquivo de metadados deve ser definido. Apenas um

arquivo de aplicação NCL ou um arquivo de documento XML representando um nó NCL a ser inserido pode ser definido por uma estrutura de metadados (pelo elemento <pushedRoot>). Mais precisamente, pode haver apenas um elemento <pushedRoot> em um arquivo de metadados. Contudo, uma aplicação NCL (e seus arquivos de conteúdo) ou um documento de especificação de um nó NCL (e seus arquivos de conteúdo) podem se estender por mais de um arquivo de metadados. Mais ainda, podem existir arquivos de metadados sem qualquer aplicação NCL ou documento XML descritos em seus elementos <pushedRoot> e <pushedData>.

5. Sistemas de Transporte

As quatro estruturas definidas: arquivo de dados, metadados, mapa-de-eventos e descritores de evento, além dos conteúdos definidos em fluxos elementares de bits, compõem todas as estruturas necessárias para a exibição de uma aplicação DTV. As quatro estruturas podem ser envelopadas em estruturas do sistema de transporte, mas podem também ter seu conteúdo distribuído, principalmente as estruturas de metadados, como discutido nas opções de transporte da Seção 5.2.

5.1 Transporte de Descritores de Evento

Comandos de edição NCL podem ser transportados usando descritores de evento de fluxo DSM-CC [6]. Como especificado em [1], descritores de evento de fluxo DSM-CC têm uma estrutura muito parecida com a dos descritores de evento NCL apresentados na Seção 4.1. Na verdade, a estrutura dos descritores de evento NCL foi definida para tornar o mais fácil possível seu envelopamento em descritores de evento de fluxo DSM-CC.

Descritores de evento de fluxo DSM-CC devem ser enviados mais de uma vez, antes do momento desejado para o disparo do comando, para garantir sua recepção, uma vez que não há garantia de entrega.

O uso de descritores de evento de fluxo DSM-CC foi a solução adotada pelo Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) [14]. Qualquer outro protocolo para envio de dados sem solicitação poderia, no entanto, ter sido adotado, incluindo o FLUTE.

5.2 Transporte de Metadados

Metadados podem ser transmitidos usando o mesmo protocolo utilizado no transporte de arquivos de dados e mapa-de-eventos, ou ainda usando um protocolo diferente. Nesta seção discutiremos esse último caso, deixando para a Seção 5.3 a discussão da transmissão integrada.

5.2.1 Metadados em Descritores de Evento

Uma alternativa para o transporte das estruturas de metadados é tratá-las como parâmetros dos comandos de edição NCL *addDocument* e *addNode*, a serem transportados no campo *privateDataPayload* dos descritores de evento.

Nesse caso, o conjunto de pares {cUri, cId} dos comandos *addDocument* e *addNode* é substituído por parâmetros da estrutura de metadados, que, por sua vez, definem, como apresentado na Figura 6, um conjunto de pares {"uri", "component_tag.structureId"} para cada arquivo transmitido sem solicitação (*pushed file*).

Voltando ao exemplo da Figura 1, o par {cUri; cId} na Figura 4 seria substituído pela estrutura de metadados serializada da Figura 6. Na estrutura de metadados, os atributos *component_tag* dos elementos <pushedRoot> e <pushedData> devem, nesse caso, ser definidos obrigatoriamente, uma vez que a estrutura não é mais transportada no mesmo fluxo que transporta os arquivos de dados da aplicação NCL.

Devido ao baixo overhead envolvido, a transmissão de metadados em descritores de evento provavelmente é a solução mais adequada para sistemas de IPTV que não utilizam Seções MPEG-2 no transporte de dados.

5.2.2 Metadados em Seções de Metadados MPEG-2

Outra alternativa para o transporte de estruturas de metadados é encapsulá-las em Seções de Metadados MPEG-2, transportadas em

⁴ Schema XML disponível em:

<http://www.ncl.org.br/NCL3.0/profiles/NCLMetadataFile.xsd>

fluxos MPEG-2 do tipo “0x16” [15]. Cada Seção de Metadados MPEG-2 poderá conter dados de apenas uma estrutura de metadados. Contudo, uma estrutura de metadados pode se estender por várias Seções de Metadados. Para tanto, um campo da seção (*section_fragment_indication*) é usado, indicando se a estrutura de metadados se estende por mais de uma seção, e, no caso de se estender, se a seção carrega a primeira parte da estrutura, ou a última parte, ou uma parte intermediária.

No campo *privateDataPayload* do descritor de evento dos comandos de edição NCL *addDocument* e *addNode* (vide Figura 3), os pares de referência {*cUri*, *cid*} devem ter os parâmetros *cUri* com o valor “null”. O parâmetro *cid* do primeiro par deve identificar o fluxo elementar TS [15] do tipo= “0x16” e a estrutura de metadados (campo “*structureId*” da seção de metadados) que carrega o caminho absoluto do documento NCL ou da especificação do nó NCL (o caminho no servidor de dados). Se outras estruturas de metadados forem usadas para relacionar arquivos presentes no documento NCL, ou na especificação do nó NCL, a fim de completar os comandos *addDocument*, ou *addNode*, com conteúdos de mídia, outros pares de referência {*cUri*, *cid*} devem ser definidos no comando. Nesse caso, o parâmetro *cUri* deve ter o valor “null” e o parâmetro *cid* correspondente no par deve referir-se ao *component_tag* de um serviço [15] e ao *structureId* do metadado correspondente.

Caso sejam enviadas sem a solicitação do receptor, Seções de Metadados MPEG-2 devem ser repetidas periodicamente para garantir que as estruturas de metadados sejam recebidas pelo cliente, independente do momento da sintonização do serviço que demanda a execução da aplicação DTW.

5.3 Transporte Conjunto das Estruturas

Como já mencionado, as estruturas de metadados, arquivo de dados e mapa-de-eventos podem ser transmitidas usando o mesmo protocolo. As soluções propostas pelo middleware Ginga-NCL são discutidas nesta seção. Elas se diferenciam principalmente pelo overhead imposto pelas várias camadas de encapsulamento das estruturas.

5.3.1 Carrossel de Dados

Para transmissão das estruturas, o carrossel de dados DSM-CC pode ser utilizado [6]. Carrossel de dados é a forma mais simples de transmissão cíclica de dados DSM-CC. Nele não existe qualquer indicação sobre o que consistem os dados. As especificações ATSC [16] e ARIB [8] fazem uso dessa modalidade de transmissão.

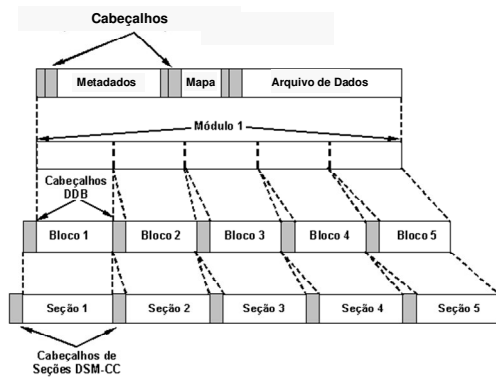


Figura 7. Carrossel de dados DSM-CC.

Um carrossel de dados consiste de uma sequência de módulos, que define um ciclo. Módulos são transmitidos um após o outro até que todos os módulos de um ciclo tenham sido transmitidos, quando todo o processo recomeça. Um módulo pode ser inserido mais de uma vez em um mesmo ciclo do carrossel.

Não existe nenhuma estruturação de mais alto nível acima do módulo definida pelo padrão DSM-CC. Entretanto, o Ginga-NCL pode dar essa semântica de mais alto nível, transportando nos módulos do carrossel suas estruturas de metadados, arquivo de dados e mapa-de-

eventos, em uma solução bem mais expressiva que os outros padrões mencionados nesta seção.

Cada módulo pode conter várias estruturas, desde que o tamanho seja menor que 64 KBytes. Não é possível dividir um arquivo de dados em mais de um módulo. Dessa forma, arquivos com tamanho maior que 64 KBytes devem ser transportados em um único módulo. Esse é o único caso em que um módulo pode exceder o tamanho de 64 KBytes. Arquivos em um módulo podem vir de qualquer parte do sistema de diretórios, eles não têm a necessidade de pertencerem ao mesmo diretório. Cada módulo é quebrado, por sua vez, em blocos, como mostra a Figura 7, que são então transportados em Seções Privadas MPEG-2 [15]. Note, na figura, que as estruturas de metadados, mapa-de-eventos e arquivo de dados precisam ser de alguma forma encapsuladas em mensagens. No Ginga-NCL esse encapsulamento será o de mensagens BIOP [6], o mesmo usado nos carrosséis de objetos apresentados na próxima seção, ou em Seções NCL, conforme discutido na Seção 5.3.3.

5.3.2 Carrossel de Objetos

O carrossel de objetos DSM-CC [6] pode ser usado como alternativa para transmissão das estruturas. Essa foi a solução escolhida quando da adoção do Ginga-NCL pelo Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) [14]. O protocolo de carrossel de objetos DSM-CC permite a transmissão cíclica de objetos de evento e sistemas de arquivos.

Objetos de evento são utilizados para mapear nomes de eventos de fluxo em *ids* de eventos de fluxo, definidos pelos descritores de evento, cumprindo o mesmo papel das estruturas mapa-de-eventos. Assim, quando um comando de edição NCL precisa ser enviado, um objeto de eventos DSM-CC deve ser criado, mapeando a *string* “*nclEditingCommand*” em uma *id* de evento de fluxo. O objeto é então colocado em um carrossel de objetos DSM-CC e enviado em um fluxo elementar MPEG-2 TS [15]. Um ou mais descritores de evento de fluxo DSM-CC com a *id* previamente selecionada podem ser então criados e enviados em outro fluxo elementar MPEG-2 TS.

Carrosséis de objetos DSM-CC são também usados para o transporte de arquivos organizados em diretórios. Um demultiplexador DSM-CC é responsável por montar a estrutura recebida no dispositivo receptor. Parâmetros dos comandos de edição, especificados como documentos XML (documentos NCL ou nós NCL que se quer adicionar) podem, assim, ser organizados em sistemas de arquivos a serem transportados nos carrosséis.

Cada carrossel de objetos contém uma árvore de diretórios, que é quebrada em uma série de módulos, que podem conter um ou mais arquivos (objetos de arquivo) ou diretórios (objetos de diretórios e objetos Service Gateway). Um diretório contém o nome dos seus componentes filhos e ponteiros para localização do conteúdo dos mesmos no carrossel. Objetos Service Gateway (*srg*) representam um conceito similar a um diretório. A principal diferença é que um objeto Service Gateway identifica o diretório raiz da árvore de diretórios de um carrossel de objetos. Isso significa que existirá um e apenas um objeto Service Gateway em um carrossel de objetos. A localização do Service Gateway é transmitida em uma mensagem *DownloadServerInitiate* (DSI) [6] ao sistema receptor.

Note assim que a própria estrutura do carrossel de objetos representa a mesma estrutura de metadados (com a diferença única que a estrutura de metadados não perde a referência da raiz do diretório) definida na Seção 3.3, embora não a envelope literalmente. O conjunto de pares de referência transportado pelo descritor de evento de fluxo completa toda informação necessária para que o receptor consiga mapear as referências do documento da especificação XML (aplicação NCL ou especificação de um nó NCL) em seu sistema de armazenamento local. Nesse caso, a especificação XML deve ser enviada no mesmo carrossel de objetos que carrega o objeto de eventos. O parâmetro *cUri* do primeiro par de referências deve ter o caminho absoluto da especificação XML (o caminho no servidor de

dados). O parâmetro *clid* correspondente no par deve fazer referência ao IOR (endereço) da especificação XML (*carouselId*, *moduleId*, *objectKey*) [6] no carrossel de objetos. Se outros sistemas de arquivos precisarem ser transmitidos usando outros carrosséis de objeto a fim de completar o comando de edição (como usual nos comandos *addDocument* ou *addNode*) com conteúdo de mídia, outros pares {*cUri*, *clid*} devem estar presentes no comando. Nesse caso, o parâmetro *cUri* deve ter o caminho absoluto da raiz do sistema de arquivos (o caminho no servidor de transmissão de dados) e o respectivo parâmetro *clid* no par deve fazer referência ao IOR (*carouselId*, *moduleId*, *objectKey*) de qualquer arquivo ou diretório filho da raiz no carrossel de objetos (o *service gateway* do carrossel).

Voltando ao exemplo da Figura 1, a transmissão dos arquivos de dados, dos metadados e do mapa-de-eventos (objeto de evento) da aplicação NCL “O Primeiro João” seria realizada pelo carrossel da Figura 8.

ServiceDomain = 0x1	
moduleId = 0x1	moduleId = 0x2
...	...
objectKey = 0x1	objectKey = 0x1
objectKind = srg	objectKind = dir
2 bindings	4 bindings
binding #1	binding #1
objectName = applications	objectName = background.png
objectType = dir	objectType = fil
IOR = 0x1,0x1,0x2	IOR = 0x1,0x2,0x2
binding #2	binding #2
objectName = mediaGar	objectName = soccercon.png
objectType = dir	objectType = fil
IOR = 0x1,0x2,0x1	IOR = 0x1,0x2,0x3
...	binding #3
objectKey = 0x2	objectName = soccerAdv.mp4
objectKind = dir	objectType = fil
1 binding	IOR = 0x1,0x2,0x4
binding #1	binding #4
objectName = primeiroJoao.ncl	objectName = form.htm
objectType = fil	objectType = fil
IOR = 0x1,0x1,0x3	IOR = 0x1,0x2,0x5
...	...
objectKey = 0x3	objectKey = 0x2
objectKind = fil	objectKind = fil
data	data
...	objectKey = 0x3
objectKey = 0x4	objectKind = fil
objectKind = ste	data
eventList	data
eventName =	objectKey = 0x4
"nclEditingCommand"	objectKind = fil
eventId = 0x3	data

Figura 8. Carrossel de objetos para Figura 2.

No carrossel, dois módulos são gerados e identificados com valores “0x1” e “0x2”. O identificador de cada objeto é apresentado através do campo “objectKey”. O objeto (do tipo “srg”) que representa o *Service Gateway* é identificado com o valor “0x1” e é encapsulado no Módulo “0x1”. Como consequência, sua IOR deve ser transmitida na mensagem *DownloadServerInitiate* definida como “0x1,0x1,0x1” (Service Domain=0x1, id do módulo=0x1 e id do objeto=0x1). Os objetos que representam o diretório “applications” e o arquivo “primeiroJoao.ncl” são identificados pelos valores “0x2” e “0x3”, respectivamente, e também são encapsulados no Módulo “0x1”. Já os objetos que representam o diretório “mediaGar” e os arquivos com os conteúdos de seus filhos são identificados com os valores “0x1”, “0x2”, “0x3”, “0x4” e “0x5”, respectivamente, mas encapsulados no Módulo “0x2”. Ainda no Módulo 1, um objeto de eventos (na Figura 8 o objeto do tipo “ste”) mapeia a string “nclEditingCommand” no identificador “0x3” de evento.

Ainda no exemplo, um descritor de evento de fluxo deve ser transmitido com o valor de *eventId* apropriado, no exemplo “0x3”, e o valor “0x05” no *commandTag*, indicando um comando *addDocument*. O parâmetro *cUri* conterá o esquema (no caso do SBTVD, “x-sbtvd”, indicando, opcionalmente, o recebimento no carrossel) e o caminho absoluto do documento NCL (“C:\nclRepository\applications”, de acordo com a Figura 2). Finalmente, a IOR do documento NCL no carrossel de objetos é transportada no parâmetro *clid* (*carouselId* = 0x1, *moduleId* = 0x1, *objectKey* = 0x3).

5.3.3 Seções NCL

Seções NCL nos permitem transmitir as três estruturas de dados anteriormente definidas: mapas, metadados e arquivo de dados. Cada Seção NCL pode conter os dados de apenas uma estrutura. Contudo, uma estrutura pode se estender por várias Seções. Estruturas de dados podem ser transmitidas em qualquer ordem e quantas vezes forem necessárias.

Seções NCL podem ser encapsuladas em carrosséis de dados, como indica a Figura 7. Nesse caso, o carrossel é usado apenas para a transmissão cíclica das estruturas, a um custo grande de overhead de encapsulamento.

Seções NCL podem, alternativamente, ser encapsuladas diretamente em um tipo específico de Seção MPEG-2 (identificado por um valor do campo *table_id* de uma seção privada [15]), diminuindo o overhead de encapsulamento. Cada Seção MPEG-2 pode conter apenas uma Seção NCL.

Seções NCL podem também ser encapsuladas em outras estruturas de dados. Por exemplo, o encapsulamento MPE (*Multi-Protocol Encapsulation*) [5] pode ser usado. Nesse caso, Seções NCL seriam Seções MPEG-2 de datagrama. Elas podem ainda ser envelopadas em outro formato de dados de protocolo que não MPEG-2 System, como, por exemplo, pacotes FLUTE [10].

O primeiro byte do cabeçalho de uma Seção NCL identifica o tipo da estrutura transportada (0x01 para metadados; 0x02 para arquivos de dados, e 0x03 para mapa-de-eventos). O segundo byte carrega um identificador único da estrutura (*structureId*) no fluxo de transporte. O fluxo e o identificador da estrutura são aqueles que devem ser associados pela estrutura de metadados, através dos atributos *component_tag* de um serviço e *structureId* dos elementos <pushedRoot> e <pushedData>, a localizadores de arquivos (URL).

Depois do segundo byte, vem uma estrutura de dados serializada, que pode ser a *mappingStructure* (como ilustrado pela Figura 5), ou a estrutura de metadados (um documento XML, conforme Schema XML introduzido na Seção 4.3), ou uma estrutura de arquivo de dados (um conteúdo de arquivo serializado). O demultiplexador de Seções NCL é responsável por montar toda a estrutura da aplicação no dispositivo receptor.

Quando Seções NCL são utilizadas, o campo *privateDataPayload* do descritor de evento (vide Figura 3) deve carregar o par de referências {*cUri*, *clid*}. No entanto, nesse caso, os parâmetros *cUri* têm sempre o valor “null”. No caso dos comandos *addDocument* e *addNode*, o parâmetro *clid* do primeiro par deve identificar o fluxo elementar (“*component_tag*”) de um serviço e a estrutura de metadados que ele transporta (“*structureId*”). A estrutura de metadados, por sua vez, contém o caminho absoluto do documento NCL ou da especificação do nó NCL (o caminho no servidor de dados) e a estrutura arquivos de dados relacionada (“*structureId*”) transportada em Seções NCL do mesmo fluxo elementar. Se outras estruturas de metadados adicionais forem necessárias para completar os comandos de edição *addDocument* ou *addNode*, outros pares {*cUri*, *clid*} devem se fazer presentes no comando. Nesse caso, os parâmetros *cUri* devem também ter o valor “null” e os parâmetros *clid* correspondentes devem referir-se ao *component_tag* e à estrutura de metadados transportada (*structureId*) correspondente.

No caso de se usar Seções privadas MPEG-2 para transporte direto, o começo da Seção será delimitado pelo campo *payload_unit_start_indicator* de um pacote TS [15]. Depois dos quatro bytes do cabeçalho TS, a carga (payload) do pacote TS começa, com um campo ponteiro de um byte indicando o início de uma Seção NCL [15]. No mesmo fluxo elementar que carrega a especificação XML (o arquivo do documento NCL ou o documento XML de especificação de um nó NCL) é recomendado que uma estrutura mapa-de-eventos seja transmitida, para que seja mapeado o nome

“*nclEditingCommand*” no *eventId* do descritor de evento que transportará os comandos de edição.

A Figura 9 ilustra o descritor de eventos e o mapa-de-eventos para a aplicação NCL do Exemplo 1, transportados por meio de Seções NCL. A estrutura de metadados enviada é a mesma da Figura 6. Pela figura, um fluxo elementar MPEG-2 do serviço 0x01 (*component_tag* = “0x01.0x09”) é gerado, carregando todo o sistema de arquivos do programa interativo (o arquivo NCL e todos os arquivos de conteúdo de mídia, conforme a Figura 2). O mapa-de-eventos criado (*structureType*=“0x03”; *structureId*=“0x0C”, na Figura 9), mapeia o nome “*nclEditingCommand*” ao valor de *eventId* (valor “0x03”, Figura 9). O descritor de evento define para *eventId* o valor “0x03”, e para o *commandTag* o valor “0x05”, que indica um comando *addDocument* [1]. O parâmetro *cUri* tem o valor “null” e o parâmetro *cid* o valor (*component_tag*= “0x01.0x09”, *structureId*= “0x0A”).

Descritor de Eventos	Mapa de Eventos
<pre>eventId = 0x03 eventNPT = 0 privateDataLength = dataLen() commandTag = 0x05 sequenceNumber = 0 finalFlag = 1 privateDataPayload = "TV Ginga", "NULL", "0x01.0x09", "0x0A" FCS = checksum()</pre>	<pre>eventId = 0x03 eventNameLength = 0x11 eventName = nclEditingCommand</pre>

Figura 9. O primeiro Joao em Seções MPEG-2.

6. Considerações Finais

O oferecimento de serviços de transporte assíncronos em sistemas de TV digital permite a transmissão de dados e aplicações interativas em modo *push*, ou seja, por vontade do provedor de serviços, sem a solicitação do usuário. Por sua vez, o tratamento de eventos assíncronos pelo middleware habilita o controle do ciclo de vida de uma aplicação e, mais ainda, a modificação de seu comportamento durante sua apresentação nos receptores de TV.

Este artigo apresenta as contribuições dos autores à especificação Ginga-NCL para a definição de uma ampla infra-estrutura de suporte a serviços de transporte assíncronos de forma extensível e independente do sistema de distribuição de TV digital. Os requisitos para tal suporte são atendidos por meio de estruturas de dados e metadados que podem ser adequados à codificação de transporte específica de cada sistema de distribuição, conforme também demonstrado neste artigo. Além disso, a introdução desses metadados pelos provedores de serviços isenta o autor de uma aplicação NCL da preocupação sobre as localizações das mídias tal qual codificadas em seu armazenamento e transporte.

As contribuições apresentadas e especializadas para o SBTVD encontram-se presentes na implementação de referência do middleware Ginga-NCL, disponível para download sob licença livre na Comunidade Ginga (www.softwarepublico.gov.br). Essa implementação de referência possui uma arquitetura baseada em componentes e, por isso, o suporte a novas funcionalidades ou especificidades de ambiente pode ser facilmente agregado. Por exemplo, a especialização do componente de codificação de dados para outros sistemas de transporte além das normas do SBTVD poderia ser rapidamente desenvolvida.

Mais especificamente, o desenvolvimento de novos componentes para a especialização da implementação Ginga-NCL para ambientes IPTV está em andamento, buscando a conformidade com a recomendação ITU-T H.761 [3]. Os sistemas de transporte oferecidos serão o Carrossel de Objetos DSM-CC e o protocolo FLUTE [10]. De fato, FLUTE é um protocolo de transporte para aplicações e dados mais direcionado ao ambiente IPTV, uma vez que oferece pronto suporte a comunicação multicast, independe da existência de multiplexação dos dados com o fluxo audiovisual principal, além de possuir mecanismos para tolerância a congestionamento de rede e correção de erros.

A partir da especialização da codificação de dados para o transporte em IPTV, pode-se vislumbrar novas frentes de trabalho, como a adequação desse transporte em serviços peer-to-peer TV (P2PTV) [17]. Nesse contexto, não somente o fluxo audiovisual principal de um canal P2PTV pode estar segmentado (e de alguma forma redundante) em diversos pontos da rede, mas também as aplicações interativas. O tratamento de tal distribuição granular do conteúdo interativo compreende um interessante trabalho futuro.

7. REFERÊNCIAS

- [1] ABNT NBR 15606-2 Associação Brasileira de Normas Técnicas. Digital Terrestrial Television Standard 06: Data Codification and Transmission Specifications for Digital Broadcasting, Part 2 – GINGA-NCL: XML Application Language for Application Coding (São Paulo, SP, Brazil, November, 2007). http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Ing_2008.pdf.
- [2] ITU-R Recommendation BT 1699-1, 2009. Harmonization of declarative application formats for interactive TV. Geneva, January, 2009.
- [3] ITU-T Recommendation H.761, 2009. Nested Context Language (NCL) and Ginga-NCL for IPTV Services. Geneva, April, 2009.
- [4] Soares L.F.G., Rodrigues R.F. 2006. Nested Context Language 3.0 Part 8 – NCL Digital TV Profiles. Technical Report. Departamento de Informática da PUC-Rio, MCC 35/06. <http://www.ncl.org.br/documentos/NCL3.0-DTV.pdf>.
- [5] Morris, S. and Chaigneau, A. S. (2005). Interactive TV Standards. Focal Press, Elsevier.
- [6] ISO/IEC 13818-6, “Information technology — Generic coding of moving pictures and associated audio information: Digital Storage Media Command & Control”, 1996, ISO/IEC.
- [7] Moreno, M.F., Rodrigues, R. F., Soares, L.F.G. 2007. A Resource Identification Mechanism for Interactive DTV Systems. Workshop on New Techniques for Consuming, Managing, and Manipulating Interactive Digital Media at Home (ISM 2007) - CMMIDMH, p. 215-220.
- [8] ARIB Association of Radio Industries and Business. 2004. ARIB Standard B-24 Data Coding and Transmission Specifications for Digital Broadcasting, version 4.0, 2004.
- [9] RFC 3550, Standard 64, RTP : A Transport Protocol for Real-Time Applications.
- [10] RFC 3926, FLUTE: File Delivery over Unidirectional Transport.
- [11] IP Datcast over DVB-H: Content Delivery Protocols (CDP) - A101r1 (dTS 102 472 v1.3.1)
- [12] 3GPP TS 26.346 - Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs.
- [13] Costa R.M.R ET AL. 2006. Live Editing of Hypermedia Documents. In Proceedings of ACM Symposium on Document Engineering. DocEng 2006.
- [14] ABNT NBR 15606-3. Associação Brasileira de Normas Técnicas. 2007. Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital. Parte 3: Especificação de transmissão de dados.
- [15] ISO/IEC 13818-1, “Information technology — Generic coding of moving pictures and associated audio information: Systems”, 1996, ISO/IEC.
- [16] ATSC Advanced Television Systems Committee. 2000. ATSC Data Broadcasting Standard - A/90, 2000.
- [17] Video Over IP, Second Edition: IPTV, Internet Video, H.264, P2P, Web TV, and Streaming: A Complete Guide to Understanding the Technology by Wes Simpson (Kindle Edition - Aug 8, 2008).