Uma Arquitetura Flexível de Suporte à Execução de Algoritmos de Recomendação de Programas em TV Conectada

João R. dos Santos Júnior
Centro de Informática – UFPE
Caixa Postal 7851 – 50732-970
Recife – PE – Brasil
jrsj2@cin.ufpe.br

Carlos André G. Ferraz
Centro de Informática – UFPE
Caixa Postal 7851 – 50732-970
Recife – PE – Brasil
cagf@cin.ufpe.br

RESUMO
Embora seja atrativa, a grande quantidade de programas de TV gera sobrecarga de informação. Uma solução para esse problema é o uso de sistemas de recomendação, cujo objetivo é sugerir programas a partir das preferências do telespectador. Muitos sistemas têm sido propostos, porém a maioria possui uma arquitetura inflexível por ser embarcada em *set-top boxes*, além de demandar seus recursos de hardware. Este artigo propõe uma arquitetura flexível na nuvem para suportar a execução de algoritmos de recomendação de programas. Com a convergência entre TV e Internet, a TV Conectada surgiu como uma opção de consumo de novos serviços interativos. Assim, um sistema construído sobre essa arquitetura é disponibilizado como serviço, onde diversos algoritmos podem ser executados, desacoplando-os do receptor. Para validar a flexibilidade da arquitetura foram implementados dois algoritmos de recomendação e uma aplicação cliente de TV Conectada.

ABSTRACT
Although attractive, the large amount of TV programs creates information overload. One of the solutions is the use of recommender systems, which aim is to suggest programs according to the viewer's preferences. Several systems have been proposed, but most have an inflexible architecture due to being embedded in receivers and consume their hardware resources. This article proposes a flexible architecture in a cloud to support the execution of programs recommendation algorithms. With the convergence between TV and Internet, Connected TV has emerged as a consume option of new interactive services. Thus, a system built over the proposed architecture is available as a service, where different algorithms can be run, disengaging them from the viewer receiver. In order to validate the flexibility of the architecture, two recommendation algorithms and a Connected TV client application were implemented.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: System and Software – Distributed systems.

General Terms
Algorithms, Design, Performance

Keywords
Cloud Computing, Connected TV, Recommender Systems.

WebMedia'11: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web. Full Papers.
October 3 -6, 2011, Florianópolis, SC, Brazil.
ISSN 2175-9642.
SBC - Brazilian Computer Society

1. INTRODUÇÃO
Entre tantas novidades advindas da TV Digital, como as altas qualidades de vídeo e áudio, o aumento do número de programas também se destaca. Empresas de TV a cabo divulgam seus serviços por meio do número de canais. Ao mesmo tempo em que esta diversidade de conteúdo é um atrativo para os telespectadores, ela gera o problema da sobrecarga de informação. Uma consequência disso é a perda de tempo do telespectador enquanto troca de canal até encontrar o programa que gostaria de assistir.

Uma solução para lidar com o grande volume de informações é o uso de sistemas de recomendação. Basicamente, tais sistemas filtram um conjunto de itens de acordo com os interesses do usuário. Algumas técnicas são aplicadas no processo de recomendação, como filtragem de conteúdo e filtragem colaborativa [1], utilizando diversos tipos de algoritmos. A maior parte dos sistemas de recomendação usa apenas uma técnica. De modo oposto aos sistemas híbridos, que combinam mais de uma dessas técnicas. Segundo [2], os sistemas híbridos produzem recomendações mais variadas, ou seja, tendem a sugerir itens diferentes a cada interação do usuário. Contudo, eles demandam mais recursos de hardware, pois possuem um fluxo de atividade maior que um sistema com apenas uma técnica.

No cenário da TV Digital, os dispositivos receptores possuem recursos computacionais limitados. Especificamente, eles dão o suporte básico para o middleware e suas aplicações clientes. Quando se considera a execução de um sistema de recomendação embarcado nos receptores, os recursos de hardware são limitados. Ainda assim, mesmo supondo que haja recursos necessários, o sistema tende a ser inflexível, pois ele emprega, em geral, uma técnica de recomendação de modo embarcado.

Com a crescente convergência entre a Internet e a TV, a TV Conectada tem adquirido cada vez mais espaço nos lares [3]. As TVs Conectadas possuem conexão constante com a Internet e permitem o acesso a atividades equivalentes de um usuário conectado, como ler informações em tópicos de interesse, acessar e-mails e entre outras. Tais atividades são controladas por meio de

widgets [3], que, neste contexto, são aplicações independentes de um middleware de TV Digital, como Ginga ou MHP (*Multimedia Home Platform*). Alguns fabricantes de software disponibilizam *widgets* através de lojas online, possibilitando facilmente o seu uso para os telespectadores.

É diante desses dois problemas que este trabalho se encaixa. O processo de recomendação é feito sobre uma arquitetura flexível em uma nuvem computacional, onde diversos algoritmos de recomendação podem ser executados. Seguindo esta ideia, um sistema de recomendação construído seguindo a arquitetura proposta é disponibilizado como serviço em uma nuvem computacional. O lado cliente da arquitetura deve ser uma aplicação interativa de TV Conectada.

O restante do trabalho está estruturado da seguinte forma: logo a seguir, na Seção 2, é feito um levantamento dos trabalhos relacionados. Na Seção 3, a arquitetura proposta é detalhada. Na Seção 4, são apresentadas uma implementação da arquitetura, bem como uma aplicação cliente de TV Conectada. A validação da arquitetura é descrita na Seção 5. Por fim, na Seção 6, é feita a conclusão do trabalho, mencionando alguns trabalhos futuros.

2. TRABALHOS RELACIONADOS

É comum encontrar na literatura muitas propostas de algoritmos de recomendação ou de modelagem do comportamento do telespectador. Porém, trabalhos que abordem infraestruturas ou arquiteturas para esses sistemas são encontrados em menor número.

Um desses trabalhos é o PersonalTVware [4], onde é proposta uma infraestrutura de suporte a sistemas de recomendação sensíveis a contexto. A arquitetura cliente-servidor do PersonalTVware inclui dois subsistemas: Dispositivo do Usuário e Provedor de Serviços. O Dispositivo do Usuário levanta informações contextuais do telespectador. Já o servidor é responsável por todo o processo de recomendação baseado em filtragem de conteúdo, utilizando métodos de aprendizagem de máquina. Embora implemente mais de um método de aprendizagem, não é especificado se há alguma flexibilidade em termos de execução. Além disso, o Provedor de Serviços não tem escalabilidade.

Em [5], Lee et al. propõem um sistema de recomendação usando três nuvens computacionais. A primeira mantém uma base de dados dos conteúdos multimídia transmitidos por diversos canais de TV. Já a segunda nuvem possui outra base de dados que armazena os perfis dos telespectadores. Por fim, a terceira nuvem implementa um sistema de recomendação baseado em filtragem de conteúdo. Muito embora o sistema proposto por Lee et al. [5] seja escalável, apenas um algoritmo de recomendação é utilizado.

Em [6], é proposto um sistema de recomendação que utiliza conceitos da Web Semântica, chamado AVATAR. Sobre uma arquitetura cliente-servidor, o sistema permite a execução de algoritmos de recomendação, porém, de forma diferente desse trabalho, é definido apenas um critério de execução desses algoritmos. Vale ressaltar que o lado cliente do AVATAR deve ser um receptor com o middleware MHP. Como dito na Seção 1, o lado cliente da arquitetura proposta neste trabalho utiliza TV Conectada, que é independente de middleware e facilmente disponibilizado para o telespectador.

3. ARQUITETURA

A arquitetura proposta é ilustrada na Figura 1. Ela é constituída por quatro componentes principais: *Client Request Handler*, *Profile Interpreter*, *Instances Manager* e *Recommender*.

O primeiro, *Client Request Handler*, recebe preferências e envia sugestões de programas para o telespectador que solicitou recomendações na aplicação cliente. O *Profile Interpreter* é responsável pelo *parsing* das preferências do telespectador e, posteriormente, por dispô-las adequadamente para o componente *Recommender*. O perfil do telespectador é composto pela lista de programas disponíveis no EPG (*Electronic Programming Guide*) no momento da requisição e pela lista dos programas preferidos. Os programas preferidos são definidos a critério da aplicação cliente, a partir da avaliação do telespectador, seja ela feita de forma explícita (intrusiva) ou implícita. Os perfis dos usuários são armazenados na base de dados *Profile Base* para futuras recomendações. Assume-se que a privacidade desses dados seja respeitada.

Como a arquitetura se propõe a suportar a execução de algoritmos de recomendação, faz-se necessário coletar metadados dos programas a serem sugeridos. Definiu-se, portanto, que o tipo de metadado manipulado pela arquitetura seria a ontologia de programas de televisão da BBC (*British Broadcast Corporation*) [7], chamada *Programme Ontology*. A motivação para o uso de uma ontologia é a sua fácil representação de entidades de um domínio. A partir de então, as instâncias citadas durante o texto referem-se às instâncias dessa ontologia.

Cada programa contido no perfil do telespectador deve ter uma instância correspondente. Durante o *parsing*, o *Profile Interpreter* requisita o *Instances Manager* que, por sua vez, procura e retorna as respectivas instâncias de programas. O *Instances Manager* realiza a busca em dois locais: na base local de instâncias, *Local Instances*, e nos serviços de metadados, por meio do componente *Instances Collector*.

Nesta arquitetura, cada canal deve possuir um serviço de metadados, onde o *Instances Manager* possa requisitar instâncias por meio de mensagens *request/response*[1]. Os serviços de metadados são externos à arquitetura e tem uma base de dados única, contendo todos os programas exibidos pelo canal. Para encontrar uma determinada instância, primeiramente, o componente *Instances Manager* procura na base de dados *Local Instances*. Caso ela não exista nesta base, então o componente *Instances Collector* requisitar-lhe-á no serviço de metadados cujo canal possui o programa procurado. Por fim, a instância retornada será armazenada na base *Local Instances*.

O último componente descrito é o núcleo da proposta, o *Recommender*. Seu objetivo é executar algoritmos de recomendação sobre uma estrutura flexível. O *Recommender* mantém a lista de algoritmos que podem ser executados. Se existir apenas um algoritmo na lista, então o resultado das sugestões é direto. Caso contrário, um método de combinação de algoritmos, ou método de hibridização, é escolhido pelo subcomponente *Hybridization Selector*. Em [2], pode ser encontrada uma análise de sete desses métodos.

[1] Esta é a mesma estrutura de buscas de instâncias adotada pela BCC. Vide: http://www.bbc.co.uk/programmes.

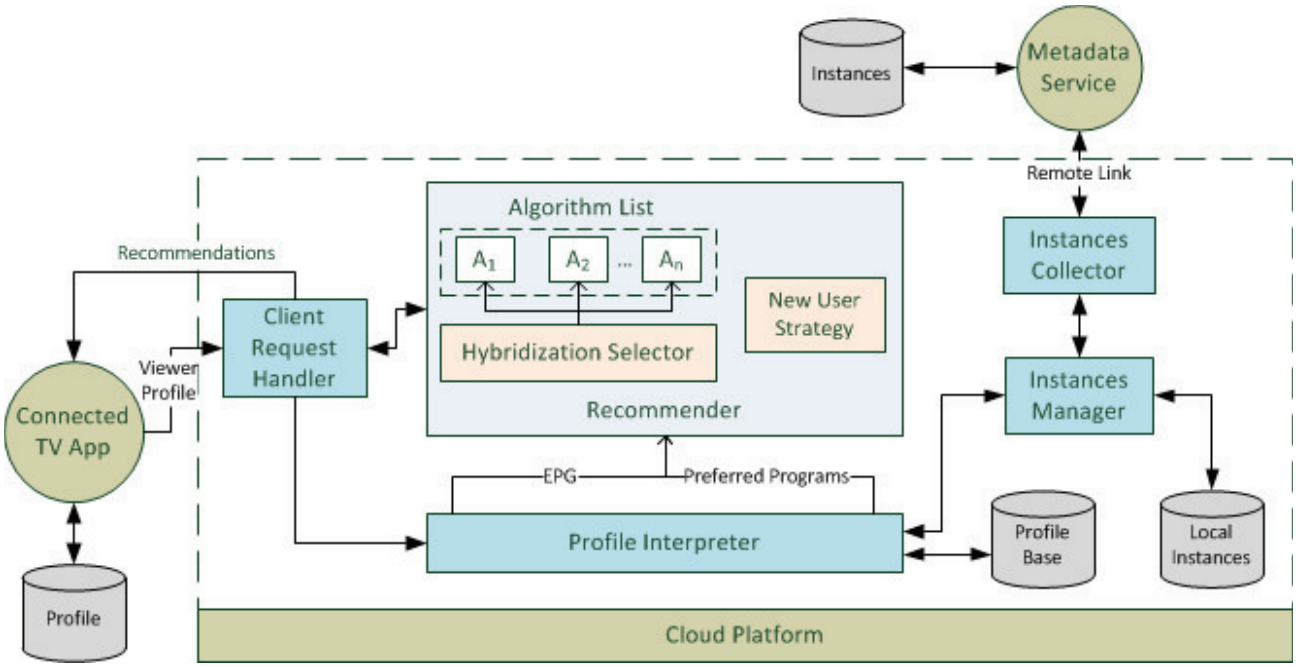

Figura 1: Arquitetura flexível para a execução de algoritmos de recomendação de programas de TV.

Embora seja externa à arquitetura, a aplicação cliente de TV Conectada tem a especificação de armazenar as preferências do telespectador na base de dados *Profile*. Sempre que o usuário requisitar recomendações, o seu perfil será enviado para o serviço de recomendação.

Uma característica essencial da arquitetura é ser escalável, uma vez que deve estar disponível na nuvem. Sabendo-se que o número de televisores é imenso, então um sistema de recomendação que utilize essa arquitetura deve prover escalabilidade para lidar com muitas requisições. Além disso, como uma nuvem computacional tem grande capacidade de processamento, os algoritmos de recomendação podem ser executados sem as limitações dos dispositivos receptores de TV.

4. IMPLEMENTAÇÃO

Um sistema de recomendação foi implementado seguindo a arquitetura proposta e é modelado pelos diagramas de classes UML das Figura 2 e Figura 3. Cinco pacotes principais compõem o sistema: *domain*, *persistence*, *metadata*, *algorithm* e *service*. Logo a seguir, eles são descritos.

O pacote *domain* (vide Figura 2) inclui quatro classes que representam as entidades de domínio do sistema: *Channel*, *Program*, *Client* e *Profile*. A classe *Channel* representa um canal de TV e, através do atributo *serviceURI*, permite o acesso ao seu serviço de metadados. A classe de domínio *Client* tem como principal atributo *hash*, que corresponde a um identificador único do telespectador. Sempre que um telespectador requisitar recomendações, ele é identificado por meio do atributo *hash*. Já a classe *Program* corresponde a um programa de TV. Nesta implementação, apenas alguns atributos foram criados na classe *Program*. Porém, o ideal é que ela contenha todas as propriedades da ontologia *Programme Ontology* [7]. Por fim, tem-se a classe *Profile* que representa, através do atributo *rating*, o nível de interesse de um *Client* em um determinado *Program*. Como antes mencionado, a computação do nível de interesse é de responsabilidade da aplicação cliente de TV Conectada.

Ainda seguindo o diagrama da Figura 2, o objetivo das classes do pacote *persistence* é prover métodos capazes de gerenciar o acesso aos objetos de domínio do sistema. Em outras palavras, as classes *ChannelDAO*, *ClientDAO*, *ProgramDAO* e *ProfileDAO* formam os DAOs (*Data Access Objects*) do sistema implementado.

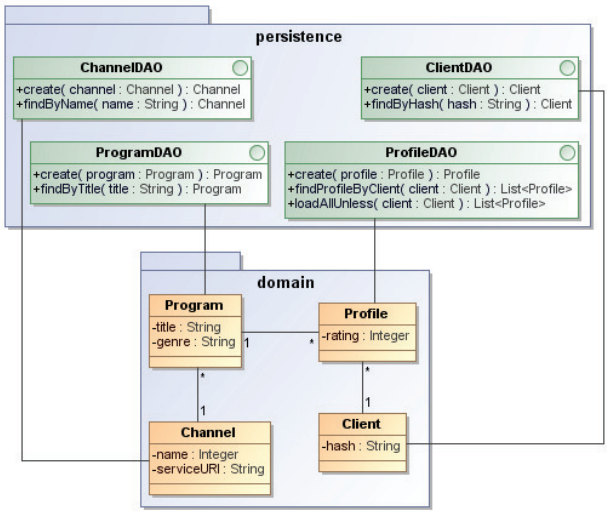

Figura 2. Classes de domínio e DAOs.

Na Figura 3, o pacote *metadata* engloba a classe *OntologyServiceHandler* que representa o componente *Instances Collector* e é responsável por buscar instâncias da ontologia de programas. Dados o canal e o título do programa, o método *findOntologyInstanceForProgram* encontra a instância correspondente do programa no serviço de metadados do canal fornecido. Primeiramente, a instância é retornada pelo serviço no

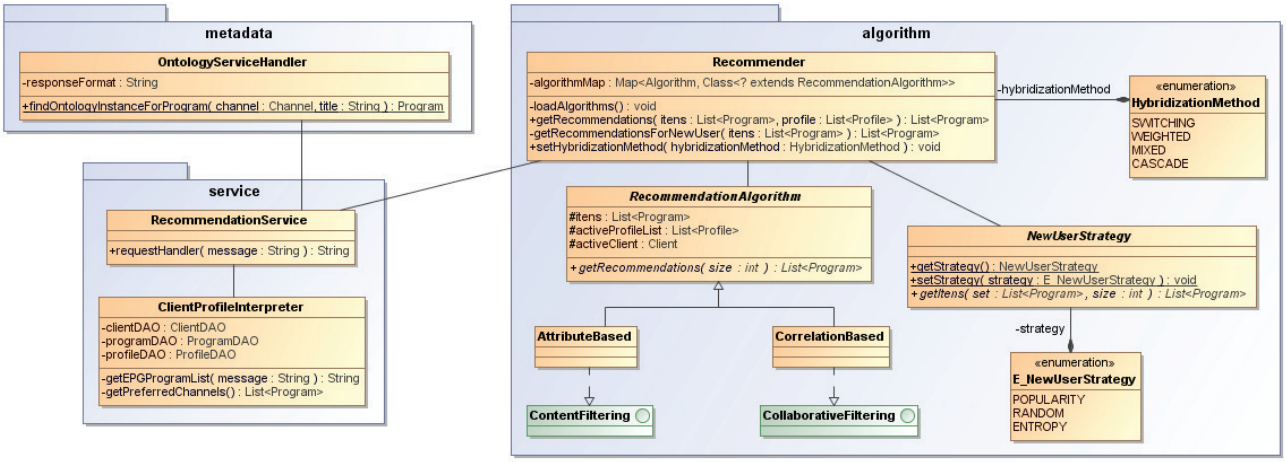

Figura 3. Diagrama de classes com os pacotes *metadata*, *service* e *algorithm*.

formato XML e, posteriormente, o método *findOntologyInstanceForProgram* retorna uma instância de *Program*.

O pacote *algorithm* inclui todas as classes necessárias para o processo de recomendação. A classe *Recommender* representa o sistema de recomendação em si. O seu atributo *algorithmMap* representa uma lista de algoritmos que podem ser executados no sistema. O método *loadAlgorithms* popula a lista *algorithmMap* com todas as classes que tem *RecommendationAlgorithm* como classe-mãe. É dessa forma que a flexibilidade da arquitetura funciona. Mais classes de algoritmos podem ser adicionadas facilmente à lista, bastando que herde a classe *RecommendationAlgorithm*.

Na implementação desse sistema de recomendação, foram criados dois algoritmos representados pelas classes *AttributedBased* e *CorrelationBased*. O primeiro algoritmo utiliza a técnica de filtragem de conteúdo e sugere programas a partir dos seus atributos, como gênero ou palavras-chave. Já o segundo, *CorrelationBased*, emprega a técnica de filtragem colaborativa e utiliza a correlação de Pearson [1] para aproximar perfis de dois telespectadores.

Uma vez com a lista *algorithmMap* populada, o método *getRecommendations* é chamado e retorna a lista de programas sugeridos. Durante a sua execução, *getRecommendations* leva em consideração o método de combinação de algoritmos de recomendação, definido no atributo *hybridizationMethod*. Nesta implementação, foi utilizado o método *switching* que executa algoritmos em sequência enquanto o número máximo de sugestões não for alcançado. Se nenhuma sugestão for retornada, um objeto da classe *NewUserStrategy* é invocado e tem a responsabilidade de gerar sugestões, seguindo alguma estratégia de seleção de programas do EPG do telespectador. Duas estratégias foram implementadas no sistema: *popularity* e *random*. A estratégia *popularity* seleciona itens de acordo com o número de votos recebidos. Já a estratégia *random* seleciona itens de forma aleatória.

Por último, o pacote *service* inclui as classes *RecommendationService* e *ClientProfileIntepreter*. Ambas correspondem, respectivamente, aos componentes da arquitetura *Client Request Handler* e *Profile Interpreter*. A primeira consiste em um web service do tipo *RestFul*, que recebe o perfil do telespectador e responde com a lista de sugestões de programas. Ambas as mensagens são manipuladas no formato XML. Quando uma requisição de recomendação chega ao sistema, o perfil é repassado para um objeto da classe *ClientProfileInterpreter*. Por sua vez, esse objeto transforma o perfil (em formato XML) em listas de instâncias da classe de domínio *Program*, seguindo exatamente o que faz o componente *Profile Interpreter*. A Figura 4Figura ilustra um exemplo de perfil.

```
<client-profile id="Client 1">
  <available-programs>
    <program title="Zorra Total"/>
    <program title="Grávida aos 16"/>
    <program title="Jornal da Clube"/>
    <program title="Bela, A Feia"/>
  </available-programs>
  <current-program title="Ponto de Luz" channel="TV Clube"/>
  <preferred-programs>
    <program title="A grande família" channel="Globo" rating="0.87"/>
    <program title="Bom Dia Vida" channel="TV Clube" rating="0.6"/>
    <program title="Clipes Comédia" channel="MTV" rating="0.76"/>
  </preferred-programs>
</client-profile>
```

Figura 4. Perfil de telespectador em formato XML.

Uma aplicação cliente de TV Conectada também foi criada. Em suma, assim que o telespectador abre a aplicação em sua TV, uma lista de programas sugeridos é mostrada na interface. A aplicação atualiza os seus programas preferidos enquanto estiver aberta. Para esta implementação do cliente, a definição de programas preferidos seguiu a seguinte condição matemática:

$$\alpha = T_v / T_t, \text{ onde } \alpha \geq 0.5,$$

onde T_v e T_t são equivalentes ao tempo de visualização e total de um programa, respectivamente.

O sistema de recomendação foi implementado utilizando a linguagem Java e roda sobre uma plataforma de nuvem computacional da Google, a GAE (*Google App Engine*) [9]. Por questões de segurança, a GAE não permite a criação de *sockets*, impossibilitando o uso de web services SOAP (*Service Oriented Architecture Protocol*) na aplicação. Para contornar essa

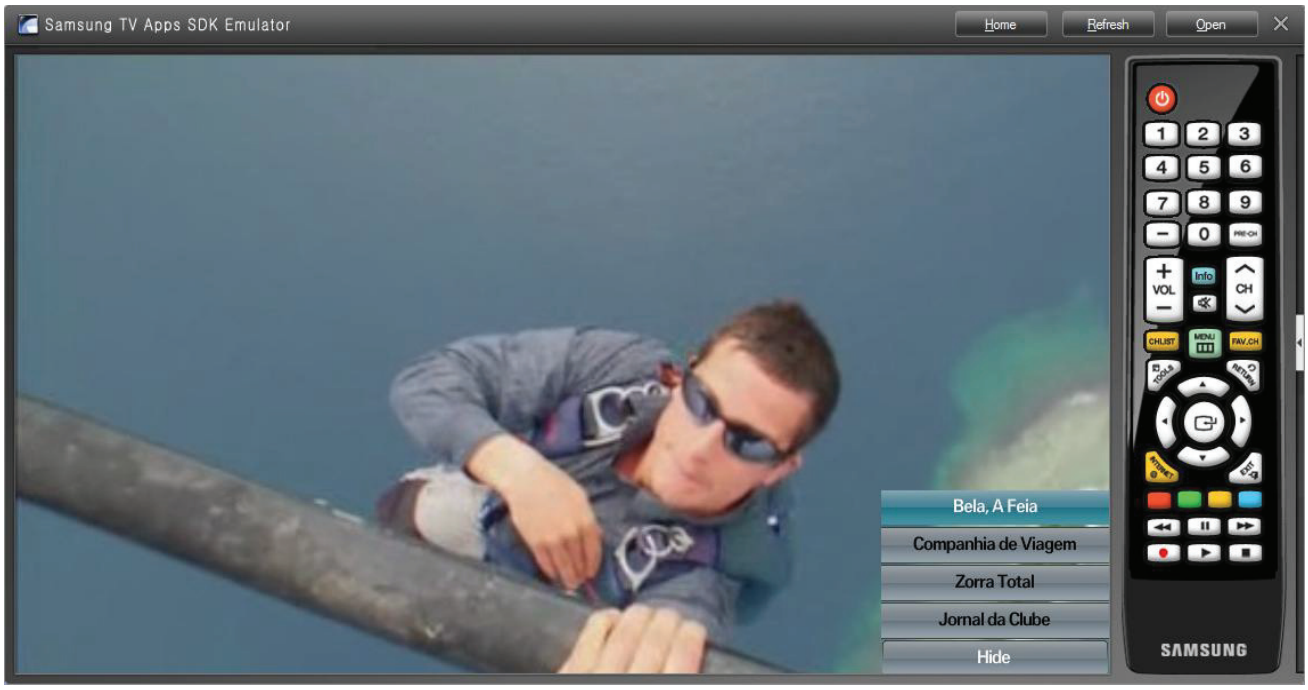

Figura 5. Aplicação Cliente de TV Conectada.

limitação, foi utilizado o framework RestLet [10] que permite o consumo de web services RestFul. O framework JPA (*Java Persistence Architecture*) [11] foi usado para a persistência dos objetos da classe de domínio. Para o *parsing* de mensagens XML, foi utilizado o *parser* SAX Parser [12]. Já a aplicação cliente foi desenvolvida através da ferramenta Samsung TV Apps SDK [13] (vide Figura 5), utilizando a linguagem Javascript.

5. VALIDAÇÃO DA ARQUITETURA

Foram criadas algumas instâncias da ontologia de programas de forma manual, tendo como base as programações semanais dos canais apresentados na Tabela 1. Bem como, a partir dessas instâncias, foram gerados perfis de telespectadores aleatoriamente (vide Tabela 2).

Tabela 1. Relação de instâncias de ontologia por canal.

Canal	Instâncias
Globo	76
TV Tribuna	80
RedeTV!	50
TV Clube	40
TV Cultura	81
MTV	39
Discovery	48
Fox	13
Foxlife	20
Home & Health	39

O critério para a validação da arquitetura foi analisar a quantidade de recomendações geradas a partir da execução de um grupo de algoritmos, definidos na implementação (vide Seção 4). Primeiramente, foram analisados os algoritmos implementados nas classes *AttributeBased* (AB) e *CorrelationBased* (CB) de forma individual. E depois, ambos algoritmos foram analisados a partir do método de seleção de algoritmos híbrida (HB), *switching*.

Os resultados deste experimento estão apresentados na Tabela 2. Lê-se a Tabela 2 da seguinte forma: o Cliente #3, com 10 programas no EPG e 2 programas preferidos (PP), recebeu 2 sugestões de programas do algoritmo AB, 3 sugestões do algoritmo CB e 5 sugestões do algoritmo HB.

Tabela 2. Número de sugestões geradas pelos algoritmos.

Cliente	EPG	PP	AB	CB	HB
#1	8	5	0	0	0
#2	10	0	1	2	3
#3	10	2	2	3	5
#4	9	6	2	1	3
#5	7	1	1	2	3
#6	9	5	1	0	1
#7	6	1	4	1	5
#8	10	4	2	0	2
#9	9	0	5	1	6
#10	8	6	2	0	2

No experimento realizado, observou-se que, embora o espaço amostral (número de instâncias) tenha sido pequeno, comprovou-se que o algoritmo de filtragem colaborativa, ao menos inicialmente, não gera mais recomendações que o algoritmo de filtragem de conteúdo em média. Porém quando se aplica a estratégia de recomendação híbrida, esse o número de sugestões aumenta.

A partir desse experimento, conclui-se que, quanto mais algoritmos de recomendação forem executados, a variedade de sugestões aumenta, comparando-se à execução de apenas um algoritmo. Dessa forma, é foi possível demonstrar a flexibilidade da arquitetura proposta.

6. CONCLUSÃO

Neste artigo, foi apresentada um arquitetura de suporte à execução de algoritmos de recomendação de programas de TV. A partir de uma implementação da arquitetura proposta, foi possível levantar alguns números sobre a quantidade de recomendações. Embora o número de instâncias criadas da ontologia de programas tenha sido pouco, foi possível concluir que a arquitetura proposta consegue executar algoritmos de recomendação de forma flexível e produzir sugestões mais diversificadas.

Como trabalhos futuros para o presente trabalho, considera-se colher mais resultados a partir do aumento do número de instâncias de programas. Além disso, é importante mensurar a qualidade das recomendações por meio das métricas de precisão, cobertura e medida F [8].

7. REFERÊNCIAS

[1] Adomavicuis, G. and Tuzhilin, A. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. In *IEEE Transactions on knowlegde and data engineering*, v. 17, n. 6, pp. 734-749.

[2] Burke, R. 2002. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, v. 12, n. 4, pp. 331-370.

[3] Montpetit, M.-J., Klym, N., and Blain, E. 2010. The Future of Mobile TV: When Mobile TV Meets the Internet and Social Networking. In *A. Marcus, A. C. Roibás, and R. Sala, editors, Mobile TV: Customizing Content and Experience, Human-Computer Interaction Series*, pp. 305–326. Springer London.

[4] Silva, F. S, Alves, L. G. P., Bressan, G. 2009. PersonalTVware: Uma Proposta de Infraestrutura de suporte a Sistemas de Recomendação Sensíveis ao Contexto para TV Digital Interativa.

[5] Lee, S., Deaho, L., Sungwon, L. 2010. Personalized DTV Program Recommendation System under a Cloud Computing Environment. In *IEEE Transactions on Consumer Electronics*. v. 56, n. 2, pp. 1034-1042.

[6] Blanco, Y. et al. 2004. AVATAR: Advanced Telematic Search of Audiovisual Contents by Semantic Reasoning. In *Proceedings of the Personalization of the Future TV Workshop*.

[7] BBC, The Programmes Ontology – making BBC programme metadata accessible to all. Disponível em: http://bbc.co.uk/programmes.

[8] Jiangshan, X. et al. 2002. The Development and Prospect of Personalized TV Program Recommendation Systems. In *Proceedings of the IEEE 4th Internation Symposium on Multimedia Software Engineering*.

[9] Google. 2011. Google Application Engine. Último acesso: 20/05/2011. Disponível em: http://appengine.google.com.

[10] Restlet. 2011. RESTful web framework for Java. Último acesso: 10/05/2011. Disponível em: http://www.restlet.org.

[11] Oracle. 2011. *Java Persistence Architecture*. Último acesso: 10/05/2011. Disponível em: http://jpa.java.com.

[12] Oracle. 2011. SAX Parser. Último acesso: 20/05/2011. Disponível em: http://download.oracle.com/javase/6/docs/api/ javax/xml/.

[13] Samsung. 2011. Samsung TV Apps SDK. Último acesso: 10/05/2011. Disponível em: http://www.samsungdforum.com.