GingaCDN: Uma Rede de Desenvolvimento Distribuído para o Middleware Ginga

Pierre Araújo, Marcel de Souza, Thales Ferreira, Caio Caroca,
Álan Lívio, Jônatas P. C. de Araujo, Raoni Kulesza, Guido L. de S. Filho
LAVID/Laboratório de Aplicações de Vídeo Digital
Departamento de Informática – UFPB
Campus I – Cidade Universitária
João Pessoa/PB – 58059-900
{pierre, marcel, thales, caio, alan, jonatas, raoni, guido}@lavid.ufpb.br

ABSTRACT
A Ginga *Code Development Network* ou simplesmente GingaCDN é uma rede de desenvolvedores de componentes e aplicações para o *middleware* brasileiro de TV Digital Ginga. Esta iniciativa faz parte do programa Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC), atualmente incubado pela Rede Nacional de Ensino e Pesquisa (RNP), criado pelo governo federal com o objetivo de desenvolver a competência nacional para inovação em comunicações digitais. Este artigo procura descrever a experiência de desenvolvimento distribuído e colaborativo utilizada no projeto através da definição de um modelo de processo de desenvolvimento e construção e uso de ferramentas específicas.

Keywords
TV Digital, GingaCDN, Middleware Ginga, DDS, DBC

1. INTRODUÇÃO
Nos últimos anos a opção por equipes de desenvolvimento distribuídas tem se tornado comum em organizações, empresas, grupos de pesquisa, etc. Esse movimento deve-se, em grande parte, à busca por redução de custos e tempo de desenvolvimento [4] [5] [6]. Levando-se em conta a TV Digital como uma área de surgimento recente e ainda em expansão, os custos tornam-se ainda maiores, uma vez que os profissionais capacitados são escassos, além não estarem necessariamente localizados na mesma região geográfica.

Nesse cenário, o Desenvolvimento Distribuído de Software (DDS) e o Desenvolvimento Baseado em Componentes (DBC) destacam-se por apresentar abordagem que permitem a redução de custos e tempo de desenvolvimento, além de possibilitar a melhoria e qualidade do software produzido [7], além da obtenção de recursos em localidades distintas [8]. A relação existente entre essas duas abordagens é apresentada por [9] como uma forma de mercado, onde o produtor fornece componentes de software para diversos consumidores e um consumidor pode adquirir componentes de software de diversos produtores, estando ambos geograficamente dispersos ou não.

Entretanto, essas abordagens apresentam limitações, que quando trabalhadas em conjunto aumentam o desafio de obter sucesso no projeto. Os riscos em DBC consistem no tempo e esforço para desenvolvimento do componente, clareza de requisitos, alto grau de reusabilidade e manutenção [10]. Em DDS, a distância geográfica entre as unidades de desenvolvimento como principal limitação, que influencia diretamente no sucesso da realização das atividades de comunicação e de coordenação do projeto [3].

Com o objetivo de superar essas limitações, neste trabalho as abordagens DBC e DDS foram aplicadas na produção do *middleware* brasileiro de TV Digital, o Ginga, inserida no contexto do projeto Ginga *Code Developer Network* (GingaCDN).

A GingaCDN constitui-se como uma rede de desenvolvedores de componentes, cuja comunicação e gerência são feitas por meio de um ambiente colaborativo, por onde também é realizada a difusão de conhecimento entre seus integrantes, e com isso capacitando novos profissionais na área. Além disso, foi definido um modelo arquitetural componentizado como ponto de partida para a construção de componentes do *middleware* e um modelo de processo de desenvolvimento distribuído, cujo objetivo é tratar aspectos de conformidade com a arquitetura especificada, verificação e validação, integração contínua e reuso dentro da rede. Este projeto também contempla uma implementação de referência para o ambiente imperativo do Ginga, chamado de Ginga-J, cujos detalhes do seu desenvolvimento podem ser encontrados em [1].

O restante deste artigo está organizado da seguinte forma: a seção 2 apresenta arquitetura do *middleware* Ginga. A seção 3 discorre sobre a GingaCDN. A seção 4 descreve o funcionamento do processo de desenvolvimento distribuído adotado na rede. A seção 5 descreve a forma como os componentes são verificados e validados na rede. A seção 6 apresenta as considerações finais, trabalhos futuros e em andamento.

2. ARQUITETURA DO *MIDDLEWARE* GINGA
O Ginga é a especificação de *middleware* do SBTVD, resultado da junção dos *middlewares* FlexTV **Error! Reference source not found.** e MAESTRO **Error! Reference source not found.**, desenvolvidos por consórcios liderados pela UFPB e PUC-Rio no projeto SBTVD **Error! Reference source not found.**, respectivamente. Essas duas soluções foram integradas no Ginga, sendo agora denominadas de Ginga-J (Máquina de Execução), e Ginga-NCL (Máquina de Apresentação), respectivamente,

tomando por base as recomendações internacionais da ITU J202 [15] e ITU J201 [16].

Além dos Ginga-J e Ginga-NCL, há o subsistema Ginga Common Core (Ginga-CC), que tem por base a especificação ITU J200 [17], cuja responsabilidade é oferecer funcionalidades específicas de TV Digital comuns para os ambientes imperativo e declarativo, abstraindo as características específicas de plataforma e hardware para as outras camadas acima. Como suas principais funções, podemos citar a: exibição e controle de mídias, o controle de recursos do sistema, canal de retorno, dispositivos de armazenamento, acesso a informações de serviço, sintonização de canais, entre outros.

A arquitetura conceitual do Ginga é apresentada na Figura 1, onde é possível visualizar componentes de funcionalidades específicas dentro de cada um dos subsistemas do *middleware*: (i) sistema operacional; (ii) camada de núcleo comum; (iii) máquina de execução Ginga-J e; (iv) máquina de apresentação Ginga-NCL.

A filosofia de desenvolvimento da GingaCDN baseia-se na forte componentização do Ginga-CC, uma vez que essa abordagem características como: (i) reuso, onde dependendo das especificações do projeto, é possível utilizar um componente na produção de um outro componente ou sistema; (ii) crescimento funcional, com a adição de novas funcionalidades internamente ao componente e manutenção das interfaces iniciais e incrementos das novas; (iii) adaptabilidade, com a possibilidade de trocar componentes que melhor se adéquam as necessidades do projeto; (iv) facilidade de testes, já que as funcionalidades do componente são testadas isoladamente, permitindo que problemas sejam detectados e corrigidos previamente a integração com outros componentes; (v) atualização, que permite ao sistema atualizar ou substituir um componente específico ao invés de todo o software quando uma atualização deve ser feita.

Segundo [7], componentes de software são unidades de produção, aquisição e desenvolvimento independentes que podem ser compostas em sistemas de funcionamento. Para permitir composição, um componente deve obedecer a um modelo de componentes específico e ter como meta uma plataforma de componentes particular. Diante disso, o objetivo foi enfatizar a modelagem de *software* através da decomposição do sistema em componentes funcionais com interfaces de interação bem definidas. Neste contexto, um modelo de componentes padroniza o esquema de instanciação, composição e o ciclo de vida dos componentes do sistema e um ambiente de execução de *software* responsável por gerenciar os componentes garantindo as especificações definidas pelo respectivo modelo de componentes.

A partir daí, foram seguidas a etapa adoção de um modelo e ambiente de componentes e a especificação de um conjunto de interfaces para o Ginga-CC compatíveis conceitualmente aos componentes propostos e adaptar seus requisitos para uma implementação que suporte as maquinas de apresentação e execução do *middleware*.

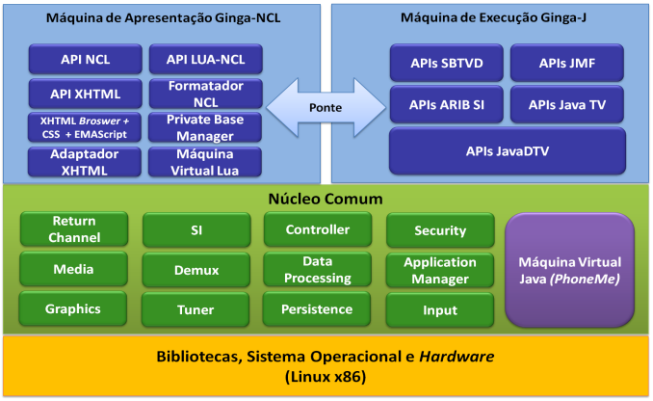

Figura 1. Arquitetura conceitual do middleware Ginga.

Como modelo e ambiente de componentes adotou-se o FlexCM [18], que segue uma abordagem declarativa, onde os componentes definem suas dependências explicitamente (interfaces requeridas) e o ambiente de execução carrega e provê as dependências através do padrão de injeção de dependências.

Para o núcleo comum do *middleware*, foram especificados os seguintes compoentes: (1) **`Tuner`** - sintoniza e controla o acesso aos múltiplos fluxos de transporte da rede; (2) **`SI`** – obtém informações de serviço do fluxo de transporte, ou seja, quais fluxos elementares (semântica) de áudio, vídeo e dados estão sendo transmitidos, além de informações como classificação indicativa, sinopses e horários; (3) **`Demux`** - disponibiliza filtros específicos para selecionar fluxos; (4) **`Media`**: comunica-se com os decodificadores (*hardware* ou *software*), a fim de gerenciar e exibir a apresentação dos fluxos elementares de áudio e vídeo; (5) **`Data Processor`** - processa e separa dados transmitidos (como aplicações) de forma multiplexada no fluxo de transporte (do inglês, *Transport Stream*) MPEG-2. (6) **`Graphics`** - permite o desenho de componentes gráficos; (7) **`Input Manager`** - tratamento de eventos disparados pelo usuário através do controle remoto, pelo painel do próprio terminal de acesso, por um teclado, ou por algum outro dispositivo de entrada; (8) **`Return Channel`** - provê interfaces para utilização do canal de retorno, por exemplo, através de um modem de linha discada, ADSL, Ethernet, Wimax ou 3G; (9) **`Application Manager`**: carrega, instancia, configura e executa aplicações; (10) **`Persistence`** - gerencia recursos de armazenamento não voláteis; (11) **`Security`** - verifica a autenticação e permissões de aplicações interativas; (12) **`Middleware Manager`** - responsável pelo gerenciamento funcional do *middleware*. Detalhes acerca da implementação componentizada do Ginga-CC, bem como da máquina de execução Ginga-J são apresentados por Kulesza *et. al.* em [1]

3. A REDE GINGACDN

A Ginga *Code Development Network* ou simplesmente GingaCDN é uma rede de desenvolvedores de componentes e aplicações para o *middleware* brasileiro de TV Digital Ginga. Esta iniciativa faz parte do programa Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC), atualmente incubado pela Rede Nacional de Ensino e Pesquisa (RNP),

A partir da normatização do middleware Ginga como padrão brasileiro de TV digital, gerou-se uma grande demanda no setor de software do mercado brasileiro. Uma das possibilidades constituiu-se na tentativa de implementações do middleware Ginga para as mais diversas plataformas de hardware. Neste sentido, começaram a surgir as primeiras implementações do *middleware*, tendo como resultado inicial o OpenGinga, uma versão *open-source* do Ginga, construído a partir do esforço conjunto dos laboratórios de pesquisa em TV digital LAViD, da UFPB, e TeleMídia, da PUC-Rio.

Entretanto, o OpenGinga, não teve durante seu desenvolvimento inicial contribuições advindas de outras instituições. Dessa forma o conhecimento no desenvolvimento de *middleware* para a TV digital brasileira sempre ficou restrito a um núcleo pequeno de pesquisadores. Em decorrência deste cenário, o GingaCDN (Ginga – Code Development Network) surgiu com o intuito de desenvolver um ambiente que permitisse o desenvolvimento distribuído de software no domínio de televisão digital no Brasil e no mundo.

O projeto constituiu-se inicialmente de 14 laboratórios de pesquisa e desenvolvimento, em 13 instituições de ensino, presentes em nove estados e quatros regiões do Brasil. Entretanto, a GingaCDN trata-se de uma rede aberta, de forma que são aceitos novos membros de qualquer parte, que desejem colaborar. Atualmente, a rede consta com um número superior a 300 usuários cadastrados. Dessa forma, os parceiros distribuídos pelos diferentes estados brasileiros deverão constituir-se em diferentes equipes que poderão interagir entre si na produção de componentes e/ou aplicações para o *middleware*.

Através do GingaCDN, tem-se a possibilidade de prover um acesso livre a tecnologias relacionadas ao desenvolvimento do *middleware* Ginga, beneficiando empresas e insttituições que atuam na área, aumentando a competitividade para disputar espaço no mercado nacional de desenvolvimento de software embarcado para televisão digital. Além disso, é possível promover a difusão de conhecimento e o intercâmbio de experiências em TV Digital entre os diversos integrantes da rede.

Dessa forma, para melhor gerenciar e suportar este cenário, a estrutura de software do Ginga, isto é sua arquitetura, teve de ser desenvolvida em termos de componentes de software. Através da criação de uma arquitetura baseada em componentes, apresentada na seção 2, o projeto possibilitou um avanço para melhor suportar as contribuições advindas dos parceiros. Agora, cada parceiro sendo formada por uma equipe poderá ser responsáveis em separado pelo ciclo de desenvolvimento de cada componente.

Dessa forma, é possível existir dentro da rede de desenvolvedores, a situação apresentado na Figura 2, no qual diferentes *middlewares* podem ser construídos a partir do componentes que melhor se adéqüem com os requisitos de um cenário especificado por uma entidade desenvolvedora do Ginga

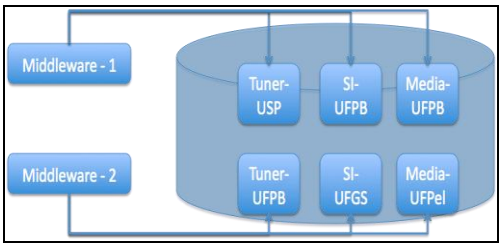

Figura 2. Construção de diferentes cenários de middlewares a partir de componentes diversos.

4. PROCESSO DE DESENVOLVIMENTO

Segundo [19], um processo de *software* é um conjunto bem definido de atividades, métodos, práticas e tecnologias, utilizadas por um grupo de pessoas, para a produção de um produto de *software*.

No desenvolvimento de equipes co-localizadas, normalmente compartilha-se e é acessível de maneira rápida um conjunto de recursos disponíveis dentro da própria organização como, por exemplo, um processo e práticas de desenvolvimento e de gerenciamento que auxiliam no desenvolvimento. Além destes recursos, particularidades envolvidas no negócio da empresa, como a proximidade com o cliente, entendimento da infra-estrutura local e o acesso a complexos laboratórios de testes são exemplos de recursos requeridos [20].

Através da separação da equipe pela distância geográfica, estes recursos passam a não ser mais efetivamente utilizados. Por essa razão, geralmente faz sentido co-localizar as atividades de certo um estágio do processo (por exemplo, localizar a fase de projeto, próximo ao cliente), e distribuir as outras fases em unidades que estão geograficamente distribuídas [20].

Figura 3. Fases do processo de desenvolvimento da GingaCDN.

O processo de desenvolvimento colaborativo do projeto GingaCDN foi desenvolvido com o intuito de padronizar práticas ágeis e objetivas de implementação distribuída de componentes para *middlewares* de TV Digital. Com isso, foram definidos papéis dentro do ambiente colaborativo da rede, através dos quais múltiplos usuários podem modificar diferentes seções

correspondentes à suas responsabilidades no projeto de maneira simultânea. Tais papéis estão distribuídos nas cinco fases definidas no processo de desenvolvimento colaborativo. A Figura 3 apresenta as fases do processo, bem como os papéis a ela relacionados. Dessa forma, ficam claras para cada um dos membros da rede quais são suas atribuições e responsabilidades.

O papel **Manager** representa a função de gerência, cuja responsabilidade é garantir o êxito do desenvolvimento do *middleware*, gerenciando as entregas de *releases* e o acompanhamento das atividades do *Leader*. Além disso, entre suas atividades estão: (1) criação de subprojetos no ambiente de desenvolvimento; (2) especificar requisitos de componentes; (3) Relacionar todas as especificações, *templates*, tutoriais e outros materiais complementares que auxiliem no desenvolvimento.

O **Leader**, por sua vez é o papel responsável pela gestão de todos os recursos de um sub-projeto, desde os documentos relacionados ao projeto até os próprios desenvolvedores. Dentre suas atividades estão (1) criar *tickets* de desenvolvimento obedecendo ao plano de *releases* criado pelo *Manager;* (2) atribuir *tickets* para os desenvolvedores; (3) acompanhar o desenvolvimento dos entregáveis e; (4) gerenciar riscos.

A função do papel **Developer** é a implementação do código do componente, obedecendo aos requisitos especificados para o componente pelo *Manager*, além de construir os seus respectivos testes unitários.

O **Reviewer** é responsável pela revisão dos códigos e testes unitários gerados pelos desenvolvedores. Suas atividades podem ser detalhadas em: (1) revisar código do componente; (2) verificar conformidade ao padrão de codificação; (3) validação de testes de unidade, com a execução de testes externos e internos; (4) análise de cobertura de testes de unidade; (5) inspeção manual do código do componente; (6) gerar relatório de revisão.

O papel **Integrator** representa a pessoa ou equipe responsável pela criação e execução dos testes de integração dos componentes, dentro de um cenário de uso do *middleware*. As suas atividades são: (1) identificar a abordagem de implementação mais apropriada para um dado teste; (2) definir plano de Teste de Integração; (3) implementar, configurar e executar os testes; (4) registrar resultados e verificar a execução dos testes; (5) analisar erros de execução e recuperar-se deles e (6) gerar relatório de integração.

As etapas do processo foram definidas tendo em vista particularidades do domínio da TV Digital, como: velocidade de evolução do mercado, modularidade e a alta complexidade do código fonte envolvido.

O início do processo de desenvolvimento é marcado pela fase de **Concepção**, sob a total responsabilidade do *Manager*, que cria um novo subprojeto dentro da comunidade. Nessa fase são geradas as especificações dos componentes e das APIs, estabelecido o plano de entrega de *releases* e quais requisitos serão implementados em cada uma, além da criação de *tickets* para os *Leaders.*

Na fase **Elaboração** *Leader* cria os *tickets* para o *Developer*, obedecendo ao cronograma de *releases*, da mesma forma que os *tickets* de *Leader*, que dependendo da necessidade, serão atribuídos ou disponibilizados ao desenvolvedor que deseje realizar a tarefa

Na fase **Construção,** os *Developers* que trabalharão nas tarefas de desenvolvimento criadas pelos *Leaders*. Nesta etapa são utilizados os mesmos documentos gerados na etapa anterior, obedecendo ao cronograma definido pelo *Leader*. A tarefa é dada como concluída, quando o desenvolvedor gera o código do componente, junto com os testes de unidade e casos de teste.

A etapa de **Revisão**, ou **Testes** envolve dois papéis: o *Reviewer* e o *Integrator*. Inicialmente o primeiro analisa todos os documentos gerados juntamente com o componente desenvolvido, verificando sua conformidade à todas as especificações documentadas e requisitos definidos inicialmente. Com base nos resultados obtidos, o *Reviewer* produz um relatório de revisão. Ao fim, com o componente e toda documentação, o *Integrator* realiza um teste de integração do componente, gerado no sub-projeto, integrando ao projeto como um todo.

Caso o componente seja reprovado nessa última etapa, devido a um *bug*, a não atender um determinado requisito, ou falhar na integração, volta-se à etapa de elaboração para que o *Leader* possa instruir os desenvolvedores sobre o é necessário ser feito para corrigir o componente. A essa etapa, dá-se o nome de **Transição**, cuja responsabilidade atribui-se ao *Manager*. Ela consiste apenas de uma revisão do que foi produzido nas etapas anteriores para verificar se o resultado consiste com as especificações iniciais determinadas na criação do subprojeto. Caso a análise seja positiva, o componente é dito como integrado e sua versão será disponibilizada no repositório público da comunidade com intuito de ser utilizado por qualquer outro membro da mesma para futuras implementações ou como referência para um novo projeto.

Para suportar este processo, servir como centro de conhecimento e memória do time, fornecer a todos os membros uma visão de geral do projeto e criar um senso de comunidade [21] faz-se necessário o uso de uma ferramenta. Diante disso, foram pesquisadas e estudadas ferramentas que atendiam aos seguintes critérios: (i) aplicabilidade (ii) interface simplificada e (iii) suporte a funcionalidades de gerenciamento de projetos. Ao final do estudo optou-se pela ferramenta **Redmine** [22], por trata-se de um *software* livre que permite customização e apresenta uma maior relação de recursos em comparação com as outras ferramentas estudadas.

Diversas adições foram feitas ao código original Redmine por meio de *plugins*, refletindo-se em funcionalidades essenciais para facilitar a comunicação da equipe, bem como atingir os objetivos da comunidade. Dentre as customizações realizadas, destacam-se o Ambiente de Revisão Integrado e o Sistema de Ranking dos Usuários da Comunidade (Figura 4).

Por tratar-se de um projeto onde a comunicação entre equipes separadas geograficamente, percebeu-se a necessidade da manutenção constante de uma versão estável dos componentes presentes no repositório público da rede, de forma que uma falha local não refletisse em problemas para outras unidades de desenvolvimento.

Figura 4. Sistema de Ranking dos Usuários da Comunidade.

Utilizamos a ferramenta *BuildBot* [24], desenvolvida com o objetivo de obter uma integração contínua entre os diversos componentes desenvolvidos na rede, bem como buscar a manutenção constante de uma versão estável do *middleware.* Dessa forma, é feita a compilação periódica ou programada de todos os componentes presentes no repositório público e do *middleware,* verificando e reportando possíveis erros aos seus respectivos responsáveis.

5. VERIFICAÇÃO E VALIDAÇÃO DOS COMPONENTES

O teste trata-se de uma atividade essencial para que se atinja bons níveis de qualidade em produtos de *software*. Seguindo esses conceito e normas sobre processos de teste foi desenvolvido um processo de verificação e validação iterativo e incremental baseado em artefatos de teste definidos na norma IEEE 829-1998 [23].

O processo de Verificação e Validação (V&V) dos componentes tem como base a proposta de testes para o Ginga-J baseada no FlexCM, podendo ser encontrada em [11]. Na Figura 5 são apresentadas as fases do processo de verificação e validação para os componentes do *middleware*, o qual possui cinco fases distintas: (i) Testes de Unidade, (ii) Revisão do Componente, (iii) Inspeção de Compatibilidade entre Licenças, (iv). Testes de Integração e (v) Testes de Sistema.

Na fase de **Testes de Unidade**, é feito o planejamento inicial dos testes com base em componentes específicos, no nível de código de implementação. Tem como marco o uso de casos de uso específicos para implementação do componente e de seus testes unitários, e a primeira release do componente, feita pelas células de desenvolvimento (parceiros da rede GingaCDN). Esta fase é executada paralelamente ao ciclo de desenvolvimento, onde os testes unitários são criados após a codificação dos componentes. Ao término, o líder de desenvolvimento solicita uma revisão do componente, para saber se o que está sendo feito, está sendo feito corretamente.

Na etapa de **Revisão de Componente,** todo código deverá ser avaliado por um revisor de código. Neste caso o papel do revisor é avaliar se o desenvolvedor produziu código que poderá atrapalhar a semântica do projeto ou a manutenção num futuro próximo; também deve verificar se o código produzido é eficiente e atual com relação aos procedimentos e padrões definidos para o projeto. Assim, nesta fase é executado uma revisão nos códigos gerados, tanto o da implementação do componente quanto dos testes unitários. Está tarefa é realizada por um revisor, que é uma pessoa externa da equipe de desenvolvimento do componente.

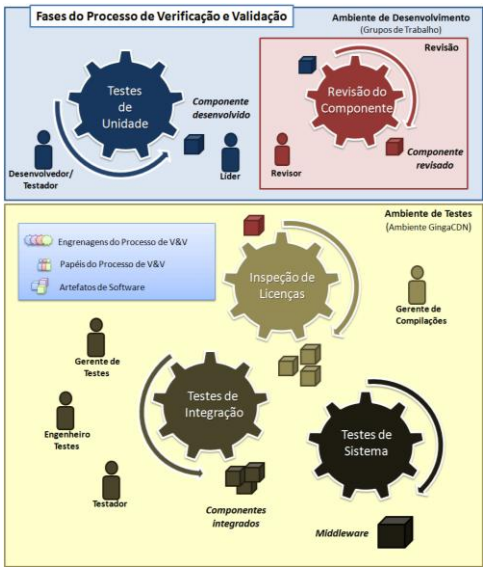

Figura 5. Visão Geral do Processo de V&V.

A fase seguinte, denominada **Inspeção de compatibilidade entre licenças,** é realizada a validação das licenças que acompanham cada componente do Ginga-CC numa determinada configuração de arquitetura. Para cada composição (ou configuração) de componentes do núcleo comum do *middleware* dever-se analisar se é possível a compatibilidade de todas as licenças presentes, e qual é a licença final.

Os **Testes de Integração e de Sistema** são realizados depois que o componente desenvolvido foi aprovado na revisão e ter licenças compatíveis entre os componentes a serem testados. Nessa fase, ele é testado juntamente com outros componentes que também estão sendo desenvolvidos no intuito de validar a consistência e a comunicação entre os componentes e o funcionamento do *middleware* como um todo.

A partir desta fase, é iniciado um processo de testes interno, realizado pela equipe de qualidade do GingaCDN, para auxiliar no gerenciamento, planejamento, elaboração e execução dos testes de integração e de sistema.

6. CONSIDERAÇÕES FINAIS

Este trabalho apresentou a experiência na construção de uma rede de desenvolvimento colaborativo de componentes e aplicações para o *middleware* brasileiro de TV Digital, o Ginga. A rede foi composta inicialmente por instituições de pesquisa em diferentes regiões geográficas e posteriormente aberta ao cadastro de usuários.

Para tornar homogêneo o desenvolvimento do *middleware*, foi definida uma arquitetura usando desenvolvimento baseado em componentes, que trouxe uma série de benefícios, entre eles: (i) conhecimento de arquitetura em nível de modelo; (ii) facilidades na configuração dos componentes individuais; (ii) configuração do sistema como um todo e (iv) possibilidade de gerenciar

diferentes arquiteturas facilitando também a execução de testes de unidade e integração de diferentes porções da arquitetura.

Dentro do ambiente de desenvolvimento colaborativo proposto para o projeto, as dificuldades impostas pela distribuição geográfica das equipes foram superadas com êxito através da utilização de ferramentas customizadas de apoio à gestão de projetos. Neste ambiente, o processo de desenvolvido foi adotado com o intuito de padronizar práticas ágeis e objetivas de implementação distribuída de componentes para *middlewares* de TV Digital, com a definição de fases e papéis bem especificados, através dos quais múltiplos usuários podem modificar diferentes seções correspondentes à suas responsabilidades no projeto de maneira simultânea.

Com as ferramentas desenvolvidas adotadas, foi possível obter uma integração contínua do código, além de contemplar desde a definição de atividades à especificação de documentos e tutoriais para guiar o desenvolvimento e o intercâmbio de conhecimento na rede. Além disso, foi proposto e adotado um trabalho para a definição de um modelo de validação e verificação de *middleware* para TV Digital.

7. REFERÊNCIAS

[1] Kulesza, R. *et. al.* 2010. Ginga-J: Implementação de Referência do Ambiente Imperativo do Middleware Ginga. In: Webmedia 2010 – Simpósio Brasileiro de Sistemas Multimídia e Web, 2010, Belo Horizonte. Anais do WebMedia 2010 - Simpósio Brasileiro de Sistemas Multimídia e Web, 2010. *(to appear).*

[2] Sengupta B, Chandra S, Sinha V. A research agenda for distributed software development. International Conference on Software Engineering 2006.

[3] Prikladnick, S. M., Sabrina, J. L. N. A. MuNDDoS: A Research
Group on Global Software Development. In: IEEE International Conference on Global Software Engeneering. Los Alamitos: IEEE Computer Society Press, 2006. v. 1. p. 251-252.

[4] Herbsleb, J. D. e Grinter, R. E. Splitting the Organization and
Integrating the Code: Conway's Law Revisited. In: International Conference on Software Engeneering (ICSE), p. 85- 95, Los Angeles, CA, 1999.

[5] Grinter, R. E., Herbsleb, J. D., e Perry, D. E. The Geography of Coordination: Dealing with Distance in R&D Work. In: Internationl Conference on Supporting GroupWork, Phoenix, AZ, Novembro, 1999, p.14-17.

[6] KIEL, L. Experiences In Distributed Development: A Case Study. In:
ICSE International Workshop on Global Software Development, Portland, Oregon, EUA, 2003.

[7] Szyperski, Clements (2002). "Component Software: Beyond bject-Oriented Programming". Second Edition, Addison-Wesley.

[8] Herbsleb, J.D., Grinter, R.(1999) "Splitting the organization and integrating the code: Conway's Law revisited", In 21th International Conference on Software Engineering (ICSE'99).

[9] Oliveira, J. P. F. et al. 2007. Uma Infra-estrutura Colaborativa de Suporte ao Desenvolvimento Distribuído Baseado em Componentes. In: SBSC 2007 - Simpósio Brasileiro de Sistemas Colaborativos., 2007, Rio de Janeiro. Anais do SBSC 2007 - Simpósio Brasileiro de Sistemas Colaborativos., 2007.

[10] Crnkovic, Ivica. 2003. Component-Based Software Engineering–New Challenges in Software Development". In Information Technology Interfaces (ITI'03). 2003.

[11] Caroca, C.; Tavares, T. A. . Um Modelo de Processo de Testes para os Componentes do Núcleo Comum do Middleware OpenGinga. In: IX Workshop de Teses e Dissertações do WebMedia 2009 – Simpósio Brasieiro de Sistemas Multimídia e Web 2009.

[12] Leite, L. E. C., et al. 2005. FlexTV – Uma Proposta de Arquitetura de Middleware para o Sistema Brasileiro de TV Digital. Revista de Engenharia de Computação e Sistemas Digitais. 2005, Vol. 2, pp. 29-50.

[13] Soares, L. F. G. 2006. MAESTRO: The Declarative Middleware Proposal for the SBTVD. Proceedings of the 4th European Interactive TV Conference. 2006.

[14] SBTVD. 2005. Projeto do Sistema Brasileiro de TV Digital. Disponível em: http://sbtvd.cpqd.com.br. Acesso em: 15/06/2010.

[15] ITU-T Recommendation J.202: Harmonization of procedural content formats for interactive TV applications. 2003.

[16] ITU-T Recommendation J.201: Harmonization of declarative content format for interactive television applications. 2004.

[17] ITU-T Recommendation J.200: Worldwide common core – Application environment for digital interactive television services. 2001

[18] Miranda Filho, S. et al. 2007. Flexcm - A Component Model for Adaptive Embedded Systems. In: COMPSAC IEEE International Computer Software and Applications Conference, 2007, Beijing. p. 119-126.

[19] Pressman, R. S. Software Engineering. A Practitioner's Approach. Fifth Edit, 2001.

[20] The Geography of Coordination: Dealing with Distance in R&D Work.

[21] CARMEL, E. Global Software Teams – Colaborating Across Borders and Time Zones. Upper Saddle River, NJ: Prentice Hall 1999. p 269.

[22] Redmine: http://www.redmine.org/ Acessado em 05/07/2010.

[23] IEEE Std. 829. 1998. Standard for Software Test Documentation. 1998.

[24] Ferramenta BuildBot. http://dev.openginga.org:8010/. Acessado em julho de 2010.