Ginger-Video-3D - Adaptação de uma ferramenta de *rendering* gráfico/codificação vídeo para execução paralela em sistema *peer-to-peer* de partilha de ciclos.

João Manuel Rebelo da Cruz Morais[1]

[1] Instituto Superior Técnico, Universidade Técnica de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal

joao.c.morais@ist.utl.pt

Abstract. Este documento descreve a investigação realizada no seguimento do trabalho descrito em [1], mas com um âmbito aplicacional mais reduzido e centrado no processamento de vídeo. O desenvolvimento de uma infrastrutura *peer-to-peer* de partilha de ciclos na qual qualquer pessoa pode submeter as suas aplicações levanta algumas importantes implicações. No sentido de compreender que implicações são estas, é feita uma análise a algumas soluções genéricas de computação distribuída e são apresentadas as suas limitações face a uma infrastrutura *Grid* desta natureza. É feito ainda um estudo ao trabalho relacionado na área da manipulação de alto desempenho de conteúdo multimédia e ainda na área de adaptação transparente de aplicações existentes a ambientes não nativos, com o objectivo de descobrir problemas e soluções, comuns a este trabalho.

Keywords: partilha de ciclos, processamento paralelo e distribuído, *grids*, *gridlets*

1. Introdução

A computação distribuída ou em *Grid* envolve um grande número de sistemas e recursos heterogéneos, dispersos geograficamente e sobre o domínio de diversas instituições. É o objectivo de uma infrastrutura *Grid* abstrair este conjunto de entidades como um único recurso virtual e dinâmico resolvendo ao mesmo tempo problemas como segurança, gestão de recursos e dados e prestação de serviços de localização. [1]
A observação que um grande número de computadores desktop na Internet estão em geral subaproveitados face às suas capacidades, gerou nos últimos anos um grande interesse no aproveitamento destes recursos livres, e tornou-se o veículo ideal para o aparecimento de várias infrastruturas *Grid* para realização de computação distribuída cujo exemplo mais conhecido e de maior sucesso é o SETI@home [2].

2. Motivação

Apesar de a Internet ter motivado o aparecimento de vários projectos de computação distribuída, as infrastruturas *Grid* que os suportam são normalmente exploradas unicamente em projectos científicos ou em processos de negócio, impossibilidando o utilizador comum de fazer *deploy* da sua própria aplicação. Projectos populares de partilha de ciclos como o Folding@Home contornam a questão do porquê de contribuir, por se tratar de causas colectivas que resultam em benefício para a humanidade.

Algumas soluções têm sido propostas para colmatar o problema do acesso aberto mas controlado aos recursos numa *Grid*, como o OurGrid [3] e o Integrade [4] mas ambas tiveram um sucesso muito reduzido por se tratarem de soluções incompletas e com propósitos pouco interessantes do ponto de vista do utilizador comum.

O GINGER [5] tem como objectivo criar uma infrastrutura *Grid* onde a posição de altruísta dos utilizadores na rede é substituída por uma posição mutualista, onde todos se podem ajudar a si próprios ao mesmo tempo que ajudam outros, dando acesso a tecnologias *Grid* a qualquer pessoa ou comunidade interessada.

Com o crescimento do acesso residencial a banda larga surgiram novas oportunidades de partilhar informação na Internet, sendo uma das mais proeminentes a partilha de vídeos *home-made*. O Youtube é o melhor exemplo do crescimento desta área, contando com cerca de 10% (e a crescer) de todo o tráfego realizado na Internet [6].

No entanto existe ainda um passo essencial na compressão ou transcodificação dos vídeos produzidos, antes de serem passíveis de colocar na Internet, operações demoradas que poderiam ver o tempo atenuado através da divisão da carga computacional por vários recursos numa *Grid*.

O objectivo do Ginger-Video-3D implementar as ideias apresentadas no Ginger [5] e aplicá-la em concreto num sistema de processamento de vídeo, concentrando-se também na questão pouco abordada nas abordagens actuais, de como dividir um trabalho em várias unidades lógicas que possam ser executadas de forma independentemente em diferentes máquinas, passando ainda pela pesquisa de métodos de adaptação aplicações já existentes para um ambiente de execução distribuída.

Assim conseguimos uma relativamente simples adaptação de programas comuns de processamento intensivo que convencionalmente correm de forma sequencial, mas que

transparentemente através do Ginger estão de facto a tirar partido de uma arquitectura distribuída, com consequentes melhoramentos ao nível do desempenho e tempos de resposta, melhoramentos estes perceptíveis do ponto de vista do utilizador.

3. Trabalho Relacionado

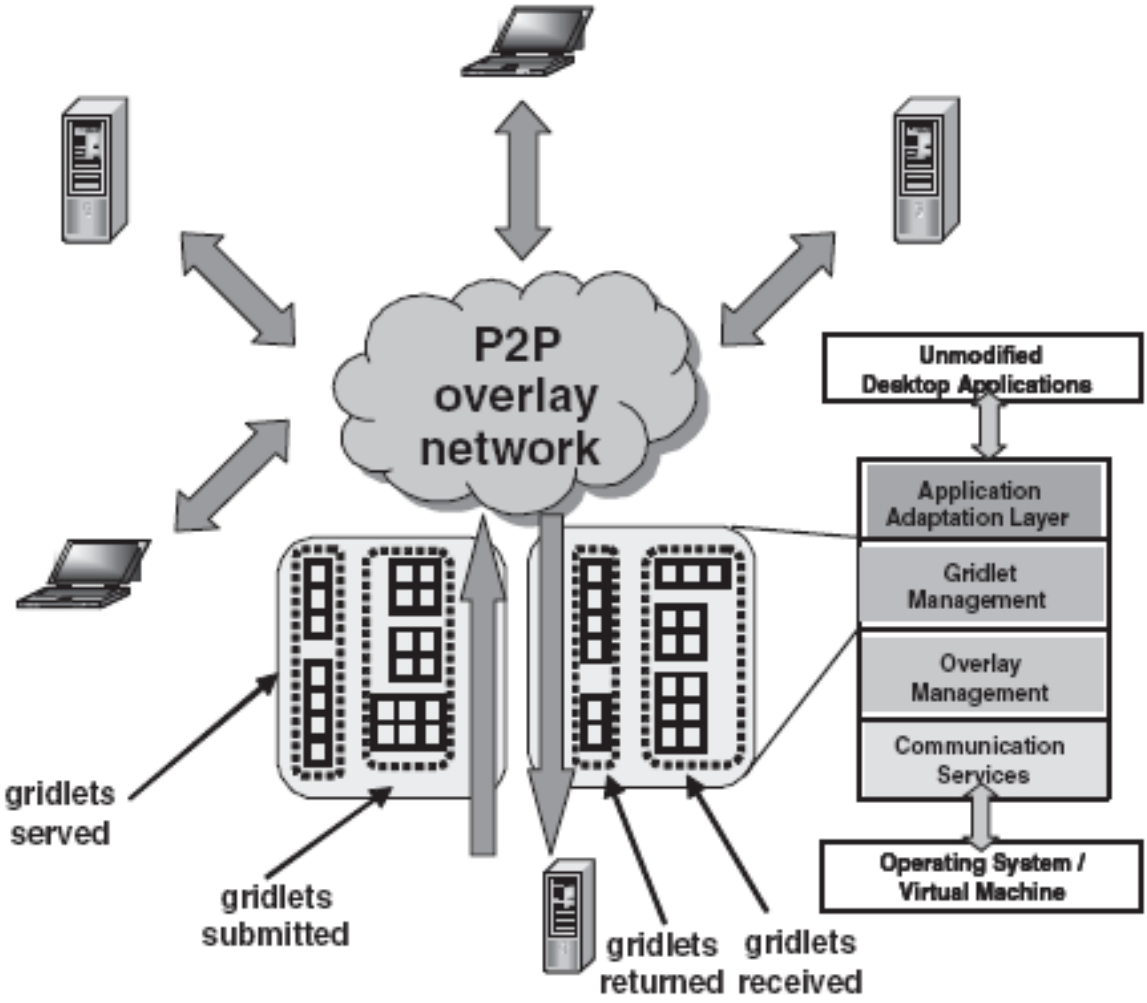

Figura 1: Vista geral do GINGER

A figura 1 apresenta uma visão geral da arquitectura do Ginger (detalhada em [5] que é composta por várias camadas independentes. De acordo com os objectivos do Ginger-Video-3D vão-se apenas abordar as que têm maior interesse para este trabalho, que são: desktop *grids* e partilha de ciclos, processamento multimédia distribuído e adaptação de aplicações.

3.1. Infrastruturas e Arquitecturas *Grid* de Partilha de Ciclos

A computação distribuída quando aplicada à utilização de recursos em estado passivo e partilhados por diversas fontes heterogéneas, levanta uma série de interessantes questões:

- **Incentivos**: que tipos de incentivos são necessários para motivar os utilizadores a ceder os seus recursos (nomeadamente ciclos de relógio)?
- **Descoberta de recursos e escalonamento eficiente**: Como encontrar recursos na rede num ambiente dinâmico e de livre acesso? Como seleccionar os melhores ou mais apropriados?
- **Reputação**: Que clientes demonstram um comportamento confiável que irão devolver uma resposta correcta? Quais responderão primeiro? Qual é a garantia que a "pergunta" irá eventualmente obter uma "resposta"?
- **Justiça:** Como garantir uma partilha de recursos justa, entre os vários utilizadores da *grid*, de forma a impedir monopólios no acesso a estes?
- **Segurança**: Como é que um cliente e a rede como um todo se podem defender contra ataques de *DDoS*, pedidos com código malicioso ou inútil e ainda respostas manipuladas com valores incorrectos?

De seguida resumem-se algumas infrastruturas e arquitecturas *Grid*, que visam resolver estes e outros desafios que a computação *Grid* levanta.

Cluster Computing on the Fly [7] descreve uma arquitectura *peer-to-peer* de partilha de ciclos, propondo soluções a problemas de computação intensiva idênticos aos enfrentados pelo projecto SETI@home e ainda a aplicações onde os recursos são usados como *Point-of-Presence*, útil por exemplo para a monitorização e análise de tráfego numa rede. Aplicações de grande carga computacional, são escalonadas através de um processo designado de *Wave Scheduler* que organiza os recursos por zona geográfica com o objectivo de aproveitar os períodos nocturnos de menor actividade. Implementa ainda um sistema inteligente e eficaz de detecção da validade dos clientes através de um sistema de pergunta/resposta. No entanto o projecto parece ter-se ficado pela promessa, não tendo ainda disponibilizado um *middleware* sobre o qual se possa construir novas aplicações *Grid*.

O BOINC [2] é um *middleware* de computação voluntária de aplicações não comerciais tornado popular por conhecidas aplicações como SETI@home e o Folding@home. Nestes sistemas existe um coordenador central que envia as unidades de trabalho para os clientes que actuam de forma altruísta não recebendo qualquer recompensa pelo seu trabalho, para além do registo da sua contribuição. Para além da motivação óbvia para o bem comum que poderá provir destes projectos, o BOINC oferece uma interface didáctica que permite visualizar alguns dos passos da execução do processo, fazendo também um esforço por gerar interesse através da competição entre grupos de utilizadores, contabilizando-a através de um elaborado sistema de créditos.
O modelo de programação está bem delineado no entanto o acesso à infrastrutura e aos recursos obriga ao desenvolvimento de clientes e servidores BOINC o que torna esta solução inviável para o utilizador comum. Para além disso, as aplicações têm de ter um cariz científico e necessitam de passar um processo de controlo e certificação, o que limita bastante o âmbito das aplicações que podem ser executadas na rede.
Assim, no BOINC as unidades computacionais acabam por ser corridas directamente sobre o sistema operativo sem qualquer solução de *sandboxing* já que se assume que as entidades são confiáveis. Para prevenir problemas de *man-in-the-middle* o código é assinado com a chave privada da entidade responsável pela unidade de trabalho.
O envio de resultados maliciosos é atenuado através do envio de diversas cópias da mesma unidade de trabalho para diversos clientes, sendo feita uma posterior votação para aceitar a resposta com maior probabilidade de estar correcta.

O Condor [8] realiza uma parte importante das funcionalidades de uma *Grid*, descrevendo um mecanismo de gestão e partilha de recursos em ambientes sobre o mesmo domínio administrativo, capturando ciclos inutilizados em estações de trabalho comuns para a realização de tarefas que requerem não só um grande poder de processamento mas também uma grande quantidade de informação.
É na sua essência um sofisticado sistema de escalonamento de processos garantindo um acesso justo e equilibrado aos recursos disponíveis garantindo que o uso da máquina não é afectado, quando esta for requisitada pelo utilizador da mesma.

Globus [9] é um *toolkit* organizado sobre a forma de um conjunto de *workflows* desenvolvidos sobre uma camada *SOA* que por sua vez assenta e pode utilizar um conjunto de ferramentas que simplificam o escalonamento, partilha e monitorização de

recursos distribuídos por uma qualquer rede, permitindo modelar e construir uma infrastrutura *Grid* à medida das necessidades. A abordagem do Globus é de estruturar os componentes duma *Grid* em camadas numa aproximação *bottom-up*, começando por funcionalidades de baixo nível como comunicação, autenticação, serviços de informação e acesso a dados, desenvolvendo depois vários serviços de alto nível sobre estas. O objectivo do Globus é de revelar uma infrastrutura que abstraia a *Grid* como um metacomputador virtual em permanente mudança, permitindo a fácil criação de novos serviços ou aplicações de alto desempenho adaptadas a este ambiente.
O Globus têm como característica dominante (e mais valia) ser o projecto que implementa o maior número de standards do OGF (*Open Grid Forum*), um consórcio de diversas entidades que têm por objectivo normalizar os vários componentes que habitualmente se encontram em infrastruturas de computação em G*rid*.

Integrade [4] é um *middleware* que utiliza uma arquitectura inter-cluster apoiada em gestores clusters que comunicam entre si e que por sua vez controlam um conjunto de máquinas que cedem recursos ao *overlay*. Este tipo de arquitectura foi desenvolvida para suportar aplicações MPI ou de *bag-of-tasks* (com/sem comunicação inter-processo) e também para dar um maior controlo sobre as topologias de rede onde estas podem ser executadas. Neste sistema os clientes estão ligados de forma centralizada ao gestor do cluster e desconhecem da existência uns dos outros, logo a submissão é feita através deste que fará o escalonamento e reencaminhamento do binário pelos restantes clientes.
Outra aspecto interessante do Integrade é o facto da gestão de recursos ser baseada em análise de padrões de utilização, em 2 fases de aprendizagem de forma a melhor perceber como e quando estes recursos podem ser utilizados. Com vista a recuperar de processos incompletos existe um sistema de *checkpoints* do estado da aplicação que permite a sua posterior continuação a partir desse ponto.

O MOSIX [10] é um sistema de gestão de clusters Linux que permite um conjunto de nós x86 actuar como um único computador paralelo. Os utilizadores podem correr código sequencial ou paralelo, ficando a cargo do MOSIX e encontrar e alocar os recursos necessários à sua execução. Para além da gestão automática dos recursos o Mosix oferece migração transparente de processos (criados por *fork*) entre nós e um ambiente seguro de execução que previne que processos remotos de aceder a recursos protegidos, através da captura de chamadas ao sistema.

O Ourgrid [3] é um *toolkit* que tal como o Integrade federa clientes em diferentes clusters e apresenta-se como uma solução completa para correr aplicações *bag of tasks* em *grids*. O sistema pode ser visualizado como uma arquitectura *peer-to-peer* entre *peers* OurGrid que por sua vez servem de servidor central para os clientes e recursos.
Este *toolkit* fornece ao utilizador ferramentas de abstracção relativas à composição da *grid*, escalonamento das aplicações pelos diferentes nós, gestão de recursos através de um mecanismo de rede de favores (recompensa clientes que mais contribuem), e segurança na execução das tarefas através de virtualização (*sandboxing*) de processos. O objectivo do OurGrid é possibilitar a qualquer utilizador uma forma acessível de submeter a sua aplicação num sistema distribuído de partilha de ciclos, no entanto presentemente o sistema não tem completo o componente de virtualização, denominado SWAN, logo só pode ser utilizados junto de fontes fidedignas o que perde um pouco a sua utilidade prática.

Butt et al. [11] descreve um protótipo de um sistema de partilha de ciclos que aproveita tecnologias *peer-to-peer* já existentes como o Pastry para permitir uma gestão eficiente da entrada e saída de clientes e também a fácil descoberta de recursos, deixando de haver a necessidade de uma entidade central. Neste projecto as aplicações correm sobre a Java Virtual Machine simplificando assim as questões de portabilidade e segurança na execução de código remoto.
Um dos objectivos principais é garantir que o sistema é justo para todos os participantes. Assim é proposto um elaborado sistema de créditos e ainda um mecanismo de detecção de progresso nas tarefas para compensar da forma mais correcta possível o utilizador que cedeu os ciclos de processamento, mesmo que não tenha conseguido por algum motivo terminar o processamento da unidade de trabalho que lhe foi confiada.

Legion [12] à semelhança do Globus procura construir uma camada de abstracção à volta da complexidade e diversidade que são características naturais sistema distribuído de partilha de ciclos. Alguns dos problemas aos quais o Legion dá maior relevância incluem escalabilidade, transparência no acesso a recursos, tolerância a faltas, segurança para os utilizadores e para os donos dos recursos, autonomia dos domínios, gestão de

recursos, extensibilidade e facilidade de desenvolvimento de novas aplicações que explorem elevados graus de paralelismo.
Uma característica interessante é que do ponto de vista do Legion todos os componentes (utilizadores, dados, aplicações, políticas, etc.) são objectos logo têm acesso imediato as vantagens do paradigma de objectos, como reutilização, contenção de falhas e redução de complexidade. Outro aspecto é que o Legion deixa à consideração do programador que tipo de políticas de segurança, replicação e alocação de recursos serão utilizadas no sistema, permitindo decidir qual o melhor *trade-off* dado o processo em questão.

O Sun Grid Engine [13] é um software *open-source* de gestão de recursos que faz a associação de requisitos de software ou hardware a um conjunto de recursos disponíveis e possivelmente heterogéneos. É usado tipicamente em clusters e é responsável por aceitar, escalonar, despachar e gerir a execução remota de um ou mais processos distribuídos.
Baseia-se numa arquitectura cliente-servidor composta por um servidor central que supervisiona e verifica que as politicas de utilização definidas pelos administradores da grid estão a ser cumpridas e que é também responsável pela interacção com o utilizador que submete as unidades de trabalho. Não é uma ferramenta de uso pessoal, sendo mais orientada a empresas ou a meios académicos.

Em seguida faz-se um resumo das características mais importantes de alguns dos projectos referidos acima e de maior utilização na actualidade:

	BOINC	**Globus**	**OurGrid**	**Sun Grid Engine**
Tipo de plataforma	Middleware	Toolkit	Middleware	Monitor
Arquitectura	Centralizada em cada projecto	Configurável	Federada	Centralizada
Âmbito	Projectos cientificos	Empresarial ou Académico	Qualquer	Empresarial
Acesso	Limitado - necessidade de certificação Gratuito	Necessidade de montar rede Gratuito	Aberto e gratuito	Necessidade de montar rede Versão Gratuita
Modelo de programação	Bag-of-tasks	Bag-of-tasks	Bag-of-tasks ou MPI	Não definido
Funcionalidades	Distribuição de carga Sistema elaborado de créditos Validação fiável de resultados Interfaces utilizador	Segurança Gestão de recursos e de dados Protocolos de comunicação Detecção de falhas Portabilidade de processos	Descoberta e gestão de recursos Escalonamento de processos Virtualização de aplicações Justiça através de rede de favores	Distribuição de carga Escalonamento de processos Gestão de quotas de acesso Monitorização de rede e de recursos

2.2. Processamento Multimédia Distribuído e Paralelo

Esta área é de grande interesse para este trabalho uma vez que os conteúdos multimédia são de uma forma geral de grande dimensão e necessitam de elaboradas técnicas e algoritmos de compressão para se tornarem viáveis para utilização no dia-a-dia.
Esta redução de tamanho, é conseguida à custa de uma enorme intradependência da informação multimédia o que dificulta a sua segmentação em blocos básicos passíveis de serem processados em paralelo. Esta intradependência é conseguida codificando apenas diferenças entre imagens ou amostras de som relativas a um ponto independente que contenha um segmento da informação completo e independente dos restantes. No caso do vídeo, como de forma geral as imagens apresentam reduzidas diferenças entre elas, excepto quando existem transições abruptas de cena, é possível obter uma grande redução na quantidade de informação a transmitir.
De seguida apresentam-se alguns estudos e soluções relativos a esta área.

Yang et al. [14] apresenta uma arquitectura baseada em clusters Linux interligados pelo *toolkit* Globus e monitorizado pelo Sun Grid Engine, para o processamento em paralelo de uma aplicação de *ray tracing*. São ainda feitos testes que demonstram que efectivamente a distribuição da carga por vários computadores acelera a obtenção de resultados e são ainda feitos estudos para identificar qual é a melhor relação entre o número de processadores envolvidos e o tempo final do processo.

Akramullah et al. [15] descreve uma implementação de um codificador de vídeo MPEG-2 em tempo real, usando uma aproximação de paralelismo de dados, explorada ao nível de cada frame individualmente e que corre sobre o Intel Paragon XP/S, um multi-processador com memória distribuída e arquitectura MIMD com MPI.
O funcionamento geral do algoritmo consiste em dividir cada frame em vários macroblocos que são delegados para cada processador e que comunica com os restantes para fazer uma correcta compensação de movimento nas fronteiras do seu macrobloco.
São apresentados também alguns mecanismos de controlo do débito de cada canal e de adaptação dinâmica das matrizes de quantização.

Ewerth et al. [16] propõe a aplicação de computação *Grid* e tecnologia de *WebServices* para análise de conteúdo multimédia nomeadamente detecção de cortes, texto, faces, movimentos, etc.
Neste caso o objectivo do trabalho é a detecção de transições num vídeo, um processo moroso que consiste em comparar diferenças de histogramas numa sequência de imagens. O algoritmo desenvolvido é composto por um serviço de gestão do fluxo vídeo de entrada e do número de tarefas a serem submetidas à *Grid* (com base nas dimensões do vídeo e débito do vídeo), um outro que faz o redireccionamento dos fluxos de cada segmento de vídeo para nós com recursos disponíveis e um terceiro que procede então à execução do algoritmo de detecção de cortes.
Os resultados experimentais apontam para desempenhos notáveis face ao processamento num só nó.

Kola et al. [17] propõe soluções para o problema dos sistemas de partilha de ciclos actuais não escalarem de forma satisfatória em aplicações *data-intensive*, como acontece no processamento de vídeo. A principal diferença para as soluções existentes é que neste sistema a transferência e processamento de dados são considerados como actividades diferentes, apesar de um surgir em seguimento do outro.
Uma rede de dependências é construída com este propósito garantindo não só a ordem pela qual as tarefas se desenrolam mas também permite determinar de forma mais precisa e eficiente o aparecimento de falhas no sistema e seu posterior tratamento.
São ainda sugeridos diferentes modelos de transferência de dados até aos nós computacionais e ainda um *case study* específico do armazenamento de 500 Terabytes de vídeo miniDV.

Parallel Horus [18] é uma biblioteca de programação para clusters que permite que programadores implementem aplicações multimédia paralelizadas através da escrita de programas totalmente sequenciais. Um conjunto de tipos de dados multimédia e operações primitivas de processamento e análise de imagens (Image Algebra) suportam a paralelização recorrendo a MPI para troca de mensagens. Com vista a optimizar o desempenho do sistema é feita uma subsequente análise em tempo de execução com vista a optimizar o programa e reduzir a comunicação entre processos a um mínimo (*lazy parallelization*).

Esta iniciativa teve origem na necessidade de analisar e detectar padrões suspeitos em imagens de vídeos de câmaras de vigilância, portanto os requisitos de escalabilidade e desempenho são de crucial importância principalmente tendo em conta que o de volume de informação está na ordem de dezenas de petabytes.

Outros trabalhos referentes a processamento distribuído ou paralelo (em multiprocessadores) de informação multimédia estão em Li et al. [19] que demonstra várias estratégias de codificação de vídeo em paralelo num sistema multiprocessador, Fellow et al. [20] que dá uma explicação extensa sobre sistemas distribuídos multimédia e Little et al. [21] que apresenta algumas considerações sobre a composição e comunicação de objectos multimédia em diversos tipos de redes.

2.3. Adaptação de Aplicações

A adaptação de aplicações é parte integrante do Ginger que tem como um dos seus objectivos a adaptação automática de programas não modificados para serem executados em sistema de partilha de ciclos.

Huang et al. [23] descreve um processo de conversão semi-automática de programas C para serem executados em serviços Triana. Apresenta duas ferramentas para o efeito: o JACAW que é uma ferramenta de encapsulamento de código baseada na Java Native Interface e que gera uma interface Java para qualquer rotina C; o MEDLI assiste o utilizador na descrição dos mapeamentos entre os tipos do Triana e os tipos C envolvidos na chamada de uma rotina em particular.

Sneed [26,27] identifica formas e técnicas de encapsulamento e descreve um processo de reengenharia de interfaces de um programa *legacy* escrito em Cobol, que extrai toda a informação necessária à execução de cada módulo do programa. Desta forma cada módulo (mais o *wrapper*), torna-se independente do programa e pode ser utilizado por outros módulos, locais ou remotos.

De Lara et al. [28] sugere extensões aos modelos tradicionais de replicação (optimistas e pessimistas), nos quais as réplicas além de um determinado número de versão possuem

também um nível de fidelidade relativo à versão original. No caso de uma imagem, esta métrica pode ser por exemplo o factor de compressão ou a resolução da mesma.
O cenário alvo é a edição de documentos em dispositivos móveis ligados a um repositório central. De modo a aumentar a flexibilidade deste sistema, estes documentos são decompostos em variados componentes multimédia (texto, páginas, imagens, gráficos, etc.), sendo feita uma adaptação posterior conforme a largura de banda disponível e enviadas versões de baixa fidelidade e dimensão mais reduzida para o dispositivo.
Além da redução de tamanho dos componentes, o aumento da granularidade permite adoptar uma postura *lazy* e só transmitir partes do documento quando estas são realmente necessárias além de reduzir possíveis conflitos entre utilizadores a editar o documento de forma concorrente.
O artigo apresenta ainda uma implementação (CoFi – *consistency and fidelity*) aplicada sobre o Outlook e o Powerpoint através de extensões aos mesmos e assente sobre um *proxy* que gere de forma transparente a comunicação com o sistema replicado.

No seguimento do artigo anterior, De Lara et al. [29] entra um pouco mais detalhe nos mecanismos de adaptação a aplicações, nomeadamente através do uso de *API's* exportadas por estas.
A abordagem gira em torno de um mecanismo de adaptação iterativa a aplicações que suporta adaptação ao nível de dados através de técnicas de decomposição e transformação de componentes (como vimos anteriormente) e também ao nível de controlo do comportamento do programa através da definição de políticas – por exemplo, retornar controlo ao utilizador quando todo o texto for carregado. É importante referir ainda, que neste sistema não existe qualquer alteração às aplicações.
O artigo descreve ainda um sistema de adaptação iterativa a aplicações de produtividade em Linux e Windows, denominada Pupeteer. Este sistema comporta-se de forma semelhante ao CoFi [28], mas além da utilização de "proxies" no acesso aos repositórios de dados fornece ainda uma série de módulos (núcleo, "drivers", transcodificadores e políticas) que uniformizam as diferentes *API's*, tipos de documentos e plataformas onde as aplicações podem ser executadas. Com esta simplificação a implementação de políticas torna-se bastante mais cómoda já que uma política *à priori* funcionará de forma igual para qualquer uma das aplicações.

Um resultado particularmente interessante é que além das *API's* do Office (COM) e OpenOffice (Corba) serem bastante diferentes, nota-se que a adaptação de aplicações nunca esteve no pensamento dos engenheiros que a desenharam, levantando grandes desafios e mesmo impossibilidades para o Pupetter.

4. Análise

As soluções apresentadas no ponto 2.1. propõem *middlewares* e ferramentas para o desenvolvimento de aplicações em ambientes distribuídos, grande número destas dando especial ênfase à utilização ciclos livres de computadores comuns. Algumas delas concentram-se em maximizar o uso destes recursos, outras propõem elaborados esquemas de créditos para aferir a justiça no acesso aos recursos e outras tem ainda como objectivo fornecer ferramentas configuráveis com as quais se possa construir aplicações que explorem computação *Grid* ou outras ferramentas mais elaboradas.
No entanto nenhuma se preocupa em suportar transparentemente aplicações já existentes, explorando técnicas de adaptação das mesmas a estas infrastruturas, com vista a acelerar o seu tempo de resposta trazendo as potencialidades da computação *Grid* até ao utilizador comum. Além desta limitação a ausência de ferramentas que simplifiquem o processo de descrição e decomposição do problema de partida em sub-problemas e subsequente classificação e escalonamento de cada um destes, levantam barreiras altas de entrada para muitas aplicações de uso comum.
Acima de tudo estes projectos não foram ainda capazes de fornecer uma infrastrutura descentralizada, suficientemente genérica e de acesso verdadeiramente livre, que permita a execução distribuída de aplicações desktop num ambiente heterogéneo como a Internet, juntando o sucesso de aplicações P2P de partilha de ficheiros como o BitTorrent e aplicações de partilha de ciclos como o SETI@home.

5. Arquitectura/soluções proposta

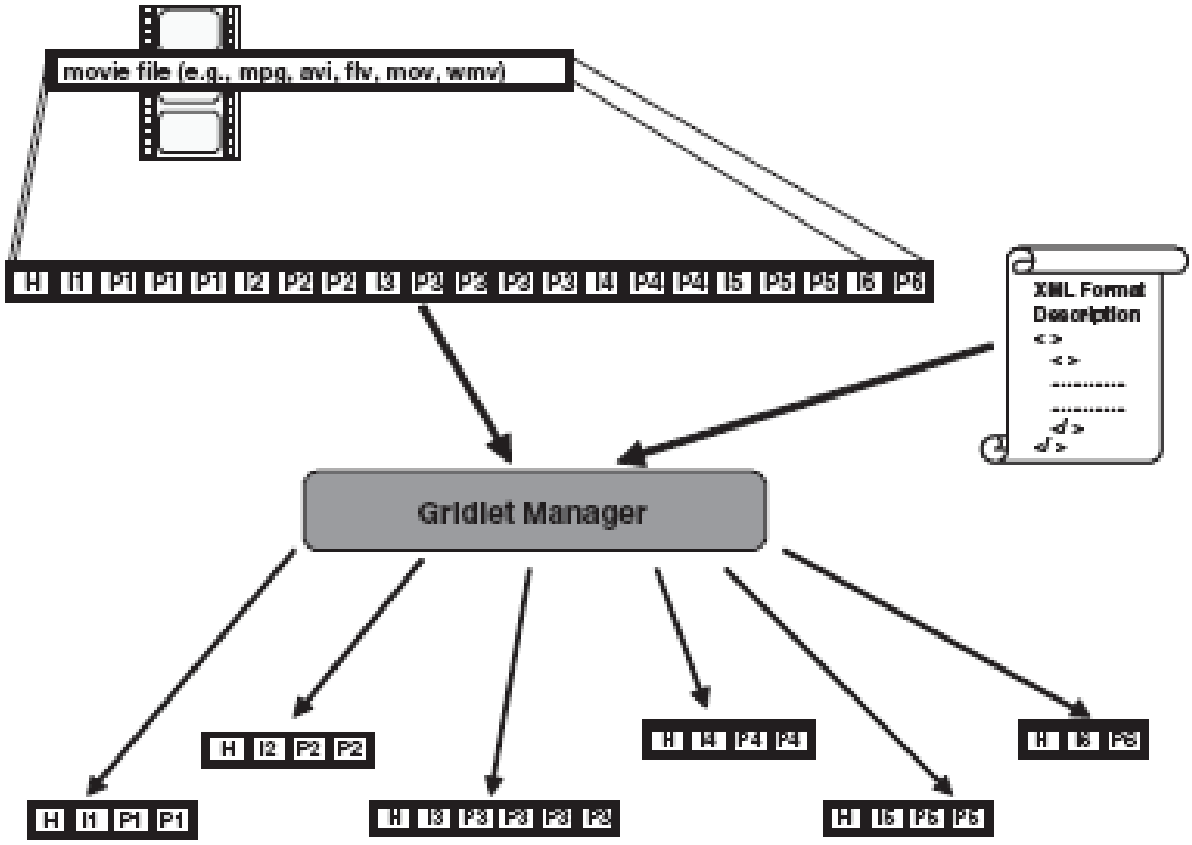

Figura 2: Processo de criação de *gridlets* a partir de um ficheiro de vídeo

A solução proposta baseia-se no modelo aplicacional apresentado em GINGER [5] que gira à volta do conceito de *Gridlet*, uma unidade elementar que descreve uma porção de computação e as operações que devem ser efectuadas nos dados contidos na mesma. Desta forma a *Gridlet* é a moeda que controla a economia por trás da *Grid*.

A figura 2 apresenta o processo de criação de novas *Gridlets* com base num ficheiro de vídeo. A divisão de um ficheiro desta natureza não pode ser arbitrária pois existe intra-dependência entre os dados. O *Gridlet Manager* (GM) é responsável pela partição inteligente dos dados e pela adição de outra informação que seja compreensível pelas aplicações que fazem o verdadeiro trabalho. Para tal o GM baseia-se no conteúdo de um documento XML que descreve o formato, explicita pontos de partição e de reassemblagem das *Gridlets* e outros cabeçalhos essenciais. Existirá uma fase inicial de adaptação da aplicação de forma a escolher os descritores correctos seguido da criação das várias *Gridlets*, distribuição na rede para potencialmente vários clientes, processamento remoto e finalmente reassemblagem de resultados na fonte, novamente com base nos descritores escolhidos. Todo este trabalho é feito em *background* dando a

ilusão que a aplicação está a correr localmente, quando na verdade grande percentagem do trabalho (ou mesmo 100%) está a ser feito em máquinas remotas.

O trabalho será dividido em 3 grandes módulos. De seguida iremos apresentar objectivos e problemas a resolver em cada um deles:

Adaptação da aplicação

Neste módulo pretende-se encontrar que descritores serão necessários para adaptar a aplicação ao ambiente distribuido. Vai consistir numa emulação da linha de comandos que analisará a aplicação invocada e os argumentos passados à mesma. Com base neste comando irá ser procurado um descritor da aplicação que por sua vez irá retornar os descritores dos formatos que irão ser utilizados. Ex: >app in.a out.b – será necessário primeiro encontrar um descritor para app que por sua vez irá dizer que o formato de divisão é indexado pela extensão do argumento 1 e o formato de reagreagação estará na extensão argumento 2. Ficheiros, descritores e outra informação são passados ao módulo seguinte que construirá as gridlets.
Obviamente que este é um caso extremamente simples, na realidade cada aplicação tem a sua forma distinta de receber argumentos que pode ser algo complexa. Como objectivo, pretende-se criar um método relativamente simples e completo com base em descritores de aplicação em XML, que permitam tratar todos os casos possíveis ou grande parte deles.

Divisão e Reagregação:

O foco do trabalho será dado em grande parte a este módulo. Todos os formatos com os quais o GiGi se irá deparar suportam uma divisão semânticamente correcta com base nalguma variável, habitualmente o tempo. Pontos específicos no ficheiro explicitam fronteiras de independência de conteúdo que serão usados pelo GiGi como pontos de divisão e reagragação de gridlets – designados em vídeo de *keyframes*. É importante notar que a localização destes pontos não é controlável pelo GiGi de forma que dois vídeos com a mesma duração e mesmo número de *peers* podem resultar em cenas de tamanho totalmente dispar.

O grande desafio a resolver neste módulo é o simples facto dos formatos multimédia exibirem uma forte intradependência entre campos do conteúdo. Um exemplo imediato

é o cabeçalho que tipicamente contêm o tamanho e localização de diferentes regiões do ficheiro. A divisão têm de ter em conta estas dependências e outras condições para produzir ficheiros mais reduzidos mas que continuam válidos aos olhos de qualquer "demuxer".

Inicialmente pensou-se em fazer uma *schema* que descrevesse sucintamente o formato com campos, tipos de dados, sequências, etc., havendo métodos de divisão e reagregação num ficheiro separado que usariam referências aos campos descritos na *schema*. No entanto este seria um método demasiado verboso e que inevitavelmente levaria a incompatibilidades sempre que se pretendesse actualizar a *schema*, por necessidade de corrigir tanto o divisor como o reagregador.

Mais simples, seria relacionar as funções de divisão e de reagregação da seguinte maneira:

$$\mathbf{F_D(f,d) = s^+,\ F_R(s^+,d) = f}$$

(F_D =função divisão, F_R =função reagragação, f=ficheiro, d=descritor, s=segmento de ficheiro)

Pela relação (1) entende-se que através de um único descritor é possível compor ambas as funções. Para melhor compreensão apresenta-se o seguinte exemplo para um formato hipotético em xml:

Ficheiro em formato hipotético

```
<a size="2">
    <b size="3" start="1">
        <c t="1" i="1" fp="t">a</c>
        <c t="2" i="2" fp="t">b</c>
        <c t="3" i="3" fp="t">c</c>
    </b>
    <b size="5" start="4">
        <c t="2" i="4">d</c>
        <c t="1" i="5" fp="t">e</c>
        <c t="2" i="6" fp="t">f</c>
        <c t="1" i="7">g</c>
        <c t="3" i="8">h</c>
    </b>
</a>
```

Legenda: *@fp* = ponto de fronteira, *@t* = pista, *@i* = índice

Estabelecem-se as seguintes regras a este formato:

1. *size* indica o número de elementos filhos
2. *start* indica o indice de *@i* do primeiro elemento <*c*>
3. os elementos <*c*> duma mesma pista tem de estar ordenados crescentemente pelo atribuito *@i*.
4. um elemento <*c*> que não seja ponto de fronteira têm de vir depois de um elemento <*c*> que seja ponto de fronteira e esteja na mesma pista.
5. o ponto de fronteira a ser usado na divisão tem de estar na pista 1

Com base nestas regras construimos uma rede de dependências entre os diferentes atribuitos e elementos e assentando um único ponto de fronteira no segundo elemento <*c*> da pista 1, dividimos o ficheiro em 2 segmentos:

Segmento 1	**Segmento 2**

`<a size="2">`
`  <b size="3" start="1">`
`  <c t="1" i="1" kp="t">a</c>`
`  <c t="2" i="2" kp="t">b</c>`
`  <c t="3" i="3" kp="t">c</c>`
`  </b>`
`  <b size="2" start="4">`
`  <c t="2" i="4">d</c>`
`  <c t="3" i="8">h</c>`
`  </b>`
`</a>`
	`<a size="1">`
	`  <b size="4" start="5">`
	`  <c t="1" i="5" kp="t">e</c>`
	`  <c t="2" i="6" kp="t">f</c>`
	`  <c t="1" i="7">g</c>`
	`  </b>`
	`</a>`

Note-se que os atribuitos *@size* estão coerentes com o número de filhos conforme as regras. Veja-se também que pela regra 2 e 3, atributo *@i* dos elemento <*c*> do 2º elemento <*b*> foram acertados. Mais importante que isso pela regra 4 e 5 o último elemento <*c*> da pista 3 teve de ser deslocado para o segmento 1 para ficar junto do seu ponto de fronteira e naturalmente o número de elementos <*b*> aumentou por via da divisão ter ocorrido nesse ponto.

Suponha-se agora que esta aplicação recebe como entrada um ficheiro neste formato e simplesmente retorna uma cópia do mesmo. Recebidos os segmentos na fonte e novamente com base nas regras estabelecidas, pode-se efectuar a seguinte reagregação:

```
<a size="3">
    <b size="3" start="1">
        <c t="1" i="1" fp="t">a</c>
        <c t="2" i="2" fp="t">b</c>
        <c t="3" i="3" fp="t">c</c>
    </b>
    <b size="1" start="4">
        <c t="2" i="4">d</c>
    </b>
    <b size="4" start="5">
        <c t="1" i="5" fp="t">e</c>
        <c t="2" i="6" fp="t">f</c>
        <c t="1" i="7">g</c>
        <c t="3" i="8">h</c>
    </b>
</a>
```

Por comparação com o ficheiro inicial denota-se que a apesar da ordem se manter o agrupamento não é consistente. Se as regras forem suficientes e completas este ficheiro apesar de sintáticamente diferente, é semânticamente igual; é esse o nosso objectivo. Claro que é possível adicionar mais restrições para o ficheiro ser igual, mas como irá ser interpretado da mesma forma não se põe a necessidade.

Grande parte dos formatos multimédia dão flexibilidade aos *muxers* de pôr os campos onde entenderem e muitas vezes esta ordem acaba por estar dependente até das especificidades do codec áudio ou vídeo que o formato transporta. Iremos debruçar-nos sobre estas questões no decorrer do trabalho apesar de não serem prioritárias.

Como no ponto de adaptação da aplicação, os formatos *container* que iremos descrever são extremamente mais complexos e têm redes de dependências bastante maiores. De qualquer forma o exercício anterior é uma amostra do que se passa num formato multimédia, a organização é em árvore e pode haver dependências de pais para filhos e vice-versa. Os formatos que escolhemos são o Matroska e o AVI, que são dois formatos *standard* de transporte de vídeo e áudio e que estão bem documentados, além de serem relativamente simples.

Assim e visto que praticamente todos os formatos estão estruturados em árvore, a abordagem que irá ser tomada será de carregar o descritor e por sua vez ler o necessário do ficheiro para fazer uma árvore XML paralela. Nesta árvore cada elemento tem o deslocamento para o respectivo campo no ficheiro binário. Através da descrição das relações e dependências em XPath pode-se simplesmente percorrer esta árvore e encontrar a exacta localização do campo. Assim quando o número de elementos de <*b*> mudar como sabemos que <*a*> tem uma dependência para <*b*> (visto que `<a size="count(b)">`) como sabemos a localização deste elemento podemos alterá-lo

conforme. Um exemplo de um descritor para o formato hipotético descrito anteriormente está no anexo 1.

Além da aplicação principal que é a transcodificação de vídeo, caso haja tempo irá ser feita uma tentativa de segmentar uma cena de ray-tracing. Um ficheiro xml descreve uma cena e posições de uma câmara e o descritor irá dividir a cena em vários períodos. A aplicação processa a cena e gera um vídeo AVI, que é reagregado para compor a cena completa.

Distribuição de gridlets:

Este ponto irá ser dado uma enfâse mais reduzida, como tal questões como a descoberta de *peers* e a estruturação em overlay não irão ser tidas em conta. No entanto há aspectos relevantes que iremos ver como o encontro de uma "moeda" para a Grid por forma a não haver monopólio de recursos por parte de um utilizador. Esta moeda será relativa a um tempo e a um número de ciclos de relógio. Quantos mais ciclos de relógio um cliente partilhar mais ciclos de relógio podem no futuro ser usados pelo mesmo. O tempo é tido em conta para possibilitar clientes com máquinas com performance mais reduzida de também serem recompensados, apesar de com um peso menor.

Em termos de implementação pensou-se em primeiro lugar uma arquitectura P2P com o protocolo Pastry, no entanto como este ponto é secundário vamos fazer a distribuição através de RMI para controlo e transissão de ficheiros em HTTP ou FTP, com descoberta de peers através de broadcast na rede local.

[30] sugere um modelo de execução semelhante ao Ginger, através de uma descrição textual de formatos, com vista a identificar possíveis pontos de partição. No entanto os mecanismos fornecidos são extremamente mais limitados que a nossa abordagem, não tendo em conta as relações estruturais e sintácticas entre blocos num ficheiro, característica de um grande número de formatos como é o caso do formatos MPEG. Para suportar estes casos, recorre-se à invocação de programas exteriores ao middleware que fazem o trabalho de divisão e posterior agregação.

6. Planeamento

Tarefa	Duração
Criar formato de descrição	20 dias
Criação de árvore de dependências	30 dias
Divisão em gridlets	25 dias
Reagregação de gridlets	20 dias
Distribuição de gridlets	3 dias
Adaptação de aplicações	10 dias
Resolução de problemas	15 dias
Aplicação ray-tracing	15 dias

7. Conclusões

Nos últimos 10 anos a computação *Grid* passou da teoria à prática e variadíssimos estudos e propostas têm aparecido para preencher este espaço, de enorme potencial mas de grande dificuldade para o utilizar em pleno.

Projectos científicos ou com propósitos empresariais/académicos perfazem ainda a grande maioria das infrastruturas *Grid* actuais e apesar de algumas destas serem acessíveis a qualquer pessoa, habitualmente são pouco versáteis e inviáveis por razões monetárias ou burocráticas.

O crescimento do interesse público na partilha de vídeos é observável no aumento claro de tráfego para alguns sites que disponibilizam este tipo de conteúdo mas levanta algumas questões técnicas. Uma delas é relativa à compressão vídeo, uma vez que os vídeo que as máquinas de filmar convencionais produzem, têm um *bitstream* bastante superior às ligações de banda larga actuais. Como a compressão ou transcodificação são processos longos, a execução paralela usando ciclos livres de máquinas de outros utilizadores na Internet surge então como uma boa alternativa no sentido de acelerar a produção de vídeos de menor dimensão e com perdas de qualidade aceitáveis.

Com este desafio em mente o Ginger apresenta-se com uma arquitectura flexível, centrada num conceito simples de *Gridlet*, capaz de se adaptar não só a ambientes altamente diversificados como é o caso da Internet, no contexto do utilizador comum, como também a diferentes aplicações (concretamente um codificador no caso do Ginger-Video-3D), abstraindo quanto possível as dificuldades presentes na execução de código em paralelo.

8. Referências

[1] J. Joseph and C. Fellenstein, “Introduction to Grid Computing,” Apr. 2004; http://www.informit.com/articles/article.aspx?p=169508&seqNum=5.

[1] David P. Anderson, “Designing a Runtime System for Volunteer Computing,” Nov. 2006; http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/sc/2006/2700/00/2700toc.xml&DOI=10.1109/SC.2006.24.

[3] N. Andrade et al., “OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing,” *9th Workshop on Job Scheduling Strategies for Parallel Processing*, 2003; http://citeseer.ist.psu.edu/andrade03ourgrid.html.

[4] A. Goldchleger et al., “InteGrade: object-oriented Grid middleware leveraging the idle computing power of desktop machines,” *Concurrency and Computation: Practice and Experience*, vol. 16, 2004, pp. 449-459.

[5] L. Veiga, “GiGi: An Ocean of Gridlets on a "Grid-for-the-Masses",” May. 2007; http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/ccgrid/2007/2833/00/2833toc.xml&DOI=10.1109/CCGRID.2007.54.

[6] J. Miller, “YouTube Comprises 10% Of All Internet Traffic,” Jun. 2007; http://www.webpronews.com/topnews/2007/06/19/youtube-comprises-10-of-all-internet-traffic.

[7] V. Lo et al., *Cluster Computing on the ºFly: P2P Scheduling of Idle Cycles in the Internet*, 2004;

http://citeseer.ist.psu.edu/lo04cluster.html.

[8] M. Litzkow et al., “Condor-a hunter of idle workstations,” *Distributed Computing Systems, 1988., 8th International Conference on*, 1988, pp. 104-111.

[9] “Globus: a Metacomputing Infrastructure Toolkit,” Jun. 1997; http://hpc.sagepub.com/cgi/content/abstract/11/2/115.

[10] A. Barak et al., “An organizational grid of federated MOSIX clusters,” *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, 2005, pp. 350-357 Vol. 1.

[11] A.R. Butt et al., “Java, peer-to-peer, and accountability: building blocks for distributed cycle sharing,” *Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium - Volume 3*, San Jose, California: USENIX Association, 2004, pp. 13-13.

[12] A. S. Grimshaw, “Legion-a view from 50,000 feet,” Aug. 1996; http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/hpdc/1996/7582/00/7582toc.xml&DOI=10.1109/HPDC.1996.546177.

[13] W. Gentzsch, “Sun Grid Engine: Towards Creating a Compute Power Grid,” May. 2001; http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/ccgrid/2001/1010/00/1010toc.xml&DOI=10.1109/CCGRID.2001.923173.

[14] Chao-Tung Yang, Chao-Tung Yang, and Chuan-Lin Lai, "Apply cluster and grid computing on parallel 3D rendering," *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, 2004, pp. 859-862 Vol.2.

[15] S.M. Akramullah, I. Ahmad, and M.L. Liou, "A data-parallel approach for real-time MPEG-2 video encoding," *J. Parallel Distrib. Comput*, vol. 30, 1995, pp. 129-146.

[16] Ralph Ewerth, "Grid Services for Distributed Video Cut Detection," Dec. 2004; http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/ismse/2004/2217/00/2217toc.xml&DOI=10.1109/MMSE.2004.47.

[17] G. Kola, T. Kosar, and M. Livny, "A fully automated fault-tolerant system for distributed video processing and off-site replication," *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, Cork, Ireland: ACM, 2004, pp. 122-126.

[18] F. Seinstra et al., "High-Performance Distributed Video Content Analysis with Parallel-Horus," *Multimedia, IEEE*, vol. 14, 2007, pp. 64-75.

[19] Ping Li et al., "Design and implementation of parallel video encoding strategies using divisible load analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 15, 2005, pp. 1098-1112.

[20] V. Li, V. Li, and Wanjiun Liao, "Distributed multimedia systems," *Proceedings of the IEEE*, vol. 85, 1997, pp. 1063-1108.

[21] T. Little, T. Little, and A. Ghafoor, "Network considerations for distributed multimedia object composition and communication," *Network, IEEE*, vol. 4, 1990, pp. 32-40, 45-49.

[22] L. Navarro et al., "Invasive patterns: aspect-based adaptation of distributed applications," Nov. 2007.

[23] Yan Huang et al., "Wrapping legacy codes for Grid-based applications," *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 7 pp.

[24] Taylor et al., "Triana Applications within Grid Computing and Peer to Peer Environments," *Journal of Grid Computing*, vol. 1, Jun. 2003, pp. 199-217.

[25] F. Seinstra et al., "High-Performance Distributed Video Content Analysis with Parallel-Horus," *Multimedia, IEEE*, vol. 14, 2007, pp. 64-75.

[26] Sneed, "Encapsulation of legacy software: A technique for reusing legacy software components," *Annals of Software Engineering*, vol. 9, May. 2000, pp. 293-313.

[27] Harry M. Sneed, "Program Interface Reengineering for Wrapping," Oct. 1997; http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/wcre/1997/8162/00/8162toc.xml&DOI=10.1109/WCRE.1997.624591.

[28] E.D. Lara et al., "Collaboration and multimedia authoring on mobile devices," *Proceedings of the 1st international conference on Mobile systems, applications and services*, San Francisco, California: ACM, 2003, pp. 287-301; http://portal.acm.org/citation.cfm?id=1066116.1066126.

[29] E. de Lara et al., "Iterative adaptation for mobile clients using existing APIs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, 2005, pp. 966-981.

[30] K. van der Raadt, "APST-DV: A Practical Framework for Scheduling and Deploying Divisible Loads on Grid Platforms."

9. Anexos:

Anexo 1: descriptor de formato hipotético

```
<gigi-format name="xyz">
 <structure>
  <element name="a">
   <attribute name="l" value="count(child::b)" />
    <element name="b">
     <rules>
      <condition expr="count(c) lte 5"/>
     </rules>
     <attribute name="ns" value="count(child::c)" />
     <attribute name="si" value="child::c[1]/@i" />
     <element name="c" is-split-point="@kp='t' and @t=1">
      <rules>
       <dependence expr="if not(@kp) then (//c[@t=./@t and @kp='t' and @i
lt ./@i])[last()]" />
       <condition expr="parent::@si >= @i and parent::following-
siblings[first()]/@si > @i" />
      </rules>
      <attribute name="t" />
      <attribute name="i" value="number(../preceding-
sibling::b[last()]/c[last()]/@i) + count(preceding-sibling::c)+1" />
      <attribute name="kp" default-value="f" />
     </element>
    </element>
   </element>
  </structure>
</gigi-format>
```