Um Suporte para Adaptação Dinâmica de Arquiteturas Ubíquas

André Luiz G. Santos, Diego Leal, Orlando Loques

Instituto de Computação – Universidade Federal Fluminense (UFF)
Rua Passos da Pátria, 156, Bloco E, 3º andar – São Domingos – Niterói – RJ – Brasil

{asantos,dleal,loques}@ic.uff.br

Abstract

Ubiquitous computing environments are highly dynamic due to changes in the set of available resources and also because of changes of user's demands in different operational contexts. Thus, applications for these environments require dynamic-adaptation support mechanisms in order to meet requirements that vary along their operational time. In this paper we present dynamic-adaptation support mechanisms based on technologies that are widely available and adopted by real world product developers. Some experiments demonstrate that the proposed mechanisms meet performance and flexibility requirements that are essential for the implementation of real pervasive systems.

Keywords: Ubiquitous, Computing, Dynamic, Adaptation, Architecture, Configuration.

Resumo

Ambientes de computação ubíqua são altamente dinâmicos devido às freqüentes mudanças no conjunto de dispositivos disponíveis e de mudanças das necessidades dos usuários em diferentes contextos de operação. Dessa forma, as aplicações nesses ambientes requerem mecanismos de suporte à adaptação dinâmica para atenderem a requisitos que variam ao longo do tempo. Neste trabalho apresentamos um suporte para adaptação dinâmica baseado em tecnologias atuais, amplamente disseminadas e adotadas em âmbito comercial. Experimentos realizados demonstram que a proposta atende também a requisitos de flexibilidade e desempenho essenciais para a implementação de sistemas reais.

Palavras Chaves: Ubíqua, Computação, Arquitetura, Dinâmico, Adaptação, Configuração.

1. Introdução

A abrangência da computação ubíqua vem crescendo cada vez mais. Ela vislumbra um mundo onde usuários, movendo-se em diferentes contextos, interagem naturalmente com diferentes dispositivos computacionais para executar diversos tipos de tarefas. Desta forma, aplicações em ambientes de computação ubíqua podem utilizar recursos (dispositivos, serviços, aplicações, etc) cuja disponibilidade pode variar ao longo da execução da aplicação, caracterizando um cenário altamente dinâmico.

Como conseqüência da mobilidade, as aplicações em execução em um ambiente ubíquo apresentam um requisito inerente de adaptação dinâmica para atenderem às mudanças no contexto onde estão imersas como, por exemplo, a disponibilidade de novos dispositivos ou serviços, o tipo de tarefa em execução, a indisponibilidade de alguns recursos, e até mesmo a presença ou ausência de pessoas no ambiente. Além disso, a aplicação pode migrar, juntamente com seu usuário, por diferentes ambientes, reforçando a necessidade de adaptação para melhor uso dos recursos disponíveis. Desta forma, para possibilitar que as aplicações migrem e sejam também adaptadas a cada ambiente, de modo a dar continuidade às tarefas em execução, podem ser utilizadas técnicas e mecanismos de configuração dinâmica, tais como os descritos em [19, 21, 1, 22, 2, 5]. Através da configuração dinâmica é possível que uma aplicação evolua de uma configuração para outra sem causar impactos consideráveis nos serviços oferecidos ao usuário. Um exemplo de configuração dinâmica é adicionar um trans-codificador de vídeo em uma aplicação de vídeo sob demanda, durante a transmissão de um fluxo de vídeo, de modo a adaptá-lo às características de um *display* existente em um determinado contexto.

Visando atender os requisitos de aplicações ubíquas, adotamos o *framework* CR-RIO, que facilita a implementação de sistemas de softwares adaptáveis, descrito em [19, 10]. Esse *framework* permite que sistemas de software sejam desenvolvidos a partir de uma visão arquitetural, como uma composição de componentes e

interligações descritas explicitamente e separadamente, em oposição ao desenvolvimento em baixo nível, feito a partir da especificação de algoritmos e estruturas de dados dos componentes individuais. Essa estratégia incentiva a separação de interesses, permitindo tratar de modo diferenciado os requisitos funcionais, que representam a funcionalidade básica de uma aplicação, de seus requisitos não-funcionais, tais como o interfaceamento com dispositivos de E/S, protocolos de transferência de dados e outras preocupações ligadas a recursos ou serviços específicos, fortemente dependentes do contexto de execução, característica inerente em ambientes ubíquos.

Utilizando o *CR-RIO*, uma aplicação ubíqua pode ter sua funcionalidade básica descrita no nível arquitetural, sendo as preocupações não-funcionais, descritas em tempo de projeto através de contratos de alto nível [10, 14, 19]. Em tempo de execução, os contratos são utilizados para automatizar o gerenciamento das adaptações necessárias para prover os recursos, ou os serviços utilizados pela aplicação. Um suporte de execução padronizado permite que o mecanismo de gerenciamento de adaptações seja reutilizado na implementação de diferentes contratos associados a aplicações ubíquas.

Complementando o *framework*, de modo a prover os mecanismos de baixo nível necessários à efetivação das adaptações de aplicações, foi concebido um suporte denominado SDA-A (*Support for Dynamic Architecture-Adaptation*), descrito neste texto. Ele visa atender requisitos de flexibilidade e desempenho, típicos em ambientes ubíquos, permitindo que os componentes e interligações da arquitetura da aplicação sejam re-configurados de modo a adaptá-la a cada contexto de execução. No nível da implementação, a capacidade de re-configuração é obtida através do uso de técnicas de reflexão computacional para a alteração dos códigos das classes das quais são derivados os componentes da aplicação. Essa alteração, realizada de maneira transparente e automática, visa incluir nessas classes métodos auxiliares padronizados necessários para o suporte de mecanismos de interceptação, que são essenciais no suporte à adaptação. Essa solução utiliza tecnologias atuais amplamente disponíveis e adotadas em âmbito comercial, não requerendo o uso de mecanismos especializados ou de uso restrito. Em particular, o uso dessas técnicas permitem que aplicações usando a especificação Java *J2SE* ou usando a especificação Java JMX [28] (ver Seção 2.4) possam ser dinamicamente re-configuradas conforme desejado.

Este trabalho está estruturado na seguinte forma. Na seção 2 é apresentado o suporte à adaptação dinâmica para arquiteturas. Um exemplo de uso e medidas de desempenho, obtidas através de experimentos realizados utilizando o suporte proposto, são apresentados nas seções 3 e 4, respectivamente. A seção 5 apresenta alguns trabalhos relacionados e finalmente na seção 6 são apresentadas conclusões.

2. O Suporte Para Adaptação Dinâmica

Em ambientes de computação ubíqua os usuários podem interagir com aplicações em qualquer lugar e a qualquer momento. Como os ambientes ubíquos são caracterizados por serem altamente dinâmicos, as aplicações devem ser capazes de se adaptarem a diferentes ambientes bem como às suas modificações e, possivelmente, migrar juntamente com os usuários por diferentes contextos.

Para podermos caracterizar as necessidades básicas de um suporte a adaptação dinâmica de arquiteturas, usamos uma aplicação de Vídeo sob Demanda (*Vídeo on Demand* - VoD) simplificada (esta aplicação será apresentada em maiores detalhes na seção 3). A Figura 1 apresenta a aplicação VoD que possui um servidor, o qual provê uma fonte de vídeo de alta qualidade e pode usar um entre dois dispositivos[1] que podem estar disponíveis em um determinado ambiente para a reprodução do mesmo. Um dos dispositivos, tipo o POCKET-PC, possui limitações quanto ao formato do vídeo que pode ser reproduzido, sendo assim necessário usar um trans-codificador para adequar o fluxo às suas características. Em uma situação quando o usuário estiver sozinho num ambiente, o sistema poderá selecionar o monitor LCD de melhor qualidade, enquanto num ambiente com várias pessoas, o POCKET-PC deverá ser automaticamente selecionado, de modo a garantir a privacidade de seu proprietário. No primeiro caso, a saída do *módulo vídeo* deverá ser redirecionada para o monitor LCD. No segundo caso, deve ser usado um *trans-codificador*, que pode ser inserido dinamicamente na configuração, para adaptar a qualidade do vídeo e redireciona-lo para o POCKET-PC. Deve ser notado que o servidor de vídeo específico, a ser usado em cada configuração, também poderia ser selecionado dentre diversos servidores disponíveis, e.g., escolher o servidor de

[1] Assumimos neste artigo que todos os dispositivos utilizados nos ambientes ubíquos, possuem um módulo de comunicação (Executor) previamente instalado no mesmo de modo a facilitar as comunicações com os elementos do suporte descrito nesse artigo.

vídeo que ofereça o menor retardo ou a melhor banda de comunicação, ou mesmo uma ponderação das duas propriedades.

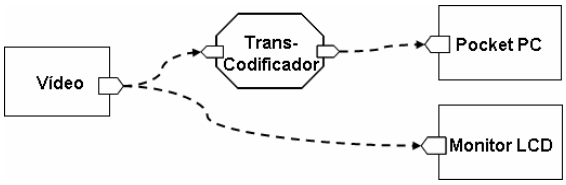

Figura 1 - Aplicação VoD Simplificada

Em outros contextos, diferentes dispositivos de visualização podem estar disponíveis e situações de "indisponibilidade" de recursos (por falha ou por estarem em uso por outras aplicações) também têm que ser previstas. Tendo em vista essas características de ambientes ubíquos, a tarefa de desenvolver uma aplicação VoD, como apresentada na Figura 1, torna-se árdua e complexa se não forem usados mecanismos de suporte adequados. Desta forma, para facilitar o desenvolvimento de aplicações para ambientes ubíquos, foi integrado ao *framework* CR-RIO um suporte especializado para adaptação dinâmica de arquiteturas denominado SDA-A (*Support for Dynamic Architecture-Adaptation*) o qual é proposto neste artigo. O *SDA-A* baseia-se no *design pattern Architecture Configurator*, descrito em [8, 18], que utiliza técnicas de Reflexão Computacional para prover os mecanismos de adaptação. Ele também provê primitivas de configuração que são usadas para colocar em execução e gerenciar arquiteturas de aplicações, descritas na linguagem de descrição de arquiteturas CBabel [29]. No *SDA-A* os requisitos funcionais são representados por *módulos* e os requisitos não funcionais por *conectores* (ver seção 2.1). Além disso, o comportamento requerido pela aplicação pode ser especificado utilizando um mecanismo de contrato de qualidade de serviço, também incluído no *framework* CR-RIO [19]. O suporte a contratos permite identificar, em tempo de execução, o serviço a ser ativado, bem como os recursos necessários para isso. O serviço pode ser colocado em prática utilizando o *SDA-A* (ver seção 3), o que pode implicar em adaptações na configuração da aplicação.

2.1. *Design Pattern Architecture Configurator*

O *design pattern Architecture Configurator*, utilizado na implementação do *SDA-A*, fornece a base para a implementação de configurações arquiteturais, fundamentada nos mecanismos de interceptação, encaminhamento e manipulação de requisições, permitindo a interligação dos *módulos* e *conectores* componentes de uma aplicação. Os *módulos*, neste caso, representam os requisitos funcionais de uma aplicação podendo ser implementados como objetos, procedimentos, funções ou programas executáveis. Os requisitos não-funcionais, que podem ser associados a questões de qualidade de serviço, são mapeados em *conectores*, os quais podem ter implementações similares a dos módulos, ou serem efetivados através de serviços de um *middleware* ou sistema operacional. Os *módulos* são interligados através dos *conectores* ou diretamente entre si. Essas ligações são definidas em função de pontos de interação específicos representados pelas *portas*. Estas são responsáveis pela ligação real entre componentes e/ou *conectores*. Existem dois tipos de *portas*: portas de saída, que são representadas pela requisição de um serviço, e portas de entrada, que são as assinaturas dos métodos disponíveis em um *módulo*, que podem ser encontradas na definição de sua interface.

Os mecanismos utilizados no *Architecture Configurator* são aplicados de forma transparente em relação aos elementos básicos da arquitetura, suportando propriedades como reutilização, abstração e separação de interesses. Em particular, eles permitem que *módulos* e *conectores* participantes da arquitetura possam ser implementados de forma autônoma e integrados de acordo com a configuração arquitetural desejada. No exemplo da Figura 1, o serviço de VoD e os componentes de visualização seriam associados a *módulos*, enquanto o elemento trans-codificador seria associado a um *conector*; conectores *default* utilizados na comunicação entre *módulos* não precisam ser explicitamente configurados na arquitetura.

2.2. Elementos que Compõem o SDA-A

O *SDA-A* é baseado em dois componentes: *Coordenador* e *Executor*, através dos quais as aplicações podem ser gerenciadas dinamicamente. Cada um deles possui responsabilidades bem definidas, além de possuir alguns métodos pelos quais eles podem interagir entre si; a Figura 2 ilustra um cenário onde o *Coordenador* interage com *N* executores.

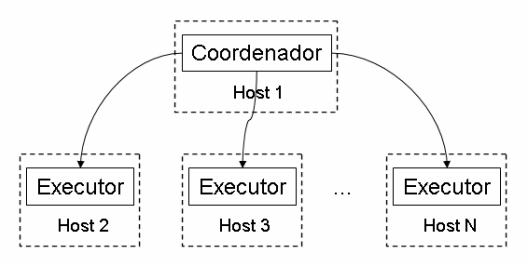

Figura 2 - Arquitetura do *SDA-A*

O *Coordenador* tem a responsabilidade de interpretar a descrição da arquitetura, elaborada em CBabel, e transformar essa descrição em informações que possam ser armazenadas em uma estrutura de dados. O *Coordenador* também disponibiliza uma interface *apply(...)*, para que as aplicações carregadas através do *SDA-A* possam ser gerenciadas externamente. Através do *Coordenador*, as aplicações podem obter informações sobre a configuração da arquitetura como, por exemplo, as interligações entre os elementos que compõem a aplicação. O *SDA-A* inclui apenas uma instância do *Coordenador*, que é responsável por delegar tarefas às instâncias remotas de executores.

O Executor tem a responsabilidade de efetivar localmente as operações designadas pelo *Coordenador*, como carregar um *módulo* ou *conector* (componente), ligar um determinado *componente* a outro, dentre outras funcionalidades que não serão detalhadas neste artigo. Em cada entidade, integrante do contexto computacional no qual os *componentes* de uma aplicação ubíqua são executados, deve existir uma instância do *Executor*. O *Executor* possui alguns métodos que são utilizados pelo *Coordenador* para gerenciar um *componente* da aplicação remotamente. Por exemplo, o método *loader(...)* carrega uma classe de um *componente* na entidade (caso a classe não tenha sido carregada anteriormente), cria uma instância da mesma e a registra localmente no *SDA-A*. O método *unloader(...)* descarrega um determinado *componente*, ou seja, remove todas as referências da instância (objeto), do *componente*, do *SDA-A* e libera o objeto para que o *Garbage Colector* da máquina virtual Java possa coletá-lo. O método *link (...)* faz a ligação de um *componente* (remoto ou local) a outro. O Executor também é responsável por adaptar as classes da aplicação do usuário de modo que elas possam ser gerenciadas pelo suporte (ver seção 2.4). Na seção 3 é apresentado um exemplo de uma aplicação gerenciada pelo *SDA-A*.

2.3. Gerenciamento Dinâmico

Desenvolvedores de sistemas distribuídos podem projetar aplicações, que serão gerenciados pelo *SDA-A*, sem ter o conhecimento do funcionamento interno do suporte oferecido por ele. Além disso, eles podem contar com recursos de interceptação de chamadas externas a outros componentes. Como mencionado na seção 2.1, os *módulos* da aplicação podem ser interligados através de *conectores*. Neste caso, a estrutura do *SDA-A* disponibiliza métodos para que os *conectores*, ou até mesmos os *módulos*, possam interceptar as chamadas que passam por eles (Ver Figura 4). Porém, para que os *componentes* desenvolvidos possam usufruir da capacidade de interceptação de chamadas, e ao mesmo tempo serem gerenciados pelo *SDA-A*, é necessário que eles possuam certos atributos e métodos (componentes de um suporte padronizado). Os atributos conterão informações referentes à localização do próximo *componente*, para onde uma chamada interceptada deve ser encaminhada. Os métodos fornecerão a capacidade de modificar as informações dos atributos, interceptarão chamadas, bloquearão o funcionamento de um *componente*, entre outras capacidades que, por brevidade, não serão discutidas neste trabalho.

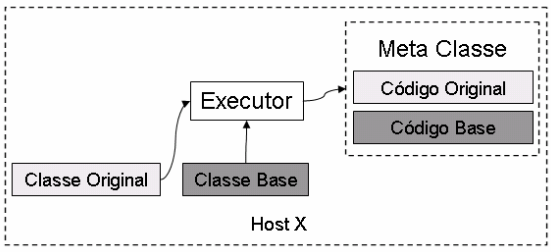

Figura 3 – Implantação de *Módulo* ou *Conector*

A Figura 3 apresenta o processo de implantação (carregamento) de um *componente* no *SDA-A*. O *Executor* localizado no *Host X* lê as informações da classe do usuário, denominada *Classe Original*, que representa o *componente* da aplicação. O mesmo *Executor* também lê a classe padronizada do suporte, denominada *Classe Base*, que contém todas as informações (métodos e atributos) necessárias para que um *componente* possa ser gerenciado no *SDA-A*. Obtidas todas as informações, o *Executor* cria uma nova classe, denominada Meta Classe, que contêm o conteúdo da classe original juntamente com o conteúdo da classe base, instancia um objeto dessa nova classe e o registra no *SDA-A*. Maiores detalhes referentes à Classe Base poderão ser encontrados em [26].

No início da seção 2, foi apresentada uma interceptação do fluxo de vídeo enviado para o dispositivo *POCKET-PC*, usada para adaptá-lo devido à restrição de formato de vídeo suportado pelo mesmo. Para facilitar a compreensão, assumiremos que o Servidor de Vídeo possui um *módulo* denominado "Vídeo" e o dispositivo móvel *POCKET-PC* possui um *módulo* denominado "*POCKET-PC*". O processo de trans-codificação (Figura 4) é totalmente transparente tanto para o *módulo "Vídeo"* quanto para o "*POCKET-PC"*, pois o *conector "Trans-Codificador"* que é inserido dinamicamente[2] entre o "*Vídeo"* e o "*POCKET-PC"*, intercepta as chamadas realizadas pelo *Vídeo* através dos métodos inseridos pelo suporte. Para o *módulo "Vídeo"*, a chamada do método "*showVideo()"* é realizada diretamente no "*POCKET-PC"* (o método *showVideo()* é responsável por apresentar no display o vídeo recebido). Porém ela é redirecionada para o "*Trans-codificador"* que converte o formato do vídeo e o encaminha para o "*POCKET-PC"*. Esse comportamento acontece devido à re-configuração da arquitetura, executada pelo *SDA-A*, no momento da inserção do "*Trans-codificador"* entre o "*Vídeo"* e o "*POCKET-PC"*.

Todas as chamadas oriundas do *módulo* "*Vídeo*" passam pelo método *_proxy_(...)* inserido pelo Executor. O Executor, durante a adaptação das classes do usuário, identifica as chamadas de serviços externos (através da descrição arquitetural da aplicação) e insere um trecho de código que redireciona tais chamadas para o método *_proxy_(...)* (a forma como é feito a inserção de códigos na classe do usuário é apresentada na seção 2.4). Uma vez que o *_proxy_(...)* recebe uma chamada ele a encaminha para o método *_foward_(...)* o qual a trata e identifica para onde ela deve ser encaminhada (a informação do destino de uma chamada interceptada é obtida quando se realiza a re-configuração da aplicação através das chamadas dos métodos "*_link_(...)*" e "*_unlink_(...)*" também inseridos pelo Executor). Desta forma, através do método *_foward_(...)* o usuário poderá definir um método o qual será acionado para interceptar e tratar a chamada do serviço (método *converteVideo( )* da Figura 4). A Figura 4 ilustra todo o processo de execução da chamada do serviço *showVideo ( )*.

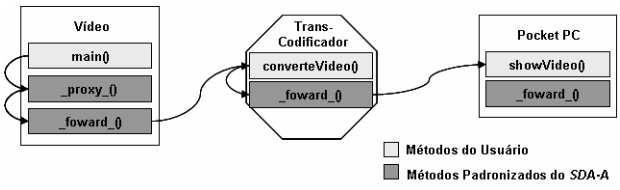

Figura 4 - Fluxo de execução da Trans-codificação

Na versão atual do suporte, os aspectos referentes à transferência dos estados dos *componentes* ficam a cargo do desenvolvedor. Contudo, já esta sendo realizado um estudo para que o suporte possa tratar este aspecto de automática.

2.4. Opções de implementação

Atualmente existem algumas propostas [21, 17] que oferecem recursos para manipulação de classes Java. Neste trabalho utilizamos o *Javassist* [9] para modificar (ou manipular) as classes desenvolvidas pelo programador, ou seja, adicionar os métodos e atributos que serão utilizados pelo *SDA-A* e/ou pelos componentes da aplicação. O *Javassist* é um *toolkit* baseado em reflexão para instrumentação de Java *bytecode*. Ele inclui um conjunto de ferramentas que oferece suporte no desenvolvimento de aplicações Java, possibilitando aos programadores criar novas classes, modificar o corpo de métodos, adicionar novos métodos a classes existentes, entre outras funcionalidades.

[2] O momento apropriado para adaptação da aplicação e o bloqueio dos *componentes* envolvidos na adaptação através dos métodos fornecidos pelo suporte, fica a cargo do desenvolvedor. O desenvolvedor poderá utilizar ou desenvolver uma aplicação externa para determinar o momento da realização das adaptações. Ver seção 3.

Com a utilização do *Javassist*, foram implementados dois tipos de gerenciamento das aplicações:

- Gerenciamento baseado no Java *J2SE*, que fornece uma opção voltada para aplicações que seguem a especificação padrão do Java. Ele faz uso apenas de recursos de reflexão estrutural do Java, juntamente com o suporte de comunicação *Remote Method Invocation* (RMI), para oferecer a capacidade de adaptação dinâmica e interceptação de chamadas para uma aplicação. O principal benefício deste gerenciamento é a sua baixa interferência, em termos de desempenho temporal, em relação ao Java J2SE (ver seção 4), minimizando os efeitos colaterais sobre aplicações que o utilizem.
- Gerenciamento baseado na especificação JMX, que tem como objetivo principal gerenciar as aplicações através dos recursos oferecidos pela especificação JMX [28]. O suporte realiza algumas adaptações nas classes da aplicação, tais como a criação das interfaces seguindo o padrão *MBean* incluído na especificação JMX. Através deste gerenciamento, o desenvolvedor pode automaticamente criar *módulos* (tipo *MBean*) da aplicação, com o intuito de oferecer serviços a outras aplicações. O principal benefício deste gerenciamento é a transformação de uma aplicação Java *J2SE* em uma aplicação Java seguindo a especificação JMX, que é uma tecnologia bastante disseminada no âmbito comercial. Deve ser notado que para *componentes* ou aplicações existentes, desenvolvidas seguindo a especificação JMX, não é necessário realizar modificações. Isso permite que a ampla quantidade de componentes e aplicações disponíveis, no contexto da tecnologia JMX, seja gerenciada pelo *SDA-A*.

Uma questão importante a ser observada é que todo o código, referente à interceptação e redirecionamento de chamadas, é inserido automaticamente e transparentemente pelo *SDA-A*, utilizando o *Javassist*. Ou seja, o *SDA-A* modifica o conteúdo das classes desenvolvidas pelo programador de forma a oferecer os recursos de interceptação aos componentes da aplicação. Um outro aspecto importante é que, para o desenvolvedor dos *módulos* da aplicação, as localizações dos serviços utilizados são totalmente transparentes (a tarefa de resolução de localização, fica a cargo do ambiente de suporte), deixando-o assim livre para dedicar-se ao desenvolvimento das funcionalidades dos *módulos*. Maiores informações sobre o mecanismo de interceptação do *SDA-A* poderão ser encontradas em [26].

3. Exemplo

Nesta seção apresentamos um exemplo mais detalhado da aplicação VoD, em um cenário de computação ubíqua, desenvolvido com base no *SDA-A*. Como mencionado na seção 1, foi utilizado o *framework* CR-RIO de modo a facilitar o desenvolvimento da aplicação incluindo seus requisitos de adaptação no ambiente ubíquo. O suporte desenvolvido ficou responsável pelas adaptações de baixo nível, determinadas por um contrato interpretado pelo *framework*. A aplicação VoD vislumbra um cenário onde o usuário transita por vários ambientes, sendo que cada ambiente possui características próprias em termos de recursos disponíveis; além disso, os usuários carregam sensores que informam suas localizações no ambiente. As informações, sobre as localizações dos usuários e sobre os recursos existentes nos ambientes, são obtidas através de um serviço de localização e descoberta de recursos (SL&D). Atualmente está sendo desenvolvido no contexto de nosso projeto [7], um *SL&D* com características integradas aos conceitos providos pelo CR-RIO. Mais detalhes sobre *SL&Ds* relacionados a ambientes ubíquos podem ser obtidos em [6, 23, 25].

A aplicação VoD (Figura 5) disponibiliza um vídeo que deve ser reproduzido em um dos dispositivos disponíveis no ambiente em que o usuário se encontra. O usuário pode transitar por diversos ambientes nos quais ele pode permanecer para executar tarefas cotidianas, incluindo assistir a um vídeo que é transmitido exclusivamente para ele. Cada ambiente possui dispositivos que podem ser usados para reproduzir o vídeo para o usuário, ou podem ser usados por outros usuários para outras finalidades. Para este exemplo, consideramos quatro tipos de dispositivos (Projetores, Monitores LCD, TV de Plasma e Pocket PC).

Com relação à reprodução do vídeo do usuário consideramos a possibilidade de presença de outros usuários no ambiente. Caso existam outros usuários no mesmo ambiente o vídeo deverá ser reproduzido em um dispositivo pessoal do usuário (tipo *POCKET-PC*). Para que esse tipo de dispositivo possa reproduzir o vídeo transmitido, será utilizado um *conector* (*trans-codificador*) responsável por trans-codificar o fluxo de vídeo para um formato compatível com o dispositivo (de MPEG para H261). Neste cenário o *framework* CR-RIO utiliza o *SL&D* para obter as informações referentes ao ambiente e à localização do usuário. O *SL&D* também fornece as informações de quais dispositivos estão aptos a reproduzir o vídeo do usuário em um determinado ambiente.

Utilizando o suporte a contratos do CR-RIO, foram definidos dois serviços de modo a atender as necessidades do usuário. O primeiro é um serviço onde o usuário está acompanhado por outras pessoas no ambiente.

Este serviço é considerado secundário devido à preferência do usuário pela reprodução do vídeo em dispositivos de maior qualidade (TV, Monitor ou Projetor). Ele visa atender a um requisito de privacidade do usuário, reproduzindo o vídeo em seu *POCKET-PC* pessoal. No segundo serviço, que é considerado preferencial, o usuário está só no ambiente e então o vídeo poderá ser reproduzido no melhor dispositivo disponível, segundo uma ordem de preferência previamente especificada; e.g., em primeiro lugar a TV de Plasma, seguido pelo Projetor e por último o Monitor LCD.

O funcionamento da aplicação VoD ocorre da seguinte maneira. Inicialmente o CR-RIO tenta estabelecer o serviço preferencial. Para tal tarefa, ele obtém as informações do ambiente onde o usuário se encontra, incluindo a existência de outros usuários no mesmo ambiente, e quais dispositivos estão disponíveis. Obtidas as informações, o CR-RIO verifica se o serviço preferencial pode ser estabelecido. Caso positivo, o serviço é implantado, utilizando o suporte proposto, e a fonte de vídeo envia o fluxo de vídeo para o melhor dispositivo disponível no ambiente onde o usuário se encontra (obedecendo a sua preferência). Caso o serviço não possa ser estabelecido, o CR-RIO procura estabelecer o serviço secundário.

O processo de implantação de um serviço, gerenciado pelo CR-RIO, através do *SDA-A*, ocorre da seguinte forma: após o CR-RIO verificar qual serviço poderá ser implantado, ele informa ao *Coordenador* do *SDA-A* (através da interface *apply(...)*) quais as configurações (ligações entre *componentes*) ou re-configurações arquiteturais que devem ser realizadas. Uma vez obtidas as informações, o *Coordenador* interpreta-as, identificando a localização dos *Executores* responsáveis pelos *componentes* que terão suas ligações (*link*) re-configuradas. No caso, se comunica com o *Executor* responsável pelo *módulo Vídeo*, e define a ligação do *vídeo* com o dispositivo do usuário. Caso o dispositivo seja o *POCKET-PC*, o *Executor* cria dinamicamente uma instância de um *trans-codificador*, que irá converter a qualidade do vídeo para uma qualidade mais apropriada e configura a ligação do *vídeo* com o *POCKET-PC* passando pelo *trans-codificador*. Esse processo de trans-codificação é necessário devido às limitações de hardware do dispositivo portátil, incluindo a baixa definição de sua tela. Quando as ligações dos *componentes* da aplicação são re-definidas, cada *componente* irá possuir as informações para onde o fluxo das chamadas deverão ser encaminhadas (como mencionado na seção 2.3).

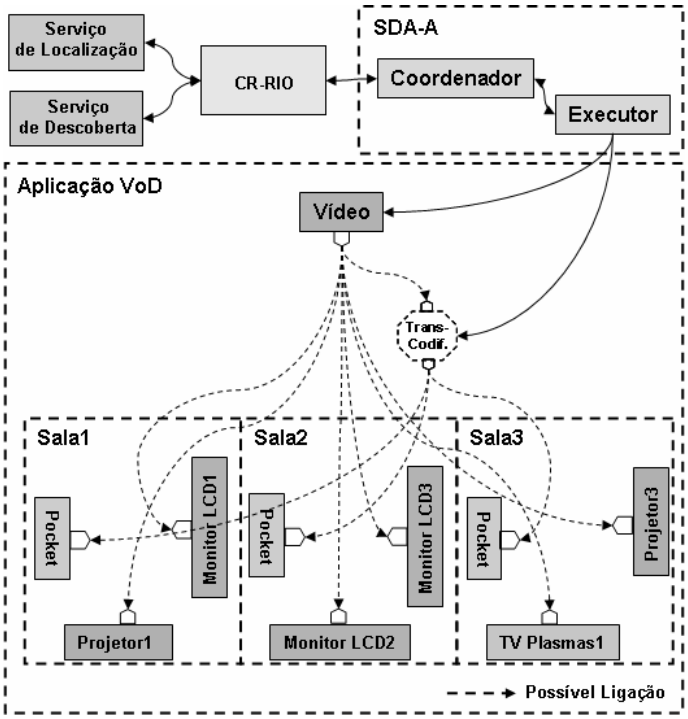

Figura 5 - Aplicação VoD no Ambiente de Computação Ubíqua

Quando o usuário muda-se de ambiente, o CR-RIO é notificado sobre a sua nova localização, e reinicia todo o processo de estabelecimento de serviço baseado no novo ambiente onde o usuário se encontra. Informações

sobre a linguagem de definição de contratos e sobre o processo de estabelecimento de serviços no CR-RIO podem ser encontradas em [10, 14, 19].

Em alguns testes realizados utilizando o suporte para gerenciar aplicações de vídeo sob demanda, deparamo-nos com problemas de perda e sincronização de *frames* de vídeo durante as adaptações da aplicação. Esses entraves não chegaram a perturbar significativamente a recepção do vídeo, mas poderiam ser inaceitáveis em algumas aplicações (e.g., em um cenário de tele-medicina, a realização de uma operação à distância). O trabalho [12] descreve mecanismos de sincronização, que poderiam ser integrados em nosso *framework* para obter um comportamento mais gracioso durante as adaptações.

4. Avaliação experimental

Para avaliar o *overhead* gerado pelo *SDA-A* na execução das aplicações e medir custo de cada adaptação efetuada nas aplicações, foi realizada uma bateria de testes (média aritmética de 10^4 invocações de um serviço de custo computacional zero) usando uma aplicação Cliente/Servidor (onde o Cliente realiza requisições, continuamente, de um serviço oferecido pelo Servidor) em uma rede local com as seguintes tecnologias:

- Especificação JMX do Java 1.5.0
- RMI do Java 1.5.0
- Gerenciamento do *SDA-A* utilizando Java *J2SE*
- Gerenciamento do *SDA-A* utilizando JMX

O parque computacional utilizado para os teste foi composto por 10 *hosts* (todos eles Pentium 4, 3.5Ghz com 506MB de RAM) rodando o sistema operacional Linux (Fedora Core 3). Foram realizados testes com 0 (o *Cliente* faz a chamada direto para o *Servidor*), 2, 4, 6, 8 *conectores* interpostos entre o Cliente e o Servidor. Todos os *módulos (Cliente e Servidor)* e *conectores* foram distribuídos em diferentes *hosts* da rede (no caso das execuções remotas). Observando os gráficos das Figuras 6 e 7 podemos ver os resultados obtidos com a execução local e remota respectivamente.

De acordo com a Figura 6, aplicações projetadas para a execução distribuída utilizando as tecnologias JMX e RMI, sendo executadas localmente, obtêm um desempenho inferior em relação às aplicações gerenciadas pelo *SDA-A*. O *SDA-A* possui um mecanismo que identifica quando um determinado *componente* realiza uma chamada de serviço para outro *componente* que se encontra no mesmo *host*. Desta forma, ele redireciona a chamada diretamente para o objeto (*componente*) na memória do *host*. Esse processo de identificação dos componentes locais contribui no desempenho da adaptação (por não precisar obter referências de objetos remotos) e no desempenho da execução (por não realizar comunicação remota, como feito no caso do JMX e do RMI). Com isso, o suporte (ambas as formas de gerenciamento) foi capaz de obter um melhor desempenho em relação ao JMX e ao RMI.

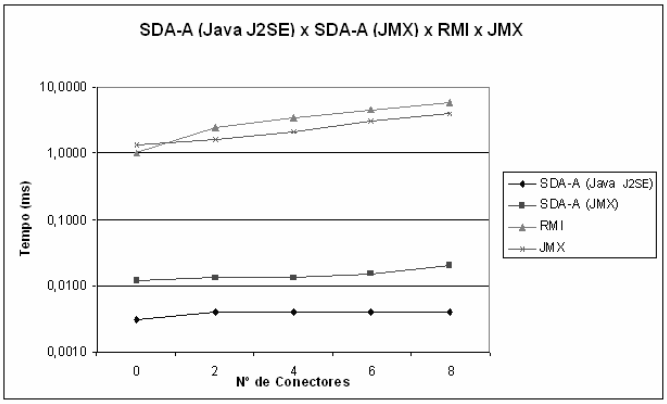

Figura 6 – Tempos de Execução Local

Existe ainda uma diferença de desempenho entre as formas de gerenciamento do *SDA-A*. O gerenciamento utilizando o *SDA-A* (JMX) tem um desempenho inferior ao gerenciamento utilizando o Java (*J2SE)* devido ao *overhead* acrescentado pela especificação JMX. O *overhead* considerado é referente ao redirecionamento de

chamadas que passam pelo MBeanServer [28], o que torna possível o gerenciamento dos objetos tipo *MBean* de forma dinâmica [28]. Finalmente, deve ser notado que para as duas formas de gerenciamento disponíveis no *SDA-A,* o custo de inserção de conectores é relativamente baixo, em relação às versões originais, o que as torna atrativas para a implementação de adaptações locais em aplicações utilizando Java ou JMX.

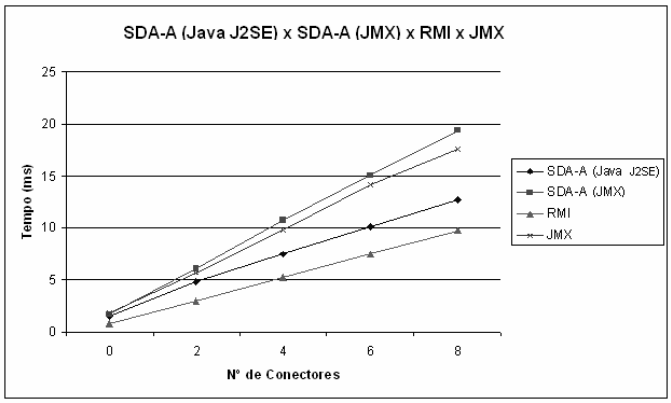

Figura 7 – Tempo de Execução Remoto

A Figura 7 apresenta os resultados dos testes remotos, onde o *SDA-A*, utilizando o Java (*J2SE)* para gerenciar as aplicações, obteve um resultado satisfatório, ficando entre o RMI e o JMX. O desempenho do RMI em relação ao *SDA-A* para nenhum *conector* interposto entre o cliente e o servidor foi 48% melhor, já para 8 conectores foi apenas 26,25% superior. Ou seja, observando a Tabela 1 é possível notar que na medida em que o número de conectores cresce, os benefícios de desempenho da utilização do RMI diminuem. A diferença de desempenho entre o RMI e o *SDA-A (Java J2SE)* é causada pelo *overhead* inserido pelo suporte para tornar possível o gerenciamento dinâmico dos *componentes*; contudo deve ser notado que o desempenho do *SDA-A (Java J2SE)* é bem superior ao do JMX. Já o desempenho do *SDA-A (JMX)*, ficou bem próximo do JMX, tornando viável a sua utilização para adaptações em contextos JMX. A pequena diferença entre o *SDA-A (JMX)* e o JMX também é causada pelo *overhead* inserido pelo suporte. A Tabela 1 apresenta uma comparação de desempenho em termos percentuais do *SDA-A (Java J2SE)* com relação ao JMX e ao RMI. A primeira linha mostra uma comparação de ganho na utilização do *SDA-A (Java J2SE)* em relação ou JMX. A segunda linha apresenta o ganho na utilização do RMI em relação ao *SDA-A (Java J2SE).*

Tabela 1 - Comparação de desempenho

Nº de Conectores	**0**	**2**	**4**	**6**	**8**
SDA-A(Java J2SE) melhor que JMX	*21,97%*	*15,12%*	*23,97%*	*28,47%*	*28,01%*
RMI melhor que *SDA-A(Java J2SE)*	*48,05%*	*37,61%*	*30,98%*	*26,15%*	*23,25%*

Além dos testes para avaliar o *overhead* inserido pelo suporte nas comunicações entre os *componentes* da aplicação, foi mensurado o custo para adicionar ou trocar um *componente* dinamicamente (com a aplicação em execução) utilizando o *SDA-A (Java J2SE)*. Através destas medias podemos saber qual será o custo para cada adaptação realizada na aplicação. O tempo gasto na troca ou adição de um *conector* varia em função de ele já ter sido instanciado anteriormente ou não (se já foi instanciado um objeto a partir da meta-classe criada pelo *SDA-A,* como visto na seção 2.3, por esta estar sendo utilizada por outros módulos da configuração). A Tabela 2 apresenta os valores de troca e adição de um *conector* em uma arquitetura carregada localmente (em um único *host*). Já a Tabela 3 apresenta os valores para uma arquitetura distribuída. Os valores observados são similares aos obtidos em outros projetos com objetivos semelhantes, e.g. [1]. Levando-se em consideração que, por mais dinâmico que um ambiente ubíquo seja, as aplicações em execução não permanecem em constante adaptação, os valores observados não devem causar um grande impacto na execução da aplicação.

Tabela 2 - Custos Locais Observados

Custos Locais	**Trocar um Conector**	**Colocar um Conector**

Criando uma Instância	23,309 (ms)	14,206 (ms)
Sem Criar uma Instância	12,275 (ms)	10,328 (ms)

Tabela 3 - Custos Remotos Observados

Custos Remotos	**Trocar um Conector**	**Colocar um Conector**
Criando uma Instância	28,665 (ms)	19,552 (ms)
Sem Criar uma Instância	21,455 (ms)	17,788 (ms)

5. Trabalhos Relacionados

A seguir apresentamos alguns trabalhos relacionados ao *SDA-A*.

O *Chisel* [16] é um *framework* que permite mudar em tempo de execução requisitos não funcionais da aplicação. A aplicação é construída separando os aspectos que não oferecem as funcionalidades, em múltiplos comportamentos não funcionais, possibilitando assim, mudar o seu comportamento não funcional apenas trocando-o por outro sem mudar a aplicação em si. O *framework* possui um gerente de adaptação interno o qual não pode ser utilizado separadamente para gerenciar a aplicações externas. Nele as adaptações são definidas em *scripts* denominados "Politicas de Adaptação" e as mesmas são iniciadas mediante eventos que são monitorados pelo *framework*. No *SDA-A* o *Coordenador* disponibiliza uma interface de gerenciamento para que as aplicações possam ser gerenciadas dinamicamente por aplicações externas. Contudo, o momento para adaptar uma aplicação fica a critério do desenvolvedor.

K-Component [11] é um modelo de componentes para construir aplicações adaptativas ao contexto. A aplicação é considerada como um grafo conectado, onde os vértices são interfaces (serviços oferecidos por instâncias de componentes) e as arestas têm papel similar aos dos *conectores*. No modelo, uma adaptação consiste em mudar as implementações de componentes e propriedades de *conectores* para estender ou aumentar os serviços oferecidos. Como no *Chisel*, ele utiliza eventos para notificar o gerente de adaptação os momentos que as adaptações devem ser realizadas e utiliza políticas de adaptação para definir como elas serão realizadas em uma aplicação. Em linhas gerais, o *K-Component* e o *Chisel* oferecem a capacidade de adaptação dinâmica similar ao *SDA-A.* Como a idéia do *K-Component* é oferecer um modelo para aplicações adaptáveis ao contexto, ele poderia ser utilizado para o desenvolvimento de aplicações em ambientes ubíquos (sem grande esforço), devido à sua capacidade de adaptação a diferentes contextos.

Dynamic Suport for Distributed Auto-Adaptive Applications [20] é um *framework* baseado na linguagem Lua [15] para o desenvolvimento de aplicações distribuídas, que podem adaptar automaticamente propriedades não funcionais de seus componentes e seu ambiente de execução. Ele possui um mecanismo de monitoramento conhecido como *LuaMonitor* e um mecanismo de adaptação denominado *Smart Proxy*. O *Smart Proxy* é responsável por realizar a adaptação do comportamento da aplicação. Ele encapsula as estratégias para seleção dinâmica de componentes, monitoração e adaptação. Ele também pode ativar diferentes componentes em tempo de execução quando necessário. Contudo, o uso do *Smart Proxy* torna a tarefa de adaptação, uma tarefa de baixo nível, pois, o desenvolvedor deve definir o código (em Lua) referente às estratégias de adaptação dentro do *Smart Proxy* enquanto no *SDA-A* é possível realizar as adaptações em um alto nível, través da interface disponibilizada pelo *Coordenador*.

O *Transparent Dynamic Reconfiguration for* CORBA [1] é um suporte que permite re-configuração dinâmica de sistemas distribuídos baseados em CORBA. Através deste suporte a re-configuração é realizada de forma transparente, para os *componentes* de uma aplicação. Essa proposta inclui um módulo responsável pelo gerenciamento de adaptação da aplicação, um módulo que contêm as informações de todos os objetos pertencentes a aplicação, e um outro módulo chamado *Reconfiguration Agent*, que é encarregado de determinar o momento apropriado para realizar a re-configuração do sistema. Apesar de o suporte ser semelhante ao *SDA-A* em relação aos elementos que compõem sua arquitetura, ele não emprega os conceitos de separação de interesses e Arquitetura de Software, implicando assim em um nível mais baixo de abstração. No *SDA-A* a determinação do momento apropriado para realizar adaptação fica a cargo do desenvolvedor ou de um mecanismo externo, como visto na seção 3, o qual determina quando é necessária uma adaptação.

6. Conclusões

Neste trabalho apresentamos o *SDA-A*, um suporte à adaptação dinâmica, baseado em tecnologias atuais amplamente disseminadas em âmbito comercial. Ele atende as necessidades de aplicações ubíquas, provendo

primitivas de configuração que podem ser usadas independentemente para colocar em execução e adaptar configurações de suas arquiteturas. Atualmente, estamos implementando, com base nas primitivas da linguagem de contrato, mecanismos de alto-nível, com poder de expressar de maneira sintética requisitos recorrentes em aplicações ubíquas, o que visa facilitar o desenvolvimento dessa classe de aplicação utilizando o *SDA-A*.

Como opções para o projetista, o *SDA-A* oferece um gerenciamento dirigido para aplicações usando o Java *J2SE*, cujo principal benefício é o desempenho, e um gerenciamento dirigido para aplicações baseadas na especificação JMX, cujo principal benefício é a transformação de componentes (ou aplicações) Java *J2SE* em componentes seguindo a especificação JMX, uma tecnologia bastante disseminada em âmbito comercial. Foram realizados experimentos os quais demonstraram que a proposta (incluindo suas duas opções) atende os requisitos de flexibilidade e desempenho, essenciais para a implementação de sistemas reais.

Além das duas formas de gerenciamento apresentadas aqui, o *SDA-A* oferece ainda uma terceira forma, não abordada neste artigo, que é baseada no Servidor de Aplicação JBoss [13, 27]. Através deste é possível gerenciar uma aplicação em execução no JBoss e até mesmo gerenciar os módulos do próprio Servidor de Aplicação dinamicamente. Maiores detalhes sobre o gerenciamento utilizando o servidor JBoss são discutidos em [26].

Para finalizar, mencionamos três pontos específicos para trabalhos futuros, que são o tratamento de questões de coordenação e atomicidade associadas às transações de adaptação e transferência dos estados dos componentes durante a adaptação. O primeiro pode ser relevante em algumas aplicações ubíquas que interagem com um ambiente que se encontra em constante mudança. O segundo é importante para garantir a re-configuração consistente das aplicações em presença de falhas, incluindo as de comunicação. E o terceiro para que os estado de um componente não seja perdido durante uma re-configuração

Referências

[1] Almeida, J. P. A.; Wegdam, M.; van Sinderen, M.; Nieuwenhuis, L.; Transparent dynamic reconfiguration for CORBA; IEEE Computer Society Press in Proceeding of the 3rd International Symposium on Distributed Objects & Application (DOA 2001), 17-20 September, 2001, Rome, Italy.

[2] Batista, T.; Rodriguez, N.; Dynamic reconfiguration of component-based applications; Proceedings of PDSE-2000a, 32-39, Limerick, Ireland, 10-11 June 2000. IEEE, IEEE Computer Society.

[3] Batista, T.; Chavez, C. V. F.; Rodriguez, N.; Conector Genérico: Um Mecanismo para Reconfiguração de Aplicações Baseadas em Componentes; In Third Ibero-American Workshop on Software Envoriments and Requeriment Engineering (IDEAS'00), Cancun – Mexico, April 2000b.

[4] Batista, T. V.; Rodriguez, N.; LuaSpace: Um Ambiente para Reconfiguração Dinâminca de Aplicações Baseadas em Componentes; In Workshop de Teses e Dissertações no Simpósio Brasileiro de Computação (CTD - SBC2001) , pp 1-8, Fortaleza, CE, Agosto 2001.

[5] Bidan, C.; Issarny, V.; Saridakis, A.; A dynamic reconfiguration service for CORBA; in Proc. IEEE International Conference on Configurable Distributed Systems, May, 1998.

[6] Capra, L.; Zachariadis, S.; Mascolo, C.; "Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems". In Proc. of International Conference on Pervasive Services (ICPS'05). July 2005. Santorini, Greece. To appear.

[7] Cardoso, L. X. T. Integração de Plataformas de Monitoração no Contexto de Arquiteturas Adaptáveis de Software. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), 2005.

[8] Carvalho, S.; Lisboa, J.; Loques, O.; Um Design Pattern para Configuração de Arquiteturas de Software, The Second Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP 2002 Conference, Itaipava, agosto, 2002.

[9] Chiba, S.; Nishizawa, M.; An Easy-to-Use Toolkit for Efficient Java Bytecode Translators; Proc. of 2nd Int'l Conf. on Generative Programming and Component Engineering (GPCE '03), LNCS 2830, pp.364-376, Springer-Verlag, 2003.

[10] Corradi, A. M.; Um Framework de Suporte a Requisitos Não-Funcionais para Serviços de Nível-Alto. Dissertação de Mestrado, Instituto de Computação (IC), UFF, Agosto, 2005.

[11] Dowling, J.; Cahill, V.; "The K-Component architecture meta-model for self-adaptive software", Proc. Reflection 2001, LNCS 2192.

[12] Ensink, B.; Adve, V.; Coordinating Adaptations in Distributed Systems. In Proceedings of the 24th international Conference on Distributed Computing Systems (Icdcs'04). ICDCS. IEEE Computer Society, Washington, DC, 446-455, March 24 - 26, 2004.

[13] Fleury, M.; Reverbel, F. The JBoss Extensible Server. In: International Middleware Conference, 2003, Rio de Janeiro. Resumos. Rio de Janeiro, 2003. p. 344-373.

[14] Freitas, G.; Cardoso, L.; Santos, A. L.; Loques, O.; Otimizando a Utilização de Servidores através de Contratos Arquiteturais, VII Workshop de Tempo Real - XXIII Simpósio Brasileiro de Redes de Computadores, Fortaleza, maio 2005.

[15] Ierusalimschy, R.; Figueiredo, L.; Celes, W.; Lua - an extensible extension language; Software: Pratice and Experience, 26(6):635–652, 1996.

[16] Keeney, J.; Cahill, V.; “Chisel: a policy-driven, context-aware, dynamic adaptation framework”; Dept. of Comput. Sci., Trinity Coll., Dublin, Ireland; Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop.

[17] Kniesel, G.; Costanza, P.; Austermann, M.; JMangler – A framework for load-time transformation of Java class files. IEEE Workshop on Source Code Analysis and Manipulation (SCAM). IEEE Computer Society Press: Los Alamitos, CA, 2001.

[18] Lisboa, J.; Loques, O.; Utilização do Design Pattern Architecture Configurator em um ambiente de suporte para Configuração de Arquitetura, The Third Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP 2003 Conference, Porto de Galinhas, agosto 2003.

[19] Loques, O.; Curty, R.; Ansaloni, S.; Sztajnberg, A.; A Contract-Based Approach to Describe and Deploy Non-Functional Adaptations in Software Architectures, Journal of the Brazilian Computer Society, V.10, N.1, julho de 2004.

[20] Moura, A. L.; Ururahy, C.; Cerqueira, R.; Rodriguez, N.; Dynamic Suport for Distributed Auto-Adaptive Applications; Proceedings of AOPDCS'02: Workshop on Aspect Oriented Programming for Distributed Computing Systems (held in conjunction with IEEE ICDCS 2002); Vienna, Austria, 2002, pp. 451-456.

[21] Nicoara, A.; Alonso, G.; Dynamic AOP with PROSE; International Workshop on Adaptive and Self-Managing Enterprise Applications (ASMEA'05) in conjunction with CAISE'05, Porto, 14 June 2005.

[22] Papadopoulos, G. A.; Arbab, F.; Configuration and dynamic reconfiguration of components using the coordination paradigm; Future Generation Computer Systems, 17(8):1023--1038, June 2001.

[23] Ranganathan, A.; Chetan, S.; Al-Muhtadi, J.; Campbell, R. H.; Mickunas, M. D.; Olympus: A High-Level Programming Model for Pervasive Computing Environments; In IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), Kauai Island, Hawaii, March 8-12, 2005.

[24] Redmond, B.; Cahill, V.; Iguana/J: Towards a dynamic and efficient Reflective Architecture for Java; in Workshop on Reflection and Meta-Level Architectures at 14th European Conference on Object Oriented Programming (ECOOP), Cannes, France 2000.

[25] Román, M.; Hess, C. K.; Cerqueira, R.; Ranganathan, A.; Campbell, R. H.; Nahrstedt, K.; Gaia: A Middleware Infrastructure to Enable Active Spaces; In IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[26] Santos, A. L. G. Técnicas de Configuração Dinâmica de Arquiteturas de Software. Dissertação de mestrado em andamento, Instituto de Computação – Universidade Federal Fluminense (IC/UFF), 2006.

[27] Stark, S.; The Jboss Group. JBoss Administration and Development Third Edition (3.2.x Series). Atlanta, 2003.

[28] Sun MicroSystems; Java Management Extensions; Disponivel na Internet; Acessado em 02/06/2005 http://java.sun.com/products/JavaManagement

[29] Sztajnberg, A. Flexibilidade e Separação de Interesses para a Concepção e Evolução de Aplicações Distribuídas. Tese de Doutorado, COPPE/PEE/UFRJ, Maio, 2002.