Desenvolvendo Software Adaptável para Computação Móvel

Francisco José da Silva e Silva
Departamento de Informática
Universidade Federal do Maranhão
fssilva@ime.usp.br

Markus Endler
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
endler@inf.puc-rio.br

Fabio Kon
Departamento de Ciência da Computação
Universidade de São Paulo
kon@ime.usp.br

Resumo— Em um futuro não distante usuários de sistemas de computação móvel terão acesso a milhares de serviços e recursos computacionais, muitos destes sensíveis à sua localização no momento do acesso. Este ambiente de computação será altamente dinâmico. Novos serviços e recursos estarão freqüentemente sendo incluídos, removidos, reconfigurados ou movidos de lugar. Os usuários, utilizando diversos dispositivos, movimentar-se-ão entre diversos ambientes e a comunicação e coordenação de todos estes componentes dar-se-á através de uma rede composta por diversas tecnologias heterogêneas. Neste ambiente de constantes mudanças, aspectos como a movimentação de usuários, variações na disponibilidade de recursos e serviços e a heterogeneidade de dispositivos deverão ser levados em consideração pelos desenvolvedores de software de forma a acompanhar este dinamismo. Este trabalho apresenta um modelo que está servindo de base para a implementação de um *framework* que tem por objetivo disponilizar um conjunto de ferramentas através das quais poder-se-á instrumentar uma aplicação distribuída de forma a torná-la adaptável em tempo de execução e, assim, suportar variações em seu ambiente de execução.

Abstract— In a near future, users of mobile computing systems will have access to a great number of services and resources. This computing environment will be highly dynamic. New services and resources will be frequently included, removed, reconfigured, and moved from one place to another. Users will be moving and the communication and coordination between all these components utilize a highly heterogenous network. In these ever changing environments, software developers must take into consideration aspects such user mobility, variations in resources availability and device heterogeneity. The software will have to be as dynamic as the computational environment it is inserted. This can only be achieved through adaptation. This work presents an adaptation model that is serving as a base for the development of a framework that will help to build distributed applications that supports dynamic reconfiguration of its components as changes in the execution environment take place.

aluno de doutorado em Ciência da Computação do Instituto de Matemática e Estatísca, Departamento de Ciência da Computação da Universidade de São Paulo

I. Introdução

O desenvolvimento de diversas tecnologias de computação e comunicação tem transformado o cotidiano de um número cada vez maior de pessoas, causando profundas mudanças na forma, na quantidade, na qualidade e na velocidade com a qual trocamos informação. O contínuo avanço tecnólogico e os esforços sendo realizados em diversas pesquisas em curso nos permite antever que os sistemas de computação em um futuro não distante serão compostos por um grande número de dispositivos heterogêneos, tanto fixos quanto móveis, conectados através de diferentes tecnologias de rede. Um ponto importante desta infra-estrutura será a mobilidade. O usuário não estará mais restrito a pontos de acesso fixos e, através da utilização de diferentes dispositivos, será capaz de acessar milhares de recursos computacionais e serviços, muitos destes sensíveis a sua localização no momento do acesso. Um outro aspecto a ser levado em consideração nestes novos ambientes são as recentes pesquisas sobre espaços ativos, como o projeto Gaia [15]. Espaços físicos como escritórios, salas de aula, residências e hospitais serão cada vez mais equipados com dispositivos de computação, formando um ambiente que integra diversos equipamentos como computadores, projetores de imagem, câmeras de vídeo, quadros brancos eletrônicos, etc. O objetivo é proporcionar informação aos usuários deste espaço, ajudando-os a realizar de maneira mais fácil suas atividades ou até mesmo permitindo a realização de novas atividades através da interação de maneira uniforme com o ambiente de computação constante destes espaços físicos.

Estes novos ambientes de computação altamente dinâmicos impõem sobre o software uma série de requisitos, entre os quais destaca-se a necessidade de mecanismos que permitam ao software adaptar-se a mudanças no ambiente decorrentes da mobilidade e de variações na disponibilidade de recursos e serviços [5]. Esta necessidade de adaptação torna-se clara ao analisarmos algumas características destes ambientes:

- Alta variação na largura de banda disponível e da latência da comunicação decorrente dos seguintes fatores: o uso conjunto de diversas tecnologias de rede, que apresentam taxas de transmissão variando de alguns Kbytes até vários Mbytes, como é o caso de tecnologias de rede sem fio [14]; mobilidade dos usuários ocorrendo rápidas mudanças na qualidade do sinal para a comunicação com a estação base. O software do sistema e as aplicações devem se adaptar a esta variação, provendo ao usuário um nível variável de detalhes e qualidade das informações sendo apresentadas, porém conservando sua semântica. Pode-se, ainda, variar os serviços disponíveis ao usuário. Por exemplo, um servidor Web pode passar a enviar figuras em preto e branco e com menor resolução caso a largura de banda disponível para a comunicação com um cliente torne-se pequena. Uma ferramenta de correio eletrônico pode desabilitar o envio do conteúdo das mensagens constantes da caixa de entrada de um usuário, restringindo-se ao seu cabeçalho pelo mesmo motivo do exemplo anterior.
- Heterogeneidade da rede: usuários de dispositivos móveis podem vir a experimentar diferentes qualidades de serviço (QoS) de acordo com o ponto de acesso provida pela rede. Os dispositivos devem também ser capazes de chavear dinamicamente entre diferentes interfaces de rede à disposição bem como o protocolo de controle de acesso ao meio (MAC), por exemplo, ao mudar de uma rede celular disponível dentro da cidade para uma rede que utiliza satélite e abrange todo o país [6].
- Heterogeneidade de dispositivos tanto com relação a propriedades de hardware quanto de software. Exemplos destas propriedades são: tamanho da área de visualização, cores e resolução disponíveis, poder de processamento, quantidade de memória real e secundária disponível, sistema operacional, modelo de componentes suportado, etc. Esta heterogeneidade também impõe ao software a necessidade de ter mecanismos para adaptação. Por exemplo, um servidor Web pode automaticamente converter um documento *Postscript* para o formato texto caso o usuário solicitante esteja utilizando um Palm III.
- Variação nos tipos e quantidade de serviços e recursos disponíveis na rede a medida que o usuário se desloca de um local a outro e a medida que novos dispositivos se aproximam do espaço geográfico ocupado pelo dispositivo do usuário.

O projeto de pesquisa aqui apresentado aborda o problema de como adaptar dinamicamente software a variações em seu meio de execução. Em particular, estamos interessados em sistemas compostos de diversos componentes distribuídos em uma rede, inseridos em um ambiente de computação móvel. A abordagem seguida foi o desenvolvimento de um modelo que está servindo de base para a implementação de um *framework* cujo objetivo é simplificar o desenvolvimento de aplicações distribuídas adaptáveis dinamicamente. Este *framework* deve prover ferramentas genéricas o suficiente de forma a permitir seu uso em diversos contextos.

Este artigo está organizado da seguinte forma: a seção II apresenta o projeto SIDAM, Sistema de Informação Distribuído para Agentes Móveis. O SIDAM é um projeto de pesquisa em curso no Instituto de Matemática e Estatística da Universidade de São Paulo e a sua aplicação de referência, um sistema de disseminação de informações de trânsito em um ambiente de computação móvel e distribuído, será utilizada para validação do *framework* a ser desenvolvido. a seção III apresenta o modelo sendo proposto como base para o desenvolvimento do *framework*. Em seguida, a seção IV descreve o projeto do avaliador de eventos cuja implementação já está em curso. A seção V descreve diversos trabalhos relacionados com a área de adaptação dinâmica e os relaciona com a pesquisa apresentada neste documento. Finalmente, a seção VI apresenta as conclusões que podem ser obtidas a partir do trabalho já realizado.

II. Aplicação de Referência: SIDAM

O objetivo do projeto SIDAM [4] é o estudo dos problemas relacionados ao projeto e implementação de sistemas de informação descentralizados utilizados tanto por usuários móveis quanto fixos. Objetiva-se o desenvolvimento de novos algoritmos, teorias e métodos relativos ao tema. Como aplicação de referência, o SIDAM utiliza um sistema *online* para disseminação de informações de trânsito em uma área metropolitana. No entanto, o modelo desenvolvido é genérico o suficiente para ser utilizado em um grande espectro de aplicações.

O projeto SIDAM adota o *modelo indireto*[1] para as interações cliente-servidor. Neste modelo, ilustrado na figura 1, todas as comunicações de e para um dispositivo móvel (*Mh - Mobile Host*) são intermediadas por estações base que atuam como representantes de todos os dispositivos móveis dentro de sua célula. O modelo do sistema consiste de dois tipos de máquinas: os computadores fixos (*Fh - Fixed Host*) e os dispositivos móveis (*Mh*), onde os *Fhs* estão conectados entre si através de uma rede fixa e confiável. Alguns *Fhs* atuam como estações base (*Mss - Mobile Support Station*). Cada *Mss* define uma região geográfica, denominada célula, na qual é capaz de se comunicar com o conjunto de *Mhs* que se encontram dentro da mesma. A informação sobre quais *Mhs* estão presentes a cada momento dentro da célula é mantida pela respectiva *Mss*. Outros *Fhs* atuam como servidores de dados (*Ds - Data Server*) e podem disponibilizar serviços tanto para máquinas fixas quanto móveis. Dentro da aplicação de referência SIDAM, os *Ds* armazenam informações de trânsito.

Para manter o tamanho da base de dados e o volume de consultas correspondentes gerenciáveis, a aplicação SIDAM divide a cidade em micro-regiões. Cada micro-região corresponde a algumas quadras e são unidades atômicas de armazenamento de informações de trânsito. Os dados das micro-regiões são armazenados em servidores de informação (*Is* -

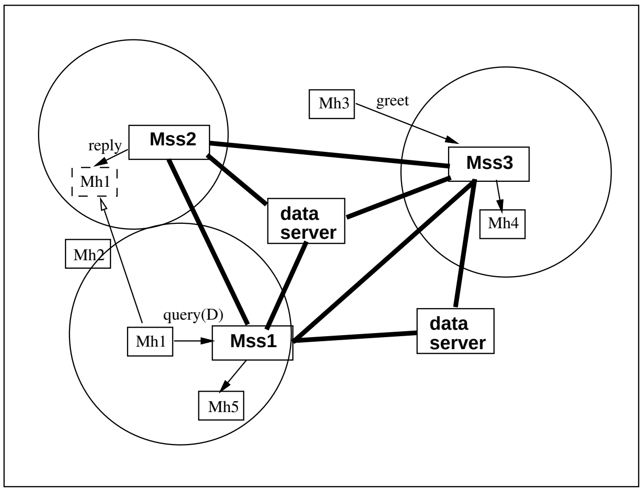

Fig. 1. Modelo SIDAM com duas estações base e cinco dispositivos móveis

Information Server). Cada *Is* pode conter dados de uma ou mais micro-regiões. Servidores de informação residem em servidores de dados (*Ds*), que são computadores fixos.

Um dos requisitos do projeto SIDAM é que as aplicações sejam capazes de se adaptar às variações de carga, padrões de requisição e a diversos cenários de disponibilidade de recursos. Nós identificamos algumas adaptações úteis dentro do contexto da aplicação de referência SIDAM:

1) **separação e agregação de *Is*** : servidores de informação podem ser entidades altamente dinâmicas. Caso um *Is* fique sobrecarregado com requisições, deve ser possível distribuir a carga de requisições através da divisão do conjunto de micro-regiões por ele armazenadas, transferindo um destes conjuntos para outro servidor de informação. Por outro lado, caso dois ou mais *Is* tornem-se sub-utilizados, o sistema pode uni-los em um único *Is*, a fim de diminuir os custos de sincronização dos estados das réplicas;
2) **replicação de *Is*** : pode ocorrer de muitos usuários móveis, localizados em uma determinada região, emitirem uma grande quantidade de requisições destinadas a um *Is* "distante" na rede fixa da *Mss* responsável pela região na qual estes usuários estão localizados. Estas requisições terão de ser roteadas através de diversas máquinas até que atinjam o *Is* destino, o que degradará a latência das consultas. Nesta situação, o sistema deve replicar o *Is* sendo requisitado em um servidor de dados "próximo" na rede fixa da *Mss* responsável pela célula na qual os usuários se encontram;
3) **variação no formato do resultado de consultas**: em situações normais, o resultado de uma operação de consulta tem o formato de uma imagem da área solicitada. As ruas são destacadas com diferentes cores, de acordo com a condição do tráfego nas mesmas. O usuário pode ainda, no seu dispositivo, realizar operações de ampliação ou redução da imagem recebida. No entanto, em ambientes de computação móvel existe uma grande variação na largura de banda disponível para a comunicação com o dispositivo móvel. Caso esta torne-se escassa, o resultado de operação de consulta deve limitar-se ao formato textual;
4) **Migração de *Is*** : A quantidade de requisições atendidas pelos *Is* residentes em um dado *Ds* pode variar bruscamente no tempo. Assim sendo, é possível que um *Ds* torne-se sobrecarregado acarretando uma degradação no desempenho do sistema. Para contornar esta situação, uma vez detectado que um *Ds* encontra-se sobrecarregado, deve-se migrar parte de seus *Is* para outros *Ds* cuja carga de trabalho corrente seja suficiente para acomodá-los;
5) **Realocação e pré-alocação de recursos**: Dispositivos móveis de usuários especiais, como aqueles usados em ambulâncias ou carros do corpo de bombeiros, devem ter os recursos necessários para sua comunicação com o sistema previamente alocados na célula destino, de forma a garantir que sempre terão acesso às informações sendo disponibilizadas. Isto pode requerer a realocação de recursos já alocados a dispositivos com menor prioridade.

III. MODELO PROPOSTO

A figura 2 representa uma visão geral do modelo proposto através de um diagrama de pacotes UML [2].

O modelo é composto por três pacotes principais. O primeiro, *monitoração*, é o responsável pelo monitoramento das entidades que compreendem o sistema. Os dados coletados pelos componentes deste pacote são armazenados em *logs* ou enviados diretamente ao pacote de *detecção de eventos*. Este último é responsável por analisar os dados coletados e identificar a ocorrência de eventos que demandem reconfigurações no sistema. No caso de sua ocorrência, os eventos são enviados ao pacote de reconfiguração dinâmica que é responsável por verificar as pré-condições de adaptação dinâmica e aplicar as ações de reconfiguração necessárias para que o sistema se adapte às novas condições do ambiente de execução. A seguir detalharemos estes três pacotes principais.

A. Monitoração

O pacote de monitoração é composto por quatro subpacotes. O primeiro destes é responsável pela interceptação das interações entre os objetos que compreendem as aplicações do sistema distribuído. Nossa intenção é seguir o modelo descrito em [10]. Neste trabalho, são utilizados *interceptadores*, descritos no padrão CORBA [13], que são inseridos entre o objeto cliente e o servidor. Sempre que o cliente invocar um método em um objeto, a mensagem correspondente a esta invocação é interceptada e posteriormente

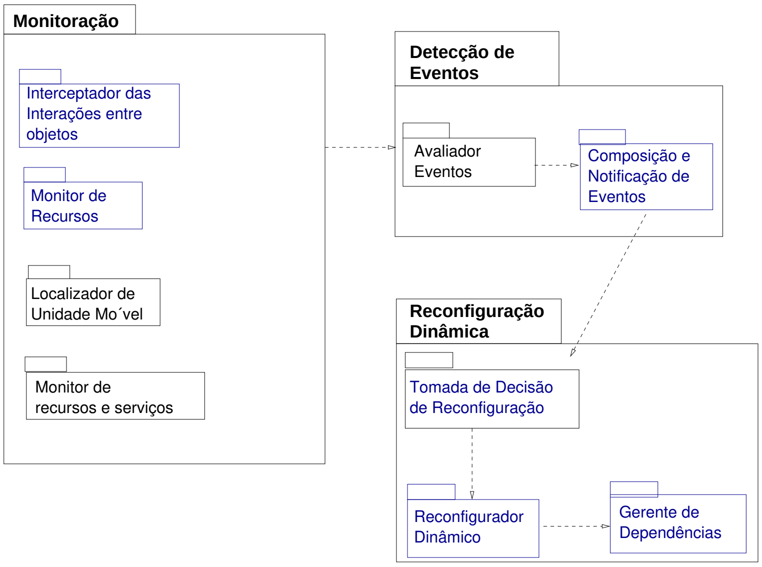

Fig. 2. Modelo para adaptação dinâmica: visão geral

reencaminhada para o objeto alvo. Através do uso de interceptadores, pode-se extrair informações úteis de cada chamada de método, armazenando-as em um *log* para que possam posteriormente ser analisadas pelo pacote de detecção de eventos. Em [10], o *log* armazena um conjunto fixo de informações que, entre outras, compreendem o nome do objeto para o qual a requisição é destinada, nome do método sendo invocado e tempo de duração da execução do método no servidor. No entanto, muitas vezes é interessante que o *log* armazene também informações específicas do sistema distribuído sendo monitorado. Por exemplo, no caso da aplicação de referência do SIDAM, poder-se-ia armazenar a micro-região para a qual a requisição se destina, bem como a informação em qual célula o cliente móvel requisitante se encontra quando da sua emissão. A análise do conjunto de requisições constantes do *log* permitiria detectar a necessidade de se replicar um dado *Is* pelo fato do mesmo se encontrar "distante" das *Mss's* através das quais dispositivos estivessem emitindo muitas requisições. Assim, para que o modelo atinja o objetivo de ser genérico o suficiente para que seja utilizado em diferentes contextos e aplicações, faz-se necessário que o próprio sistema de monitoração das interações entre os objetos da aplicação seja configurável em tempo de execução, através de componentes específicos para cada aplicação que armazenem em *logs* distintos dados específicos da interação entre os objetos da mesma.

O serviço de monitoração deve também ser capaz de monitorar a situação dos diversos recursos distribuídos existentes nos nós da rede. Exemplos destes recursos são: memória real e virtual, UCP, memória secundária e conexões de rede. Este papel é desempenhado pelo pacote *monitor de recursos*. A monitoração de recursos é baseada no conceito que denominamos *parâmetro de monitoração*, definido em [16]. O parâmetro de monitoração representa o aspecto do recurso distribuído a ser monitorado. Por exemplo, pode-se querer monitorar o percentual de uso dos processadores, a quantidade de memória real e virtual disponível e a largura de banda existente para a comunicação com os dispositivos. Para cada parâmetro de monitoração deve-se definir um conjunto de faixas (valores) de operação. O uso de diversas faixas de operação permite uma maior flexibilidade no processo de adaptação. Um servidor Web, por exemplo, poderia passar a enviar as figuras em preto e branco caso a largura de banda disponível para a comunicação com seu cliente se torne pequena e poderia passar a omiti-las caso a largura de banda caia ainda mais. O monitor de recursos notifica o avaliador de eventos sempre que um parâmetro de monitoração mudar a faixa na qual vinha operando. Para implementar esta facilidade de monitoração, propomos a utilização de *objetos de monitoração de recursos*. Para cada parâmetro, define-se um objeto de monitoração que é instanciado em toda máquina na qual existe um recurso a ser monitorado. O objeto de monitoração periodicamente verifica através de uma consulta local em qual faixa o parâmetro monitorado está operando. Sempre que o parâmetro trocar de faixa de operação, o objeto de monitoração notificará o avaliador de eventos. Através do *framework*, pretende-se disponibilizar um conjunto de objetos de monitoração para diversos recursos comumente monitorados, como os mencionados anteriormente. No entanto, o usuário poderá implementar seus próprios objetos de monitoração, devendo apenas seguir a definição das interfaces definidas no modelo.

Os dois últimos aspectos relativos à monitoração lidam com a localização de dispositivos móveis e com a descoberta dinâmica de recursos e serviços à medida que o usuário se desloca. Estes dois aspectos são de grande importância nos ambientes de computação distribuída atualmente em pesquisa e muitas aplicações podem ser desenvolvidas a partir destes. No entanto, o estágio atual de nosso trabalho concentra nossa atenção na monitoração das interações entre os objetos que compõem a aplicação e na condição de operação dos recursos disponíveis no ambiente.

B. *Detecção de Eventos*

A detecção de eventos foi sub-dividida em dois pacotes. O primeiro, *Avaliador de Eventos*, é responsável por analisar os dados coletados pela monitoração e, a partir destes, determinar a ocorrência ou não de eventos que sejam de interesse para uma possível adaptação da aplicação distribuída. A especificação deste pacote será apresentada na seção IV. Uma parte do avaliador lida com a detecção de eventos baseada na análise dos *logs* criados pelo interceptador de interações entre objetos. Estes eventos são denominados *eventos de log*. Uma outra parte do avaliador lida com a análise dos dados enviados pelo monitor de recursos. Os eventos gerados a partir desta análise são denominados *eventos de recursos*.

O segundo pacote, denominado *Composição e Notificação de Eventos* possui duas tarefas básicas. A primeira é permitir ao usuário a definição de expressões de eventos complexos a partir da combinação de outros eventos. A segunda tarefa é notificar a ocorrência de eventos, tanto simples quanto complexos. Serão notificados todos os objetos que tenham registrado interesse no evento ocorrido. Em especial, será notificado o pacote responsável pela reconfiguração dinâmica que determinará quais ações de reconfiguração devem ser executadas e controlará sua aplicação. Como ponto de partida para a implementação deste pacote, estamos atualmente investigando a utilização do EPS [11], um monitor para eventos compostos e distribuídos.

C. *Reconfiguração Dinâmica*

O pacote *Reconfiguração Dinâmica* foi organizado em três sub-pacotes. O primeiro, *Tomada de Decisão de Reconfiguração*, é responsável por receber a notificação da ocorrência de eventos e determinar quais ações de reconfiguração são necessárias para adaptar o sistema face a ocorrência dos mesmos. Um conjunto de ações de reconfiguração são enviadas ao *Reconfigurador Dinâmico*, que é responsável por aplicá-las contando, para isto, com a ajuda do *Gerente de Dependências*. Este último pacote controla as dependências existentes entre os componentes que compõem a aplicação distribuída. O gerenciamento destas dependências permite que o processo de reconfiguração se dê de forma consistente e segura.

Para implementar o gerente de dependências, planejamos utilizar os componentes denominados *Automatic Configuration Service* e *Component Configurator*, descritos em [8]. Através deles, dois tipos de dependências podem ser representadas:

- *Pré-Requisitos* necessários para carregar e ativar um componente inerte no sistema. Eles expressam as dependências existentes entre o componente a ser carregado e componentes persistentes de hardware e software. Pré-requisitos são especificados através de listas que contêm três tipos diferentes de informação: a natureza dos recursos de hardware exigidos pelo componente, a capacidade dos recursos de hardware necessária para a execução do mesmo e os serviços de software requeridos. O gerenciamento de pré-requisitos tem por objetivo garantir que todos os componentes de hardware e software necessários à execução do novo componente estejam presentes no momento de sua ativação;
- *Dependências dinâmicas* existentes entre os componentes ativos do sistema. O *component configurator* é responsável por armazenar as dependências existentes entre um componente específico e os demais componentes que compõem a aplicação sendo executados. O objetivo do gerenciamento de dependências dinâmicas é garantir a consistência em tempo de execução. Por exemplo, se um componente é descontinuado devido à ocorrência de alguma falha, os componentes que dele dependem podem ser notificados da ocorrência deste evento e, desta forma, podem realizar uma interrupção de execução suave.

IV. Projeto do Avaliador de Eventos

Para prover toda a funcionalidade e flexibilidade requeridos para suportar o ambiente de computação adaptável do futuro, faz-se necessário que o avaliador de eventos contemple os seguintes requisitos:

- **Extensibilidade**. Cada aplicação possui um conjunto de eventos que é útil somente no seu próprio contexto. Portanto, o avaliador de eventos deve permitir ao usuário estender o conjunto de eventos que estão sendo monitorados e detectados bem como modificar as especificações dos avaliadores responsáveis por sua detecção;
- **Reconfiguração Dinâmica**. As extensões e modificações a serem realizadas no código do avaliador de eventos não podem ser estaticamente compiladas em seu código. Caso contrário, necessitaríamos recompilá-lo sempre que o usuário desenvolvesse uma nova aplicação. Ao aplicar mudanças e extensões ao avaliador de eventos sua execução não deve nem mesmo ser interrompida, já que isto afetaria a detecção de eventos de outras aplicações em execução no ambiente. Portanto, o avaliador de eventos deve suportar reconfigurações dinâmicas;
- **Gerenciável**. O avaliador de eventos deve prover mecanismos através dos quais o usuário possa adicionar e remover definições para avaliação de eventos, carregá-las para memória bem como ativar e desativar sua avaliação em tempo de execução. O usuário deve dispor

de uma interface amigável para inspecionar e modificar a configuração interna do avaliador de eventos.

Um aspecto fundamental para atender aos requisitos acima descritos é a utilização de mecanismos que suportem mudanças dinâmicas no código do avaliador de eventos. Para tanto, pretendemos utilizar os componentes *Automatic Configuration Service* e *Component Configurator*, de forma semelhante ao realizado no *dynamicTAO* [9]. *dynamicTAO* é um ORB compatível com o padrão CORBA que utiliza reflexão computacional e suporta reconfigurações dinâmicas. Os aspectos internos reconfiguráveis dinamicamente são organizados em categorias. Cada categoria possui diversas implementações. Por exemplo, a categoria denominada `Concurrency` possui três modelos de *threads* suportadas pelo *dynamicTAO*: `Reactive_Strategy`, `Thread_Strategy` e `Thread_Pool_Strategy`. As implementações são disponibilizadas através de bibliotecas de carregamento dinâmico (*DLL - Dynamic Loadable Library*), de forma a permitir que sejam ligadas ao código do ORB em tempo de execução. Um repositório persistente armazena as implementações disponíveis no sistema de arquivos local e um componente denominado *Dynamic Service Configurator* disponibiliza as operações que permitem ao usuário armazenar ou apagar componentes do repositório, carregar ou removê-los da memória principal, bem como suspender ou ativar sua execução. Pretendemos utilizar um modelo semelhante no contexto da detecção de eventos.

A. *Colaboração entre Componentes*

O projeto do avaliador de eventos foi dividido em duas partes. A primeira lida com a detecção de eventos de *logs* enquanto que a segunda lida com a detecção de eventos de recursos.

1) Eventos de Logs*:* A figura 3 apresenta o diagrama de colaboração UML dos componentes responsáveis pela detecção de eventos de *logs*.

Para permitir ao usuário a flexibilidade de estender ou alterar o código do avaliador de eventos, cada evento possui um *Detector de Evento* próprio que encapsula o código responsável por sua detecção. Cada detector de evento é fornecido no formato de uma biblioteca de carregamento dinâmico. Tipicamente existirão diversos detectores de eventos ativos na memória, conforme ilustrado na figura 3, um para cada evento sendo avaliado. Tanto o código do detector de evento quanto a freqüência na qual ele deve ser executado são fornecidos pelo usuário e compõem o que denominamos de definição do detector de evento.

O *Configurador* provê uma interface através da qual o usuário pode gerenciar a execução do avaliador de eventos. Através desta interface, pode-se armazenar ou remover de um repositório permanente definições de eventos, carregar ou remover estas definições para a memória principal bem como ativar ou suspender sua execução. Este componente é utilizado tanto para eventos baseados em *logs* quanto para eventos

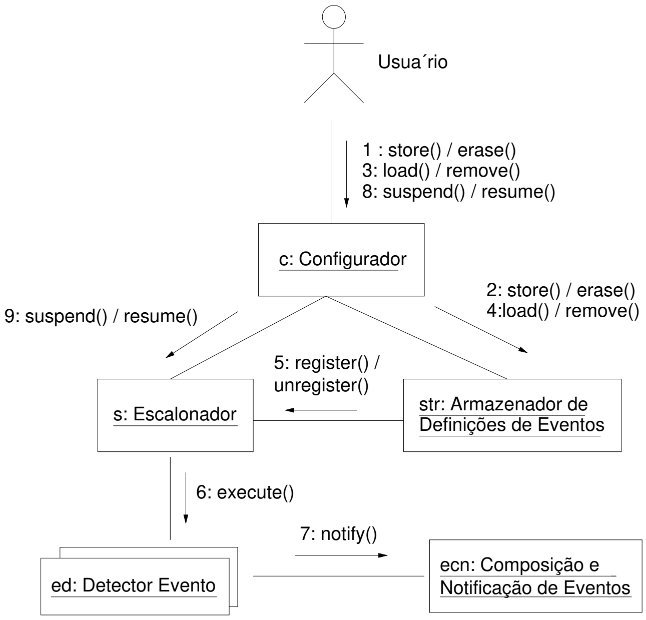

Fig. 3. Avaliação de eventos de *logs*

baseados na monitoração de recursos, fornecendo-se, assim, uma interface integrada ao usuário. Esta difere somente no método utilizado para definir eventos de tipos diferentes já que os parâmetros utilizados na definição de eventos de *logs* diferem dos utilizados na definição de eventos de recursos.

O *Armazenador de Definições de Eventos* é responsável por gerenciar o repositório permanente utilizado para armazenar as definições de evento fornecidas pelo usuário. O mesmo componente é utilizado para o armazenamento de definições de eventos baseados em *logs* (que correspondem aos detectores de eventos) bem como as definições de eventos baseados na monitoração de recursos.

Da definição do detector de evento consta a freqüência na qual ele deve ser executado pelo avaliador de eventos. Para controlar o momento correto da execução dos diversos detectores de evento ativos na memória, temos um *Escalonador*. A execução de cada detector de evento é realizada através de uma linha de controle (*thread*) individual.

Os eventos detectados são enviados ao pacote de *Composição e Notificação de Eventos*, conforme descrito na seção III-B. Este é responsável por notificar os objetos que tenham registrado interesse na ocorrência dos eventos detectados.

2) Eventos de Recursos: O diagrama de colaboração dos componentes responsáveis pela detecção de eventos de recursos está ilustrado na figura 4.

O *Objeto de Monitoração de Recurso* (*OMR*), descrito inicialmente na seção III-A, faz parte do pacote de monitoração. Sua tarefa é monitorar um parâmetro específico (como, por exemplo, o percentual de uso da UCP) em uma determinada entidade do sistema, que pode ser uma máquina, uma conexão de rede, um componente de software como o *Is* no

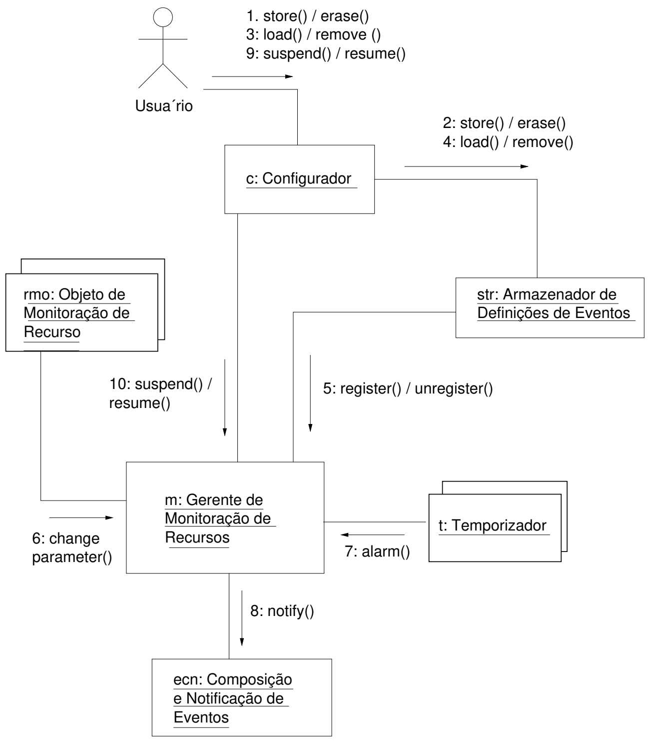

Fig. 4. Avaliação de eventos de recursos

SIDAM, etc. Tipicamente haverão diversos *OMR* espalhados pelo ambiente distribuído.

O *Configurador*, compartilhado tanto pela avaliação de eventos de *log* quanto pelos eventos de recursos, disponibiliza um método através do qual o usuário pode definir eventos. Eventos de recursos são definidos de forma diferente da descrita na seção IV-A.1. Para definir um evento de recurso, o usuário não necessita fornecer o código para a detecção. Ele informa apenas uma expressão booleana e um tempo de duração. A expressão booleana é composta por um ou mais parâmetros. Cada parâmetro da expressão possui uma faixa alvo. Por exemplo, a sobrecarga de um *Ds* poderia ser detectada caso o percentual de uso da UCP esteja acima de 80% e a quantidade de memória física disponível esteja abaixo de 5 MB. O tempo de duração determina o intervalo de tempo no qual a expressão deve permanecer verdadeira para que a notificação da ocorrência do evento seja disparado. A utilização do tempo de duração evita a geração de eventos no caso de uma situação temporária como um pico isolado na utilização da UCP.

O *Gerente de Monitoração* é o componente central deste modelo. Ele armazena as definições dos eventos presentes na memória principal e é encarregado da comunicação com os diversos objetos de monitoração de recursos instanciados nos nós do ambiente. Quando o gerente de monitoração de recursos recebe uma notificação de que um parâmetro sofreu alteração da faixa em que vinha operando, este verifica quais expressões booleanas definidas pelo usuário possuem este parâmetro. Estas expressões são avaliadas utilizando-se os valores correntes dos parâmetros relativos a entidade em questão. Se uma destas expressões for avaliada como verdadeira, o gerente de monitoração iniciará um *Temporizador* que lhe enviará uma mensagem quando o tempo de duração definido para esta expressão expirar. Ao receber esta mensagem, o gerente de monitoração notificará o pacote responsável pela composição e notificação de eventos da ocorrência do evento definido pela expressão, informando em que entidade do sistema este se deu. Se, durante o intervalo de tempo entre a avaliação da expressão como verdadeira e o recebimento da mensagem de que seu tempo expirou, o gerente de monitoração receber uma nova notificação de um objeto de monitoração de recursos que torne a expressão falsa, o temporizador é desligado e não haverá o disparo da notificação do evento.

V. TRABALHOS RELACIONADOS

Sudame e Badrinath [16] descrevem um mecanismo baseado em mensagens ICMP para propagação de informações a respeito da condição de parâmetros do ambiente de rede como, por exemplo, o custo financeiro da comunicação, latência e largura de banda. Este mecanismo pode ser estendido para comportar parâmetros específicos de ambientes de computação móvel como intensidade do sinal, localização e consumo de energia. A partir da notificação da condição dos parâmetros monitorados, ações de reconfiguração dos protocolos TCP e UDP podem ser automaticamente disparadas. O conjunto de ações disponíveis é pré-definido. Este trabalho não trata o problema de como reconfigurar dinamicamente os componentes da aplicação, mas está focado na adaptação da camada do protocolo de rede através do ajuste de parâmetros. Além disto, a monitoração, notificação e adaptação são definidos e executados localmente em máquina, não contemplando um ambiente distribuído.

Fox et al. [7] apresentam uma abordagem para o problema de como suportar variações na largura de banda e na arquitetura de software e hardware dos clientes. O enfoque deste trabalho está em aplicações de acesso a dados na Internet. Um *proxy* é inserido entre o cliente e o servidor. Ele fica localizado próximo ao servidor e é conectado a este através de uma conexão que possui grande largura de banda e baixa latência de comunicação. Sua função é recuperar conteúdo de servidores Internet a partir de solicitações realizadas pelos clientes, determinar os tipos de dados que compõem a resposta à solicitação (texto, figuras, etc. . .) e determinar quais algoritmos devem ser empregados de forma a adaptar este conteúdo à largura de banda disponível para a comunicação com o cliente bem como à arquitetura de hardware e software do mesmo. Estes algoritmos são dependentes do tipo de dado sendo transmitido e realizam tarefas como compressão, supressão de cores ou quadros (no caso de vídeo), diminuiçao da resolução de imagens, etc. O modelo que estamos propondo neste documento é mais geral e pode ser utilizado para a implementação de uma arquitetura similar a de *proxy ativo*.

Odyssey [12] é um conjunto de extensões ao sistema operacional NetBSD com o objetivo de prover suporte à adaptação de aplicações de acesso a informações requisitadas por dispositivos móveis. Apesar de possuir uma abordagem diferente da utilizada em *proxy ativo*, Odyssey utiliza o mesmo princípio de transformar o fluxo de dados sendo transmitido ao cliente de forma a suportar variações no ambiente. Na visão de Odyssey, o sistema operacional é responsável por monitorar a disponibilidade de recursos, notificar as aplicações de mudanças significativas na condição dos mesmos e tomar decisões de forma a otimizar a utilização dos mesmos. Cada aplicação, por sua vez, é responsável por decidir a melhor forma de explorar os recursos disponíveis. Considera-se que o dispositivo móvel pode estar executando diversas aplicações concorrentemente. Um aspecto que difere nosso trabalho do Odyssey é que neste último a aplicação adaptativa é dividida em dois componentes: o cliente e o servidor. Nosso interesse é tornar adaptável dinamicamente aplicações distribuídas compostas de diversos componentes espalhados em uma rede. Um segundo ponto a ser observado, é que a opção seguida por Odyssey de alterar as chamadas ao sistema NetBSD dificulta sua utilização em ambientes altamente heterogêneos, típicos da computação móvel. Odyssey foi projetado para um grupo de aplicações específicas de acesso a informações em ambientes de computação móvel. Finalmente, a adaptação se restringe a transformações nos dados sendo transmitidos, sendo que em nosso modelo, a adaptação pode tomar outras formas além desta.

Chang e Karamcheti [3] descrevem um *framework* cujo objetivo central é eliminar a necessidade de se programar explicitamente decisões relativas à adaptação no código das aplicações. O *framework* é baseado em dois componentes: uma interface de adaptação e um ambiente de execução virtual. A interface de adaptação disponibiliza o suporte na linguagem de programação para expressar a disponibilidade de múltiplos caminhos de execução (algoritmos diferentes para realizar uma mesma tarefa) bem como definir parâmetros relativos à monitoração do progresso da execução da aplicação. A partir desta monitoração, pode-se então escolher qual caminho de execução (algoritmo) deve ser seguido. As informações sobre a interface de adaptação são inseridas pelo desenvolvedor como anotações no código fonte que são processadas por um pré-processador de macros. Este pré-processador também gera agentes de monitoração e controle, que monitoram a disponibilidade de recursos e controlam a execução da aplicação, respectivamente. O código destes agentes é inserido no código da aplicação. O segundo componente do *framework* é um ambiente de execução virtual que pode ser configurado para emular diversos cenários de disponibilidade de recursos. Um *script* executa repetidamente a aplicação nestes ambientes virtuais obtendo o perfil do comportamento da aplicação para cada um destes cenários. O sistema de suporte à execução utiliza estes perfis para tomar decisões sobre qual caminho seguir de forma a satisfazer as preferências definidas pelo usuário como, por exemplo, a otimização do tempo de transmissão dos dados entre o cliente e o servidor. A abordagem seguida por Chang e Karamcheti varia da nossa em diversos aspectos. Em [3], todos os diferentes caminhos (algoritmos) estão codificados em um processo monolítico, juntamente com os agentes de monitoração e controle. Em nossa abordagem, os componentes são dinamicamente carregados para a memória, podendo ser alterados, reconfigurados ou removidos quando necessário. No nosso modelo, o código da monitoração é separado do código da aplicação. Finalmente, o foco de Chang e Karamcheti é na adaptação de uma aplicação composta por um único componente e não em aplicações compostas por muitos componentes distribuídas na rede.

Conductor [17], é um *framework* que visa prover adaptação com relação à heterogeneidade e variações na condição de operação das redes de computadores. De forma semelhante ao que ocorre em *proxy ativo* e no Odyssey, a adaptação se dá pela transformação dos dados sendo transmitidos. No entanto, ao contrário das abordagens vistas anteriormente, Conductor permite que a adaptação se dê de forma coordenada em diversos pontos da rede, um aspecto importante para a aplicação de referência SIDAM. No entanto, a adaptação é restrita a transformações no fluxo de dados trocados por componentes da aplicação. O objetivo da adaptação é compensar diferenças decorrentes da heterogeneidade dos *links* da rede e no estado de operação da mesma. Finalmente, nenhum suporte é dado para lidar com a reconfiguração dinâmica dos componentes da aplicação.

VI. Conclusões

Tecnologias como a Internet, a computação móvel e espaços inteligentes tem contribuído para o surgimento de ambientes de computação cada vez mais dinâmicos. Este dinamismo necessita ser acompanhado pelo software através de mecanismos que possibilitem sua reconfiguração em tempo de execução de forma a acomodar variações do ambiente no qual o mesmo está sendo executado.

O modelo para adaptação descrito neste trabalho tem por objetivo servir como base para o desenvolvimento de uma ferramenta poderosa e flexível que seja um elemento chave para o desenvolvimento de aplicações distribuídas adaptáveis dinamicamente que suportem variações em seu ambiente de execução. Para atingir este objetivo, propomos uma arquitetura compreendida por três pacotes principais que realizam as seguintes tarefas:

1) **Monitor:** disponibiliza as ferramentas necessárias para monitorar as interações entre os objetos que compõem a aplicação distribuída, os recursos disponíveis no sistema, a localização dos dispositivos móveis e a descoberta de novos serviços e recursos à medida que o usuário se desloca com seu equipamento;

2) **Detector de Eventos:** disponibiliza as ferramentas para a avaliação dos dados coletados pela monitoração de forma a detectar os eventos de interesse ao processo de adaptação. Disponibiliza ainda uma ferramenta de notificação destes eventos aos objetos que registrarem interesse nos mesmos;
3) **Reconfigurador Dinâmico:** provê ferramentas através das quais pode-se definir as ações necessárias para reconfigurar a aplicação de forma a acomodar as mudanças ocorridas no ambiente. O reconfigurador dinâmico utiliza informações a respeito das dependências tanto estáticas quanto dinâmicas entre os componentes da aplicação de forma a realizar a reconfiguração de forma segura.

Atualmente nossa pesquisa encontra-se no estágio de implementação do modelo aqui apresentado. Temos implementado o repositório de definições dos parâmetros de recursos a serem monitorados e das entidades do sistema que sofrerão esta monitoração bem como objetos de monitoração de recursos que controlam o percentual de uso dos processadores e a memória real disponível em nós do ambiente distribuído. Estamos atualmente implementando o avaliador de eventos para detecção de eventos de recursos e em breve pretendemos integra-lo ao código do PSIT, um protótipo da aplicação de referência do SIDAM já desenvolvido no Departamento de Ciência da Computação do IME/USP.

REFERÊNCIAS

[1] B. R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. In *1993. In Proceedings of the Fourth Workshop on Workstation Operating Systems*, Napa, California, October 1993.

[2] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1999.

[3] Fangzhe Chang and Vijay Karamcheti. Automatic configuration and run-time adaptation of distributed applications. In IEEE Computer Society, editor, *Ninth IEEE International Symposium on High Performance Distributed Computing*, Pittsburg, Pennsylvania, aug 2000.

[4] M. Endler, D.M. da Silva, F. Silva e Silva, R.A. da Rocha, and M.A. de Moura. Project SIDAM: Overview and Preliminary Results. In *Anais do 2o. Workshop de Comunicação sem Fio (WCSF), Belo Horizonte*, May 2000.

[5] Markus Endler and Francisco José da Silva e Silva. Requisitos e arquiteturas de software para computação móvel. In *Workshop sobre Métodos e Servi cos para Computação Móvel*, São Paulo, SP, oct 2000.

[6] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, abr 1994.

[7] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to network and client variation using active proxies: Lessons and perspectives. *special issue of IEEE Personal Communications on Adaptation*, 1998.

[8] Fabio Kon and Roy H. Campbell. Dependence management in component-based distributed systems. *IEEE Concurrency*, 8(1), January/March 2000.

[9] Fabio Kon, Manuel Román, Ping Liu, Jina Mao, Tomonori Yamane, Luiz Claudio Magalhães, and Roy H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, number 1795 in LNCS, pages 121–143, New York, April 2000. Springer-Verlag.

[10] Jina Mao. Monitoring and Analyzing Method Invocations in the 2K Operating System. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1999.

[11] Douglas Moreto. Monitoramento de eventos compostos em sistemas distribuidos. Master's thesis, Instituto de Matematica e Estatistica, Universidade de Sao Paulo, sep 1998.

[12] B. Noble and M. Satyanarayanan. Experience with adaptive, mobile applications in odyssey. *Mobile Network and Applications*, 4:245–254, 1999.

[13] OMG - Object Management Group. *The Common Object Request Broker: Architecture and Specification*, oct 2000. version 2.4.

[14] Evaggelia Pitoura and George Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publisher, 1998.

[15] Manuel Román and Roy H. Campbell. Gaia: Enabling Active Spaces. In *Proceedings of the 9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 2000.

[16] P. Sudame and B. Badrinath. On providing support for protocol adaptation in mobile wireless networks. Technical report, Department of Computer Science, Rutgers Universit, jun 1997. http://www.cs.rutgers.edu/pub/technical-reports/dcstr - 333.ps.Z.

[17] Mark Yarvis, Peter Reiher, and Gerald J. Popek. Conductor: A framework for distributed adaptation. In *Seventh Workshop on Hot Topics in Operating Systems*. IEEE Computer Society Press, March 1999.