5
Implementação

Este capítulo é dedicado a apresentar a implementação da arquitetura MoGrid e, mais especificamente, do protocolo P2PDP, trazendo as suposições adotadas para o seu desenvolvimento, as decisões de projeto quanto às tecnologias utilizadas, além dos cenários de uso que constituem o protocolo de descoberta e seleção de recursos e serviços. Esses cenários de uso são representados por alguns diagramas de seqüência, com o intuito de facilitar a compreensão do comportamento das entidades envolvidas no processo de descoberta e seleção de recursos e serviços. Além disso, são apresentados os detalhes da integração do *middleware* MoGrid com uma grade fixa através do Globus Toolkit [Foster & Kesselman 1997]. Este capítulo traz também maiores detalhes sobre a utilização do protocolo P2PDP e da API fornecida pela arquitetura MoGrid, através do desenvolvimento de aplicações colaborativas diversificadas e do teste de correção da implementação do protocolo utilizando o simulador NCTUns [Wang & Lin 2005].

5.1.
Tecnologias Utilizadas

A implementação foi desenvolvida na linguagem Java da SUN [Sun 1994], sendo adotado o perfil CDC (*Connected Device Configuration*) do J2ME [Sun 1999b] como plataforma de referência para a implementação. Uma versão simplificada da implementação também foi desenvolvida adotando o perfil CLDC/MIDP (*Connected Limited Device Configuration/Mobile Information Device Profile*). Nessa versão, foram implementadas somente as entidades iniciador e coordenador do protocolo P2PDP, devido às restrições do perfil CLDC/MIDP no que se refere ao controle de execução de tarefas, mecanismos de temporização e ao monitoramento de recursos, que são essenciais para a entidade colaborador. As entidades de descoberta – iniciador, coordenador e colaborador – foram explicadas em detalhes na Subseção 2.3.1.2 e na Seção 4.5. A linguagem Java foi

escolhida por ser uma linguagem multiplataforma e oferecer recursos de programação para dispositivos móveis.

O serviço monitor (Subseção 2.3.1.2), utilizado nas versões desenvolvidas para Windows XP e Linux, corresponde à implementação disponível na arquitetura MoCA (*Mobile Collaboration Architecture*) [Sacramento et al. 2004]. Esse serviço foi desenvolvido como um *daemon* executando em cada dispositivo móvel, responsável por coletar informações de estado, tais como a qualidade do enlace sem fio, nível de energia, uso de CPU e memória disponível. Essas informações são processadas pelo gerente de contexto (Subseção 2.3.1.2), que alimenta o colaborador, informando-o, quando solicitado, sobre a disponibilidade de recursos do dispositivo. Essa informação será utilizada pelo algoritmo DR (Seção 4.7) para o cálculo do retardo de envio de mensagens de resposta. A utilização dessa implementação do monitor deve-se ao fato dos estudos iniciais que conduziram a este trabalho terem sido desenvolvidos durante o processo de especificação da arquitetura MoCA.

Para que os dispositivos portáteis possam atuar como uma interface de acesso à grade fixa, utilizando os serviços e recursos que ela disponibiliza, foi necessária a definição de uma nova entidade na arquitetura MoGrid, capaz de fazer a tradução do protocolo P2PDP para os protocolos usados em grades fixas. Essa entidade, denominada *proxy* de colaboração, atua como um intermediador na descoberta e seleção de recursos e submissão de tarefas em uma grade fixa, através dos dispositivos provenientes de uma grade móvel, como ilustrado na Figura 26. O conceito do *proxy* de colaboração é genérico na arquitetura MoGrid, podendo ser especializado para tecnologias de grade fixa particulares. A implementação do *proxy* de colaboração desenvolvida nesta tese é responsável pela integração do MoGrid com o Globus [Foster & Kesselman 1997] – um *toolkit* de *software* para construção de grades fixas. O *proxy* deve ser um dispositivo presente, simultaneamente, na grade móvel e na grade fixa; qualquer dispositivo que hospede o *software* cliente do Globus e o *middleware* MoGrid é passível de se tornar um *proxy* de colaboração. A escolha do Globus Toolkit deve-

se ao fato da grade fixa do projeto ComCiDis[1], na qual foram realizados os experimentos com o *proxy* de colaboração, ter sido implantada utilizando essa tecnologia.

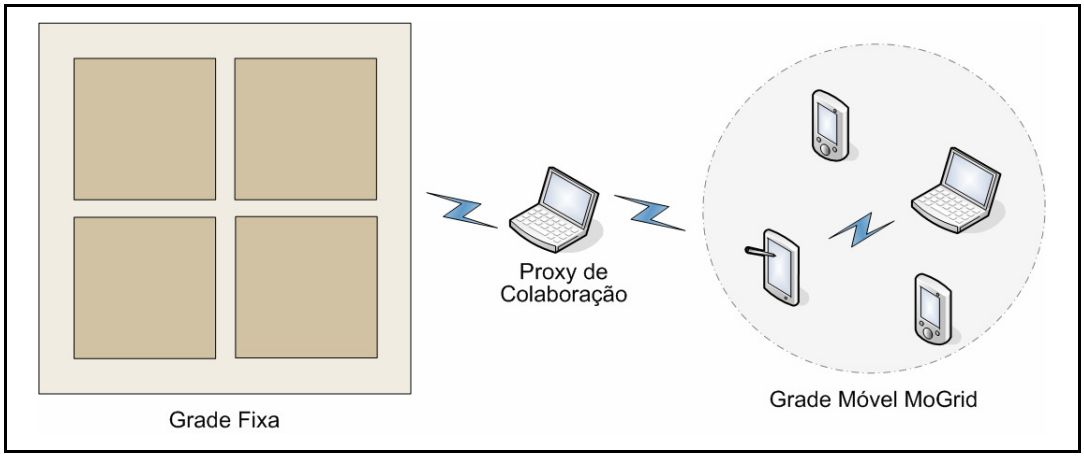

Figura 26 – O *proxy* de colaboração MoGrid.

5.2. Implementação do Protocolo P2PDP

As próximas subseções trazem detalhes sobre a implementação do protocolo de descoberta P2P. Foi adotado o paradigma de orientação a objetos tanto na modelagem quanto na implementação. As principais classes do protocolo, bem como o relacionamento entre elas, são apresentados através de diagramas de classe e de seqüência UML (*Unified Modeling Language*) [OMG 2005], com o intuito de facilitar a compreensão do comportamento das entidades envolvidas no processo de descoberta e seleção de serviços. Nos diagramas de classe, foi utilizada uma grafia diferenciada para representar as classes abstratas (itálico) e as concretas (regular). Além disso, classes que implementam interfaces têm essas interfaces identificadas pelo estereótipo "<<*nome_da_interface*>>".

[1] A página do projeto ComCiDis está disponível em <http://comcidis.lncc.br/>. Acesso em: 15 mai. 2007.

5.2.1.
Descrição de Recursos e Controle de Admissão

A Figura 27 ilustra as classes responsáveis pela descrição dos recursos e pelo controle de admissão do protocolo.

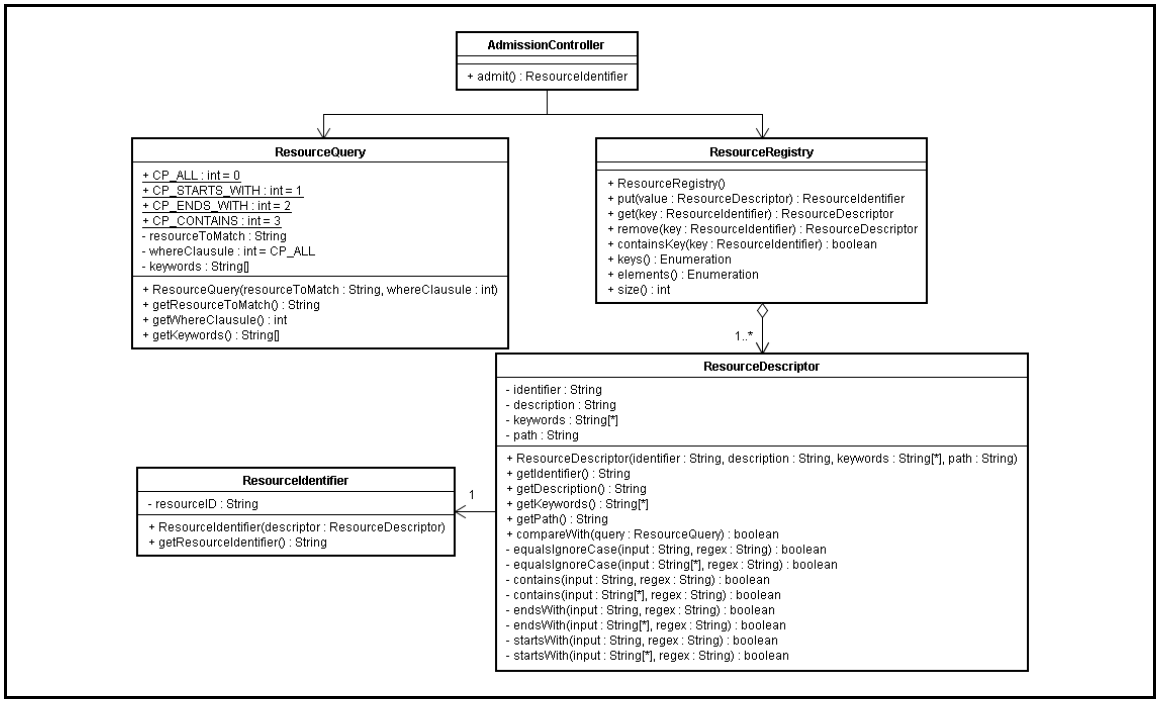

Figura 27 – Descrição de recursos e controle de admissão.

Cada recurso disponível na grade é representado por uma instância da classe *ResourceDescriptor* e é registrado, pelo dispositivo que o disponibiliza, através do método `put()` da classe *ResourceRegistry*. Ao ser registrado, o recurso é associado a um identificador único universal, representado pela classe *ResourceIdentifier*. Ao receber uma requisição de serviço, o colaborador consulta o mecanismo de controle de admissão para verificar se o serviço solicitado é oferecido pelo dispositivo; esse procedimento é efetuado através do método `admit()` da classe *AdmissionController*.

5.2.2.
Mensagens de Controle P2PDP

As mensagens de controle do protocolo de descoberta P2PDP, descritas na Seção 4.5, são representadas como subclasses da classe abstrata *P2PDPMessageImpl*, como ilustrado na Figura 28. A manipulação de qualquer campo presente nessas

mensagens pelas entidades do protocolo ocorre por intermédio desse conjunto de classes. A classe *P2PDPMessageImpl* implementa os métodos que manipulam campos comuns a todas as mensagens de controle P2PDP – `IReq`, `CRep` e `CRepList`. Esses métodos são definidos na interface <<*P2PDPMessage*>>.

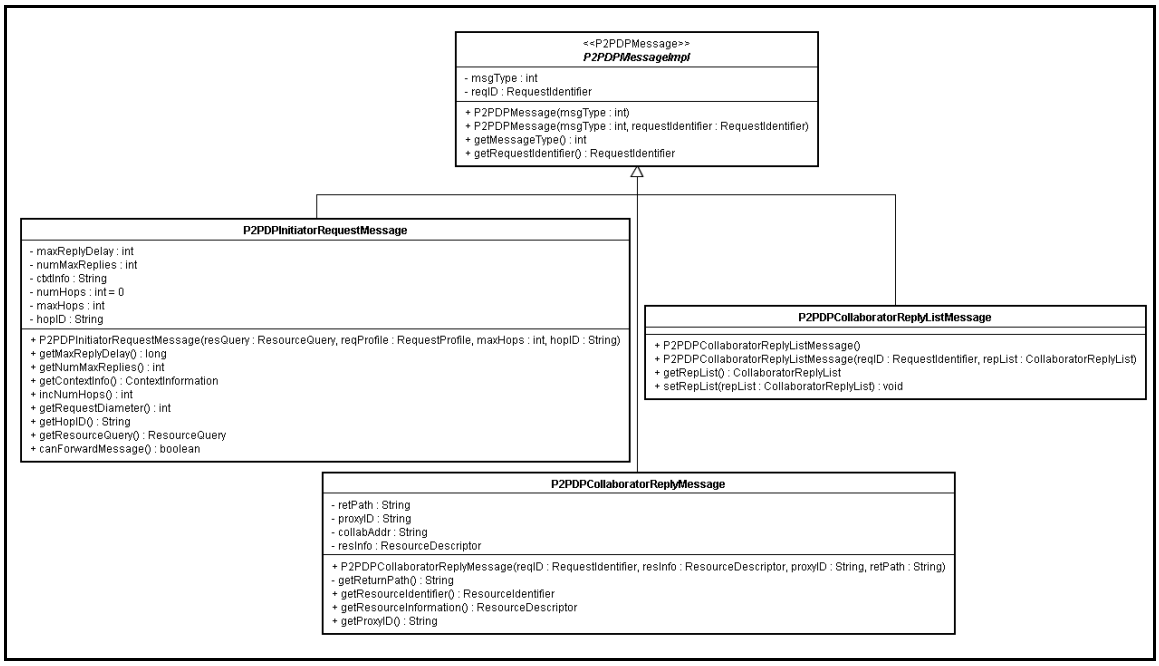

Figura 28 – Declaração das mensagens de controle P2PDP.

5.2.3. Canal de Comunicação P2PDP

As mensagens do protocolo P2PDP trafegam na MANET utilizando canais de comunicação criados a partir da classe *P2PDPConnectionFactory*, como ilustrado na Figura 29. Na implementação do protocolo P2PDP para redes sem fio *ad hoc*, apenas um canal de comunicação *broadcast* é criado, o qual é representado por uma instância da classe *P2PDPBroadcastConnection*. Isso se deve ao fato de que, nesse tipo de rede, as entidades iniciador e coordenador são implementadas no mesmo dispositivo, comunicando-se diretamente, através de chamadas a métodos das classes *Initiator* e *Coordinator*, ilustradas na Figura 30, sem a necessidade de criação de um canal de comunicação específico entre elas (classe *P2PDPUnicastConnection*). Em uma implementação do P2PDP para uma rede sem fio infra-estruturada, o coordenador pode ser implementado na estação base, que é o dispositivo mais rico em recursos da rede sem fio, podendo atuar como coordenador para todos os iniciadores da grade móvel. Nesse caso, a comunicação

entre iniciador e coordenador é feita através de um canal de comunicação *unicast* (classe *P2PDPUnicastConnection*); a comunicação entre coordenador e colaboradores utiliza, assim como nas redes sem fio *ad hoc*, o canal de comunicação *broadcast*.

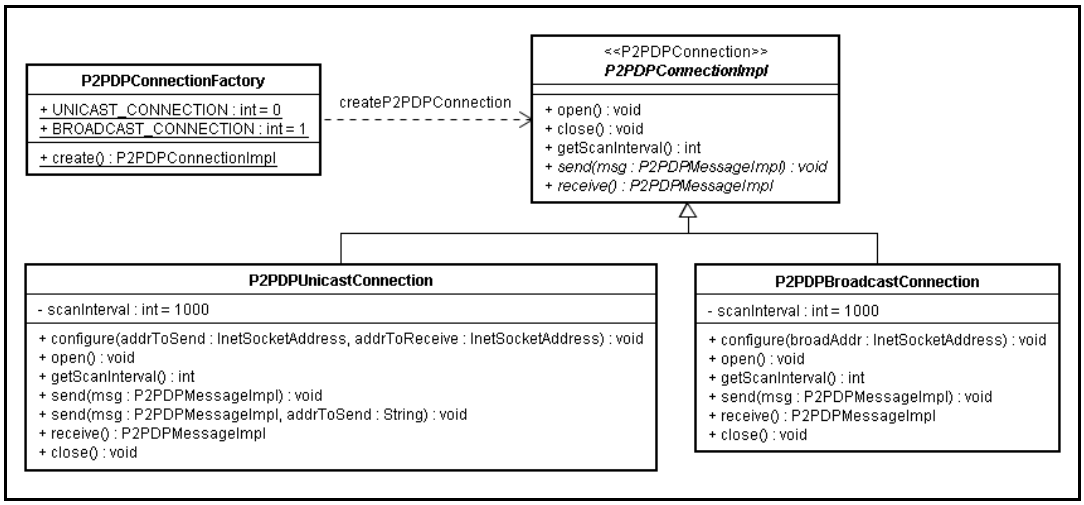

Figura 29 – Canal de comunicação P2PDP.

As classes *P2PDPUnicastConnection* e *P2PDPBroadcastConnection* são representadas na Figura 30 como subclasses da classe abstrata *P2PDPConnectionImpl* que implementa a interface *<<P2PDPConnection>>*. Essas classes oferecem às entidades do protocolo P2PDP os mecanismos para acesso ao canal de comunicação oferecido pela plataforma de *hardware* ou *software* – no caso, o sistema operacional do dispositivo sem fio.

5.2.4. Entidades de Descoberta P2PDP

A Figura 30 ilustra as classes que representam as entidades de descoberta – iniciador (classe *Initiator*) e coordenador (classe *Coordinator*) – e a interface que representa as operações fornecidas pelas subcamadas de adaptação (ASLs) da camada de transparência (interface *<<AdaptationSublayer>>*), que foram descritas na Subseção 2.3.2.3. Uma aplicação MoGrid inicia o processo de descoberta acionando o método `submitRequest()` de uma classe que implementa *<<AdaptationSublayer>>*. Essa aplicação desenvolvida deve implementar a interface *<<DiscoveryApplicationFacade>>*, que define o método `receiveReplyList()` para recepção dos resultados do processo de descoberta.

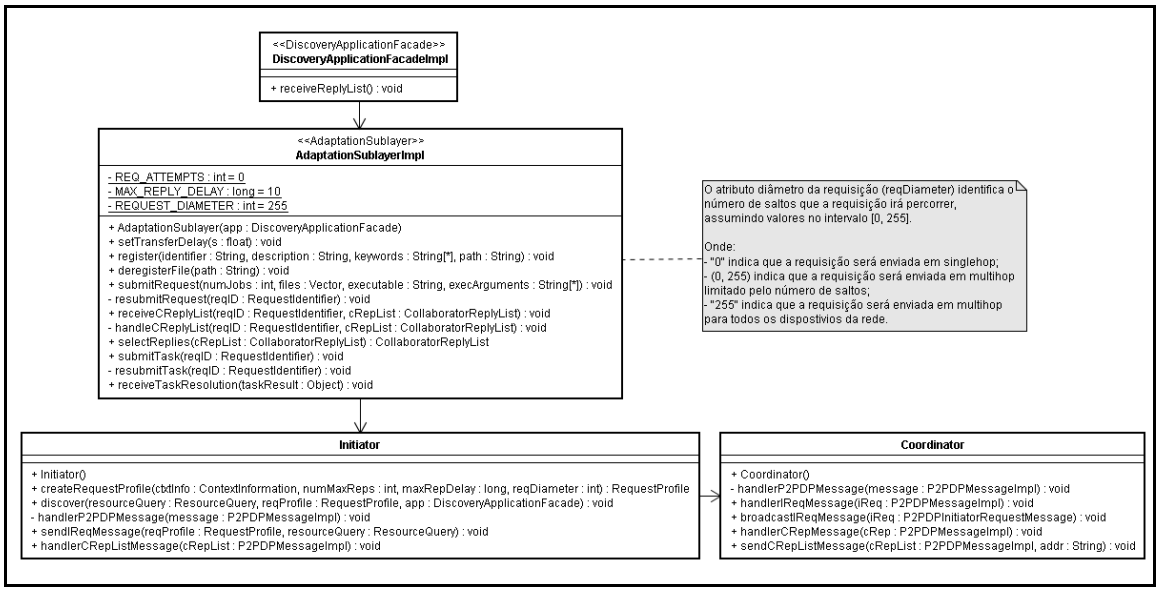

Figura 30 – Entidades iniciador e coordenador e a subcamada de adaptação.

A Figura 30 ilustra também uma implementação *default* de ASL (classe *AdaptationSublayerImpl*). Aplicações que apresentem características específicas podem demandar o desenvolvimento de ASLs também específicas; isso é conseguido por meio de classes que implementam a interface <<*AdaptationSublayer*>>. Um exemplo que ilustra a necessidade do desenvolvimento de ASLs específicas é a implementação do método `selectReplies()`. Apesar do protocolo P2PDP implementar a seleção das melhores respostas ao longo da rede (Seções 4.6 e 4.7), em algumas situações o coordenador pode receber mais respostas do que as que foram solicitadas pelo iniciador. Para atender a esses casos, o método `selectReplies()` é responsável por efetuar a seleção de respostas, mantendo o processo de seleção automático, sem a necessidade de intervenção direta do cliente da aplicação. No caso da classe *default AdaptationSublayerImpl*, esse método devolve simplesmente as primeiras respostas que chegam ao coordenador. Outras aplicações podem implementar outras estratégias de seleção nesse método por intermédio de classes específicas que implementam <<*AdaptationSublayer*>>, conforme discutido ao final da Subseção 2.3.1.2.

A Figura 31 traz as classes que representam a entidade colaborador (classe *Collaborator*), o mecanismo de controle de admissão (classe *AdmissionController*) e os serviços responsáveis pelo monitoramento das

informações dos recursos do dispositivo (classe *MoCAMonitor*) e pela inferência do seu nível de colaboração (classe *ContextListener*).

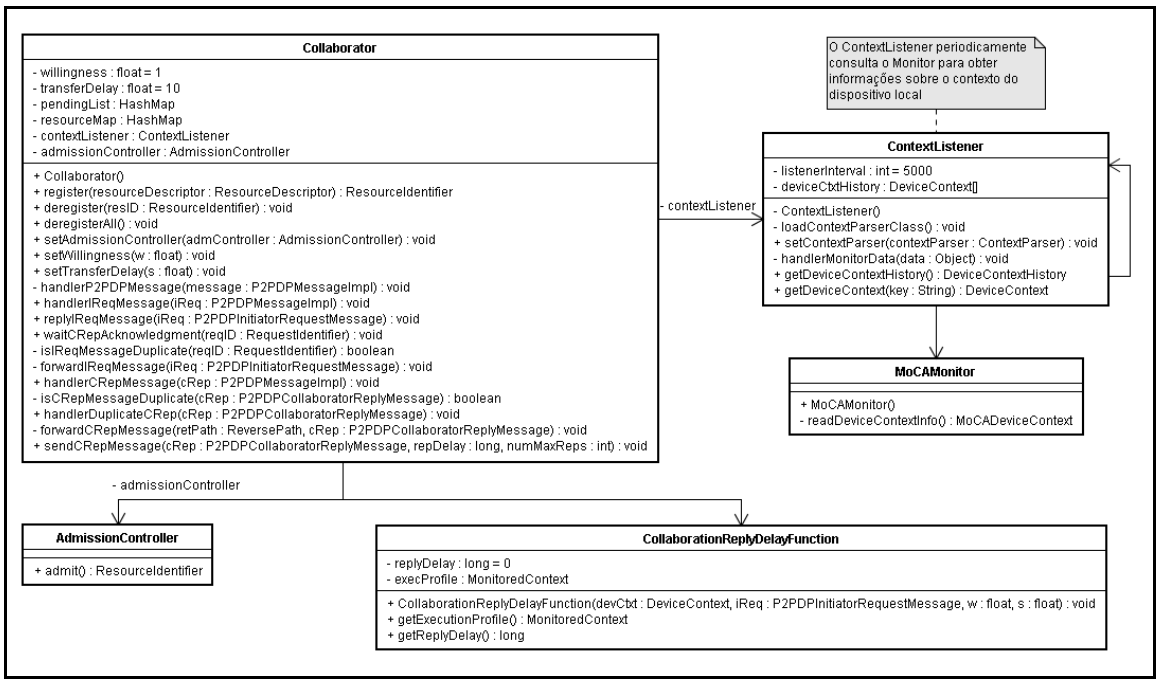

Figura 31 – Entidade colaborador.

Ao processar uma requisição de serviço, através do método `handlerIReqMessage()`, o colaborador primeiro verifica se a requisição já foi tratada por ele (`isIReqMessageDuplicate()`), descartando-a em caso afirmativo. Caso contrário, o colaborador consulta o atributo `willingness`, para definir se está disposto ($0 < $ `willingness` ≤ 1) ou não (`willingness` $= 0$) a colaborar. Se o dispositivo está atuando no modo de colaboração ativa, ele então verifica se oferece o serviço solicitado, através do método `admit()`, da classe *AdmissionController*. No modo de colaboração ativa, o dispositivo não somente auxilia no encaminhamento das mensagens P2PDP – o que caracteriza o modo passivo – como também disponibiliza os seus recursos para os demais na rede. Se o colaborador possui o serviço em questão, ele então gera uma mensagem de resposta (`CRep`) e escalona o seu envio através do método `sendCRepMessage()`. O algoritmo DR, implementado através da classe *CollaborationReplyDelay-Function*, é acionado para calcular o instante de tempo (`getReplyDelay()`) em que a resposta do colaborador será enviada, bem como coletar a informação sobre o perfil de execução do dispositivo (`getExecutionProfile()`). Essa informação é transmitida na mensagem `CRep`, podendo ser usada pela ASL da aplicação que

requisitou o serviço como critério de seleção das melhores respostas, quando essas ultrapassam o número solicitado. Logo após o envio da sua resposta, o colaborador aciona o método `waitCRepAcknowledgment()` e fica aguardando pelo reconhecimento implícito de que a sua mensagem foi encaminhada, na rede, em direção ao nó que requisitou o serviço (pseudocódigo da Figura 25). Se o reconhecimento não for recebido (`handlerDuplicateCRep()`) dentro de um intervalo de tempo específico, então o colaborador reenvia a sua resposta, dessa vez, tendo como próximo salto no caminho de retorno todos os seus vizinhos diretos, o que é definido pela utilização do endereço de *broadcast*, conforme é descrito na Seção 4.8.

É importante mencionar que o valor do atributo `willingness` (ω), definido na Equação (2) apresentada no Capítulo 4, foi calculado como $\omega_{usr}/2L$, onde ω_{usr} representa o nível de interesse (no intervalo [0, 1]) do usuário em permitir que o seu dispositivo colabore com os outros na MANET e *L* representa o número de requisições que foram respondidas pelo dispositivo. Desta forma, implementou-se um mecanismo de adaptação simples, que oferece uma reserva antecipada de recursos na medida em que reduz o indicador da disposição do dispositivo em colaborar (*willingness*) quando o número de requisições pendentes aumenta. A adaptação do valor de ω tem como objetivo não apenas oferecer níveis mínimos de garantia de que o recurso requisitado esteja disponível quando a execução da tarefa for iniciada, como também preservar o dispositivo colaborador, evitando que ele fique sobrecarregado por atender mais requisições do que a sua capacidade permite. O ajuste desse parâmetro facilita o controle de admissão pois se ω atinge o seu valor mínimo ($\omega = 0$), o colaborador passa a não responder às requisições da grade, mesmo que ele ofereça os serviços solicitados. Caso esse parâmetro seja mantido no seu valor máximo ($\omega = 1$) durante a participação do dispositivo na grade móvel, mesmo assim o cálculo do retardo no envio das mensagens garante que o temporizador τ seja configurado para um valor elevado, próximo ao limite definido pelo iniciador da requisição (`maxRepDelay`), fazendo com que a sua resposta seja preterida, sendo retirada da rede pelo mecanismo de supressão (Seção 4.8).

5.2.5.
Cenários de Uso

Esta subseção ilustra os aspectos dinâmicos da colaboração entre as entidades de descoberta e os serviços de monitoração, através da utilização de diagramas de seqüência.

A seqüência de invocação de métodos referente à descoberta de serviços, no P2PDP, é ilustrada pelo diagrama apresentado na Figura 32.

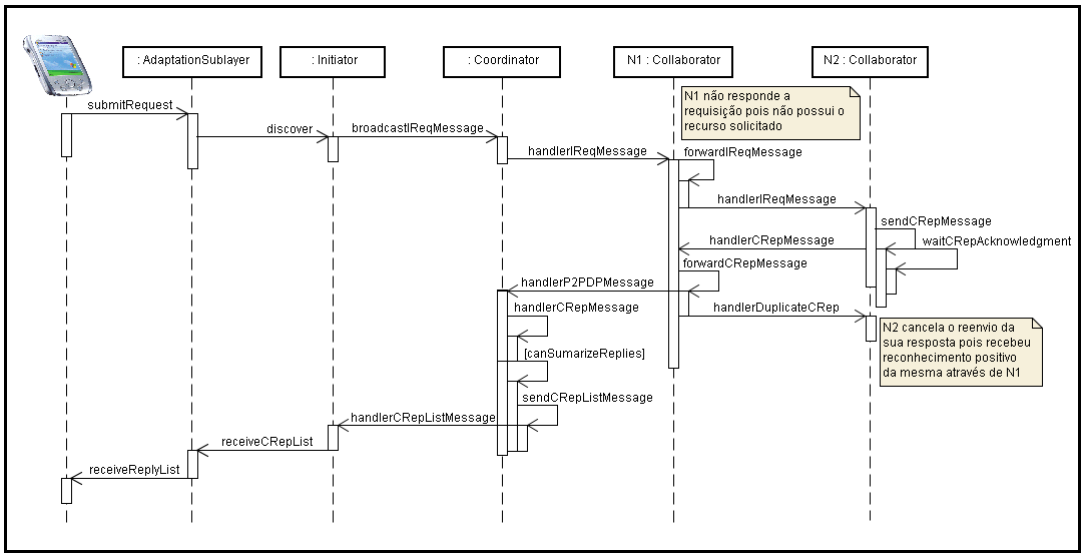

Figura 32 – Seqüência de invocação de operações associadas à descoberta de serviço.

A seqüência de invocação de métodos referente ao tratamento de uma requisição de serviço pela entidade colaborador (classe *Collaborator*), no P2PDP, é ilustrada pelo diagrama apresentado na Figura 33.

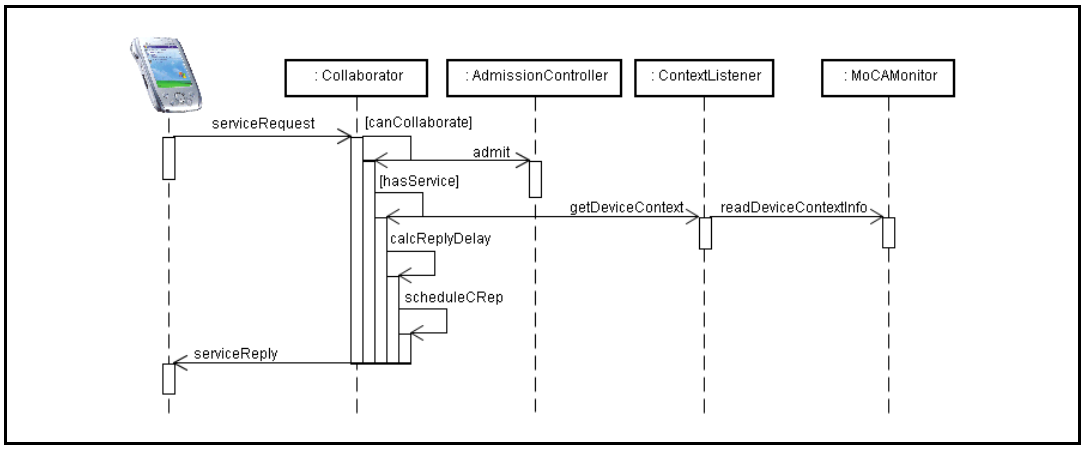

Figura 33 – Seqüência de invocação de operações associadas à recepção de uma requisição de serviço no colaborador.

5.3. Integração do MoGrid com o Globus Toolkit

Um desafio no desenvolvimento de aplicações para grades móveis é a possibilidade de integração com grades fixas convencionais. Embora grades móveis possam, a princípio, ser pensadas como agrupamentos isolados de processamento, a aproximação de uma grade móvel a um ponto de acesso para uma rede fixa pode expandir, consideravelmente, a capacidade computacional da grade móvel. Nesta seção, os serviços que compõem o Globus Toolkit são brevemente apresentados, assim como as adaptações que se fizeram necessárias na arquitetura MoGrid, especialmente, na camada de descoberta, para que os dispositivos que constituem a grade móvel pudessem ter acesso aos recursos da grade fixa Globus de uma forma transparente. Nessa direção, são discutidos aspectos como a implementação do monitoramento dos recursos e do mecanismo de controle de admissão da grade fixa.

5.3.1. O Globus Toolkit

O Globus *Toolkit* (GTK) é um conjunto de ferramentas usado na criação de sistemas e aplicações em grades. O GTK permite que problemas computacionalmente complexos sejam executados rapidamente, dividindo-os em partes e os submetendo, sob a forma de tarefas (processos), às máquinas da grade, de acordo com os recursos disponíveis nas mesmas e com as necessidades específicas do problema. Cada tarefa deve ser instanciada na máquina remota em que for executada; para isso, um programa executável (ou *script*) que implemente a tarefa pode ter que ser previamente copiado para a máquina remota. O Globus oferece mecanismos para efetuar a transferência de arquivos entre as máquinas que constituem a grade.

Para prover essas funcionalidades, o Globus *Toolkit* oferece vários serviços:

- ***Grid Security Infrastructure*** (**GSI**). Provê segurança e permite um *login* único de usuário em toda a grade. Dessa forma, um usuário não precisa efetuar *login* em cada máquina usada no processamento de suas tarefas, mas utiliza uma única credencial, que fornece o acesso temporário a todas as máquinas usadas na grade;

- ***Globus Resource Allocation Manager*** (**GRAM**). Provê mecanismos de alocação de recursos e de criação e monitoração de tarefas, informando o andamento das tarefas durante o seu tempo de execução;
- ***Grid File Transfer Protocol*** (**GridFTP**). Responsável pela transferência de arquivos (por exemplo, para cópia de executáveis) entre as máquinas da grade, por meio do protocolo FTP;
- ***Monitoring and Discovery Service*** (**MDS**). Provê uma área de acesso público que disponibiliza informações sobre a grade. O MDS provê dois tipos de serviço:
  - ***Grid Resource Information Service*** (**GRIS**). Presente localmente em cada máquina da grade; envia informações sobre a máquina (sistema operacional, CPU livre, memória disponível, etc.) para uma determinada máquina na grade, que hospeda o GIIS;
  - ***Grid Index Information Service*** (**GIIS**). Recebe as informações, enviadas pelos GRIS, que estão localmente em cada máquina da grade. Desta forma, o GIIS é capaz de consolidar informações sobre a grade como um todo, oferecendo, às máquinas que têm acesso ao GIIS, informações específicas sobre um recurso qualquer.

O GTK disponibiliza o CoG Kit [CoG 1997], implementado em várias linguagens, como Python e Java, para oferecer facilidades no desenvolvimento de aplicações para a grade Globus. Essa API permite que programadores implementem aplicações que possam ter acesso a serviços Globus, tais como submissão de tarefas (GSI, GRAM), transferência de arquivos (GridFTP) e descoberta de recursos (MDS). A implementação Java dessa API foi utilizada na implementação da integração da grade Globus à arquitetura MoGrid.

5.3.2. Requisições de Descoberta P2PDP em uma Grade Globus

Quando um usuário MoGrid requisita, no dispositivo móvel, a descoberta de recursos ou a submissão de uma tarefa, e essa requisição é recebida pelo *proxy* de colaboração, este se autentica na grade Globus apresentando um certificado de usuário e obtendo uma credencial temporária, válida enquanto durar a sessão. Não

é necessário que um usuário MoGrid possua esse certificado, porém é necessário que o *proxy* autentique um usuário Globus junto ao serviço GSI, ou seja, o usuário MoGrid personifica um usuário Globus através do *proxy*. Logo, qualquer máquina com acesso à grade Globus, e que tenha o MoGrid instalado, está apta a atuar como um *proxy* para a submissão de tarefas da grade móvel para a grade fixa.[2]

Ao receber uma requisição P2PDP, o *proxy* só enviará, ao usuário MoGrid requisitante, informações sobre os recursos dos nós da grade Globus em que puder se autenticar para a submissão de tarefas. O *proxy* envia ao cliente MoGrid exatamente o número de respostas solicitadas na requisição, encarregando-se de selecionar previamente os melhores colaboradores dentre as máquinas na grade fixa aptas a atender a requisição. O *proxy* torna transparente para o usuário MoGrid a existência de uma grade fixa e os nós da grade Globus são vistos como nós MoGrid alcançáveis através do *proxy*. Já na execução de uma tarefa, o *proxy* recebe, do usuário MoGrid requisitante, o nome do arquivo executável, seus argumentos, os arquivos que devem ser copiados para a execução e o endereço do "nó MoGrid" (na realidade, um nó na grade Globus) em que a tarefa será executada. O *proxy* é responsável também pela gerência do ciclo de vida dos arquivos que devem estar na máquina remota no momento da execução, ou seja, o *proxy* deve copiá-los antes que a execução da tarefa tenha início e removê-los logo após a sua conclusão.

Para possibilitar a comunicação entre os dispositivos MoGrid e Globus, foi necessário acrescentar um campo à mensagem de resposta `CRep` do P2PDP. Esse novo campo, denominado `proxyID`, fornece o identificador do dispositivo que atua como *proxy* entre as duas grades. Para nós que não atuam como *proxies*, `proxyID` = 0. Dessa forma, o processo de submissão de tarefas obedece a mesma lógica tanto na grade móvel quanto na grade fixa. A Figura 34 traz a representação da mensagem `CRep` modificada.

[2] Deve-se ressaltar que as questões relacionadas ao mapeamento do usuário Globus para usuários MoGrid não são consideradas nesta tese. Essas questões dizem respeito aos mecanismos de segurança, que de uma forma global não são tratados nesta tese.

```
CRep: <reqID, collabAddr, proxyID, resInfo, retPath>
```

Figura 34 – Mensagem `CollaboratorReply` (`CRep`) modificada.

5.3.3. Monitoramento de Recursos em uma Grade Globus

Para que se possa implementar o mecanismo de seleção dos dispositivos mais aptos a executar as tarefas da grade através do mecanismo de retardo no envio das mensagens de resposta, é necessário realizar o monitoramento das informações de contexto dos dispositivos na grade fixa, para que se possa efetuar o cálculo do temporizador `replyDelay` (vide Equação (2) na Seção 4.7).

Para a escolha das máquinas responsáveis pela execução das tarefas na grade fixa – considerando-se que a qualidade do enlace sem fio e o nível de energia são constantes –, os recursos levados em consideração, na especificação do perfil de execução, foram CPU, memória e espaço em disco. O monitoramento desses recursos no GTK é feito através do MDS. Como o monitor MoCA, utilizado no *middleware* MoGrid, passa as informações sobre os recursos de uma forma diferente da utilizada no MDS, foi necessário desenvolver uma interface padrão para a descoberta de recursos que homogeneizasse essas informações, permitindo, assim, a integração dos mecanismos de monitoramento de recursos entre a grade móvel e a grade fixa.

Diferentemente do monitor MoCA utilizado no MoGrid, que é executado como um *daemon* em cada dispositivo móvel, o monitor do Globus foi implementado, no *middleware* MoGrid, como um cliente do MDS que recebe informações sobre os recursos de todas as máquinas da grade fixa. No monitor do Globus (classe *GlobusMonitor*), essas informações são tratadas, sendo separadas pela identificação da máquina e enviadas, uma a uma, ao *proxy* MoGrid que solicitou a informação, mantendo a interface padrão de monitoramento de recursos MoGrid.

5.3.4. Controle de Admissão em uma Grade Globus

O MDS trata apenas informações sobre os recursos físicos das máquinas, porém algumas aplicações podem exigir recursos lógicos, como programas e *scripts* desenvolvidos em uma linguagem específica, que precisem localizar o seu ambiente de execução. Isso acontece, por exemplo, com aplicações Java que precisam de uma máquina virtual para serem executadas. Na API de descoberta MoGrid, existem operações para registro e remoção de serviços (operações `register()` e `deregister()`). O registro é feito de forma distribuída, com cada dispositivo na grade móvel se responsabilizando por gerenciar as informações sobre os serviços que disponibiliza. Como as máquinas, na grade fixa, não têm conhecimento do *middleware* MoGrid, não existe um mecanismo de registro das informações de serviços que cada máquina disponibiliza. Ao receber uma requisição de descoberta, antes de verificar quais máquinas da grade fixa atendem ao perfil de execução do serviço requisitado, o *proxy* precisa determinar quais máquinas, da grade fixa, disponibilizam o serviço. Esse procedimento corresponde às ações de um controle de admissão, que é responsável por determinar a existência dos recursos solicitados – como *softwares* instalados nas máquinas da grade fixa – aos quais o *proxy* tem acesso. Como o GTK só oferece suporte à execução de tarefas em máquinas executando sistemas operacionais da família UNIX, a solução para determinar se um dado serviço existe na grade fixa foi utilizar *shell scripts*. Cada serviço é associado a um determinado *script*, com uma lógica específica, para verificar a sua existência. Caso o serviço seja encontrado, esse *script* deve retornar a sua localização exata na máquina remota; caso contrário, o resultado é vazio.

Para otimizar o mecanismo de controle de admissão ao inicializar o *proxy*, o seu administrador pode cadastrar *scripts* para localizar os serviços mais freqüentemente requisitados. Esses *scripts* podem ser executados de forma reativa ou proativa; independentemente da abordagem adotada, o resultado obtido é cadastrado, no *proxy*, através da operação `register()` da API de descoberta MoGrid, da mesma forma como é feito nos dispositivos da grade móvel. Na abordagem reativa, cada vez que o *proxy* recebe uma requisição, ele executa o *script* correspondente ao serviço solicitado nas máquinas da grade fixa – caso

exista algum *script* associado ao serviço. Na abordagem proativa, o *proxy* executa os *scripts* cadastrados periodicamente nas máquinas da grade fixa. Na implementação do *ProxyCollaborator*, a abordagem adotada foi a proativa.

5.3.5. Implementação do *Proxy* de Colaboração

Nesta seção, são apresentadas as extensões que se fizeram necessárias na entidade colaborador para que o *proxy* de colaboração pudesse ser implementado.

A Figura 35 traz as classes associadas ao *proxy* de colaboração; o relacionamento entre essas classes é similar ao que foi apresentado na Figura 31. As alterações introduzidas no *proxy* de colaboração (classe *ProxyCollaborator*), em relação ao colaborador, dizem respeito à autenticação do dispositivo na grade fixa Globus e ao acesso aos serviços de monitoramento oferecidos pelo GTK.

Ao ser instanciado, o *proxy* de colaboração se autentica na grade Globus através do método `sign()` da classe *ProxyInit*, que, por sua vez, ativa a criação do monitor do Globus (classe *GlobusMonitor*) através do método `createGlobusMonitor()` da classe *GridAuthentication*. O processamento de requisições de serviços e das respostas a essas requisições é efetuado de forma similar ao que foi descrito para o colaborador. A diferença é que, ao tratar uma requisição, o *proxy* gera tantas respostas quantas forem as máquinas disponíveis, na grade fixa, para atendê-las, conforme descrito na Subseção 5.3.2.

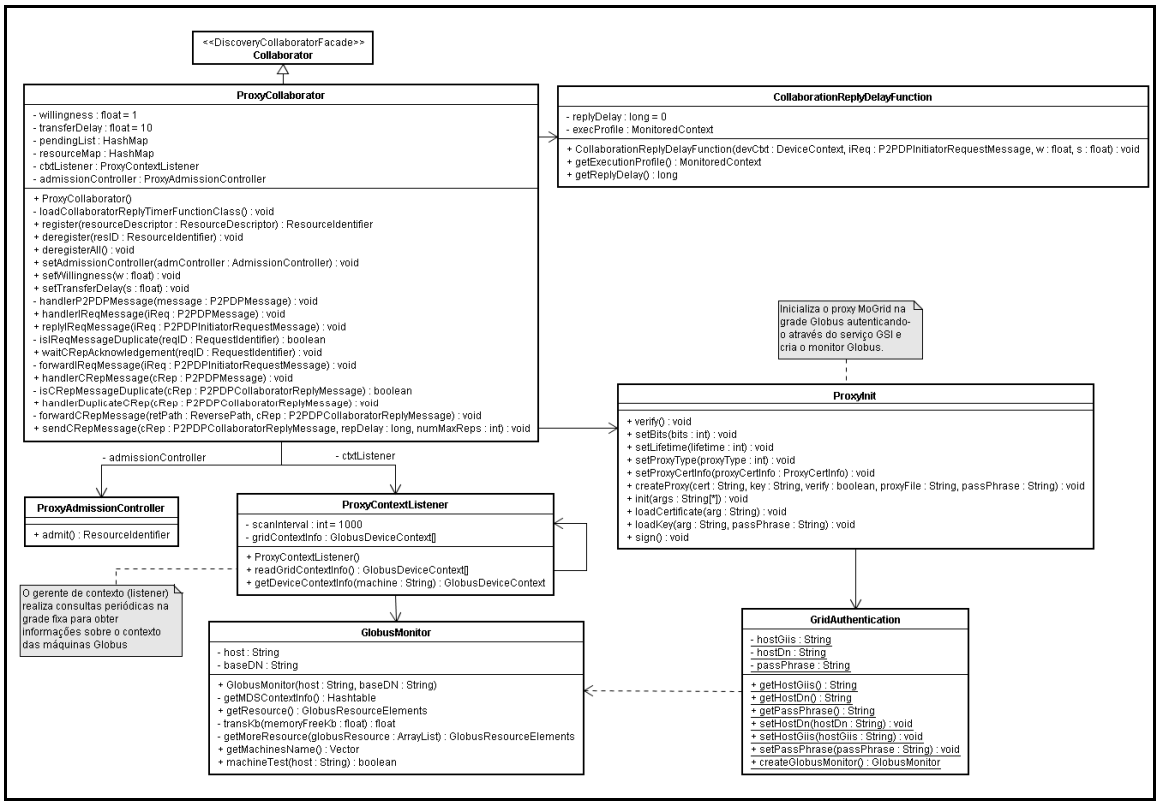

Figura 35 – Entidade *proxy* de colaboração.

A seqüência de invocação de métodos referente ao tratamento de uma requisição de serviço pelo *proxy* de colaboração no P2PDP é ilustrada na Figura 36.

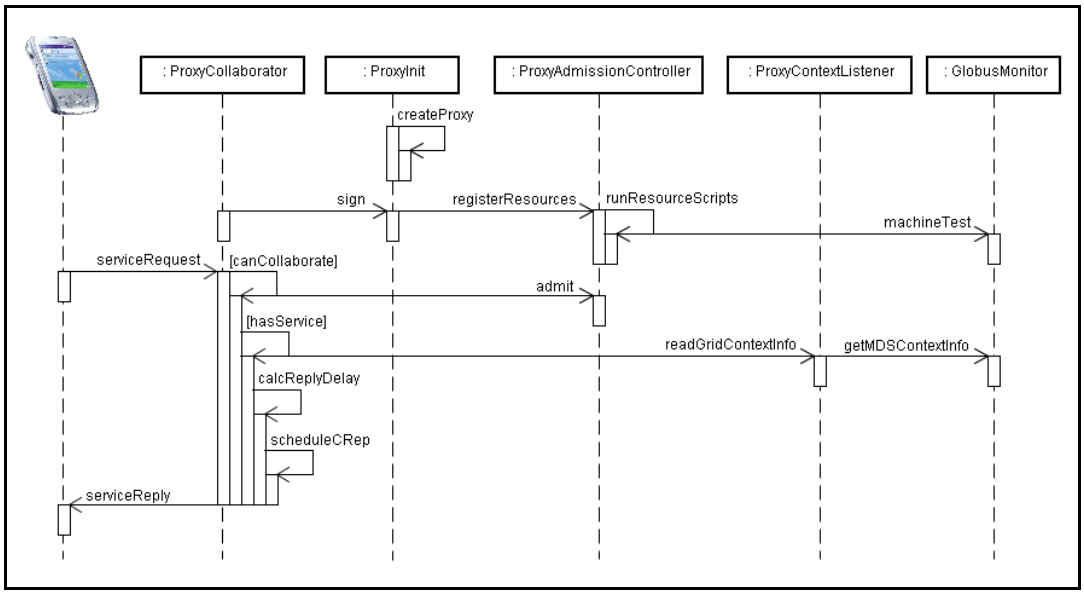

Figura 36 – Seqüência de invocação de operações associadas à recepção de uma requisição de serviço no *proxy* de colaboração.

5.4. Aplicações de Teste

Esta seção ilustra a utilização do protocolo P2PDP e da API de descoberta da arquitetura MoGrid por meio de um conjunto de aplicações colaborativas. Primeiramente, é apresentada uma aplicação P2P de compartilhamento de arquivos de áudio. A seguir, são apresentadas duas aplicações de processamento distribuído: uma aplicação de multiplicação de matrizes e uma outra que gerencia a execução de simulações paralelas do simulador *ns*-2 [ISI 1995].

A aplicação que gerencia o compartilhamento de arquivos de áudio, denominada ASA (*Audio Sharing Application*), foi inspirada em aplicações como Gnutella [Gnutella 2005] e Napster [Napster 2005]. O perfil de execução dessa aplicação é definido por uma subcamada de adaptação (ASL) desenvolvida especificamente para a família de aplicações de transferência de arquivos. Para essas aplicações, é importante balancear a qualidade do enlace sem fio e o tempo de autonomia da bateria do colaborador selecionado para prover o serviço requisitado. Portanto, para essa aplicação, o perfil de execução foi definido levando-se em consideração os recursos energia e a qualidade do enlace sem fio. Na implementação da ASL, optou-se por considerar os dois recursos com pesos iguais. Assim, os parâmetros da Equação (2) foram configurados como: $N = 2$ e $P_{energia} = P_{qualidade\ do\ enlace} = 1$. Nos testes realizados, foi utilizado o valor *default* de S (vide Seção 4.6), ou seja, $S = 10$ ms.

A segunda aplicação desenvolvida utilizando o protocolo P2PDP, denominada M3 (*Matrix-Matrix Multiplication*), usa uma abordagem mestre-escravo para multiplicação distribuída de matrizes. O objetivo dessa aplicação, desenvolvida como um "*toy problem*",[3] foi ilustrar a utilização de recursos tipicamente disponíveis em uma grade fixa por dispositivos portáteis em uma grade móvel. Nessa aplicação foi utilizado um algoritmo de multiplicação distribuída bastante simples: dadas duas matrizes, A_{mxn} e B_{nxp}, um nó-mestre (*iniciador*) calcula $C_{mxp} = AB$ selecionando m nós-escravos (*colaboradores*) e

[3] Termo comumente empregado na literatura especializada para designar a criação de uma versão simplificada de um problema complexo, usado para demonstrar um conceito através da investigação ou teste de algoritmos relacionados a um problema real.

enviando, para cada escravo i ($1 \leq i \leq m$), uma cópia da matriz B com a matriz-linha A^i_{1xn}, cujos elementos correspondem à *i-ésima* linha de A. Cada escravo i calcula a matriz-linha $c^i_{1xp} = a^i_{1xn}B$ e a retorna para o nó-mestre. Ao receber o resultado, o nó-mestre alimenta, então, cada *i-ésima* linha de C com a matriz-linha c^i_{1xp} obtida. A seleção dos nós-escravos, na MANET, é feita através do protocolo P2PDP, levando-se em consideração os nós que possuem a maior quantidade disponível dos recursos CPU e memória. Esses recursos constituem o perfil de execução da aplicação M3. A ASL utilizada por essa aplicação foi desenvolvida para a família de aplicações de processamento intensivo. Para essas aplicações, variações na qualidade do enlace sem fio não influenciam a escolha dos colaboradores para a ASL – dado que a transferência de informações entre iniciadores e colaboradores é mínima e discreta. No caso específico da aplicação M3, por ser um *toy problem*, o tempo médio de execução de cada tarefa – considerando-se matrizes com tamanho máximo 20x20 – ficou em torno de 1 ms, motivo pelo qual o tempo de autonomia da bateria do dispositivo foi desconsiderado na seleção de colaboradores. Assim, na implementação da aplicação M3 os parâmetros da Equação (2) foram configurados como: $N = 2$, $P_{CPU} = P_{memória} = 1$ e $S = 100$ ms. O valor de S foi definido empiricamente, com base nos resultados obtidos nos testes realizados com a variação desse parâmetro.

Nesse ponto é importante ressaltar que as aplicações ASA e M3 foram desenvolvidas única e exclusivamente para avaliar a correção da implementação do protocolo P2PDP e do *middleware* MoGrid, auxiliando na realização de testes que resultaram em refinamentos sucessivos dessas implementações. As aplicações ASA e M3 não foram especificadas para atender as necessidades de cenários realistas ou visando uma utilidade mais ampla.

A terceira aplicação desenvolvida foi uma ferramenta que inicia, gerencia e pós-processa um conjunto de simulações paralelas do simulador *ns*-2. O pós-processamento trata os dados coletados, nas simulações, para obtenção dos resultados em função de critérios de análise pré-definidos. A ferramenta, denominada nsTOOL (*ns dispaTcher and pOst-prOcessing Launcher*), oferece uma interface gráfica e uma interface textual para facilitar a automatização do processo de submissão de múltiplos cenários de simulação para os dispositivos da

grade móvel aptos a colaborar. Isso inclui o *proxy* MoGrid, o que possibilita o envio dos *scripts* de simulação para as máquinas de uma grade fixa Globus qualquer. Como a aplicação nsTOOL pertence à mesma família da aplicação M3 – família de aplicações de processamento intensivo –, foi utilizada a mesma ASL. Entretanto, na aplicação nsTOOL – caracterizada como uma aplicação de longa duração –, o tempo de execução de cada tarefa é considerável, pois cada colaborador é responsável por um dado conjunto de simulações. Nesse caso, o tempo de autonomia da bateria do dispositivo é essencial na seleção dos colaboradores na grade móvel que irão gerenciar as rodadas de simulação e executar o pós-processamento de seus resultados. Na implementação da aplicação nsTOOL, os parâmetros da Equação (2) foram configurados como: $N = 3$, $P_{energia} = 3$, $P_{CPU} = 2$ e $P_{memória} = 1$. Nos testes realizados, foi utilizado o valor *default* de S (vide Seção 4.6), ou seja, $S = 10$ ms.

5.4.1. Ambientes de Teste Reais

As três aplicações descritas neste capítulo foram executadas inicialmente em três cenários de teste reais distintos, descritos a seguir.

Primeiramente, cada uma das três aplicações foi executada na rede do laboratório MARTIN[4] (*Mechanisms and ARchitectures for TeleINformatics*), constituída por cinco máquinas Linux e uma máquina Windows, todas interligadas por uma rede Ethernet. Para emular o funcionamento em uma rede sem fio, foi instalado em todas as máquinas um módulo de emulação do Monitor/Sim MoCA, o que permitiu reproduzir variações das informações de contexto referentes a qualidade do enlace sem fio, além de emular a variação dos níveis de energia disponíveis através de uma função linear decrescente. A qualidade do enlace sem fio foi medida em função da intensidade do sinal de rádio (*Radio Signal Strength Indications* – RSSI), configurado no intervalo [-20 dBm, -85 dBm], com os valores limites representando, respectivamente, o melhor e o pior nível de intensidade do sinal. As informações de contexto referentes a CPU,

[4] A página do projeto está disponível em <http://martin.lncc.br>. Acesso em: 10 fev. 2007.

memória e espaço em disco foram obtidas pelo módulo de execução real do monitor MoCA em cada máquina.

Posteriormente, a aplicação M3 foi executada em uma rede sem fio de testes implementada no projeto ARES[5] (*Architectures de RésEaux de Services*) do INRIA/INSA-Lyon. Essa rede é constituída por quatro *laptops* e quatro mini-PCs (*shuttles*) executando Linux, sendo configurada em modo *ad hoc*.

Por fim, as aplicações M3 e nsTOOL foram executadas novamente na rede do projeto MARTIN, dessa vez para verificar o funcionamento do *proxy* de colaboração. Para isso, uma das máquinas dessa rede foi registrada também como parte integrante do projeto InteGridade [Integridade 2003] que oferece uma grade baseada no Globus versão 2.4.

Em todos os cenários supracitados, os mecanismos de descoberta e seleção do protocolo P2PDP apresentaram o comportamento esperado, retornando como resultado o número de respostas solicitadas na requisição, com a seleção das respostas de melhor qualidade e a supressão das respostas desnecessárias. Pela observação do arquivo de *trace* do protocolo P2PDP, que registra o envio e recepção de requisições e respostas, incluindo a ocorrência de supressões, foi possível rastrear o encaminhamento dessas mensagens e verificar a distribuição da ocorrência de supressões entre os nós que compunham o caminho de retorno das mensagens de resposta. Através da comparação do perfil de execução de todos os dispositivos respondentes, foi possível observar que os colaboradores que tiveram as suas respostas selecionadas para cada requisição foram os que apresentaram a maior disponibilidade dos recursos indicados no perfil da requisição. Monitorando um bom número dessas execuções com o comando *top* – presente no sistema operacional Linux – e o gerenciador de tarefas (*taskmgr*) – presente no sistema operacional Windows –, foi possível observar a variação na disponibilidade de recursos dos dispositivos em função do processamento das requisições de descoberta e verificar a correção dos valores de retardo de envio gerados pelos colaboradores. Entretanto, é importante ressaltar que os experimentos realizados

[5] Os testes foram realizados em cooperação com os pesquisadores Guillaume Chelius e Nicolas Bolicaut. A página do projeto está disponível em <http://www.citi.insa-lyon.fr/team/ares>. Acesso em: 10 fev. 2007.

têm um caráter inicial, dada a simplicidade das aplicações implementadas, não sendo possível obter resultados conclusivos que apresentem uma validade mais geral sobre o desempenho do P2PDP, o que é feito no Capítulo 6.

5.4.2. Ambiente de Teste Simulado

É importante ressaltar que os experimentos realizados nos cenários descritos na seção anterior têm um caráter de prova de conceito somente, dada a simplicidade das aplicações implementadas e dos ambientes de teste disponíveis. Para possibilitar testes da implementação do *middleware* MoGrid e do protocolo P2PDP em cenários mais complexos, foram realizadas rodadas de execução simulada da aplicação M3 no simulador NCTUns [Wang & Lin 2005]. Esse simulador permite a configuração de vários dispositivos virtuais em uma mesma máquina. Cada um desses dispositivos virtuais pode executar uma instância real da implementação, conferindo assim um ambiente de simulação muito similar a de um ambiente real de testes. Nos testes realizados sobre o NCTUns, os dispositivos virtuais foram organizados como uma rede sem fio *ad hoc* de saltos múltiplos. Os testes sobre o simulador NCTUns foram executados em duas etapas. Na primeira etapa, os parâmetros relacionados à mobilidade foram desconsiderados. Em uma segunda etapa, a mobilidade dos dispositivos foi considerada.

Para ser possível a execução dos testes sobre o NCTUns, foi necessário adaptar o serviço monitor de modo a atender algumas características do ambiente de simulação. Como uma única máquina executando o NCTUns é usada para simular vários dispositivos em uma MANET, o uso do serviço monitor original incorreria em um cenário irreal, com todos os dispositivos na rede sem fio apresentando as mesmas informações de estado. Para solucionar esse problema, foi implementado um serviço monitor modificado que, a partir das informações de estado reais – obtidas através do serviço monitor –, retorna valores aleatórios diferenciados para cada nó que compõe o cenário sem fio simulado.

A Tabela 4 apresenta os valores dos parâmetros usados na constituição das rodadas de simulação do cenário que não contempla a mobilidade dos

dispositivos. Foi considerado para esse cenário uma densidade fixa de dispositivos na MANET, usando topologias em grade. A escolha dos dispositivos que irão atuar como colaboradores ativos é feita de forma aleatória a cada rodada pelo *script* NCTUns que gera a topologia da rede, considerando o valor de *p*.

Tabela 4 – Parâmetros de simulação NCTUns sem contemplar mobilidade.

Parâmetro	Valores
Número de dispositivos (*N*)	16
Densidade de dispositivos por raio de transmissão	4 dispositivos
Distância média entre dispositivos	240 m
Alcance da transmissão	250 m
Percentual de dispositivos colaboradores ativos (*p*)	20%
Número máximo de respostas (*R*)	2

A Figura 37 ilustra o comportamento das entidades P2PDP no processo de descoberta em uma rodada de simulação específica no NCTUns, para o cenário determinado na Tabela 4, com os dispositivos sem fio dispostos regularmente em grade.

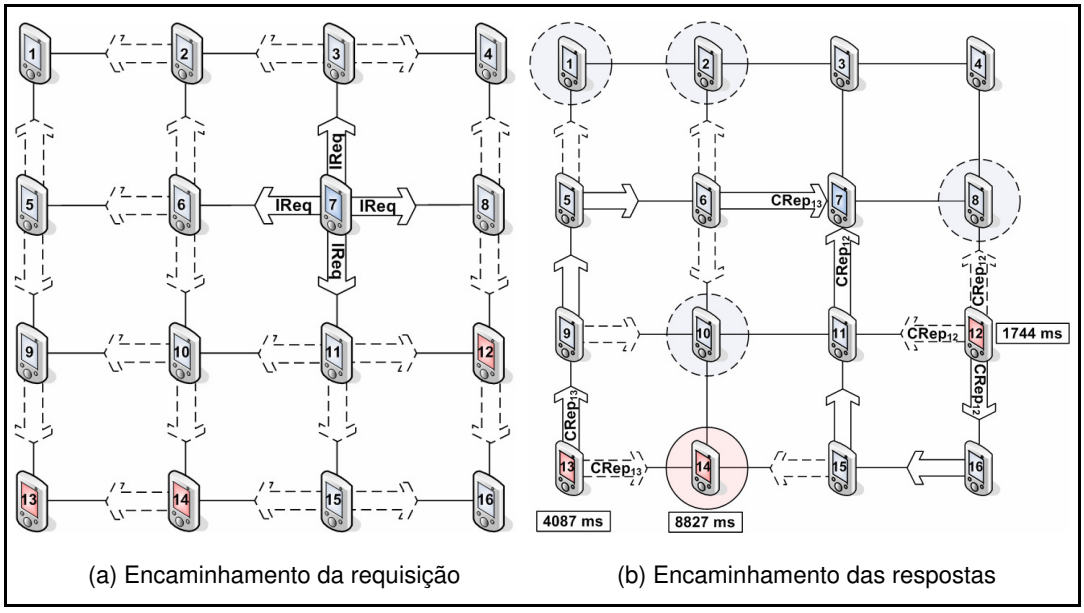

Figura 37 – Exemplo de rodada de simulação do protocolo P2PDP.

Na Figura 37, o nó 7 atua como o iniciador do processo de descoberta, os nós 12, 13 e 14 atuam no modo de colaboração ativa e os demais nós no modo passivo, apenas encaminhando as mensagens de controle do P2PDP que lhes são enviadas. Na Figura 37(a), o iniciador submete uma requisição de serviço (mensagem `IReq`) à rede, com o número máximo de respostas configurado para

dois (`numMaxReplies` = 2) e o tempo máximo de espera por respostas para dez segundos (`maxReplyDelay` = 10). Ao receber uma requisição, os colaboradores verificam duas condições: (*i*) se eles estão no modo de colaboração ativo (0 < `willingness` ≤ 1), ou seja, se eles podem colaborar, e (*ii*) se eles oferecem o serviço solicitado. Cada colaborador ativo calcula o retardo de envio da sua mensagem de resposta (`replyDelay`), de forma independente e distribuída, utilizando o algoritmo DR, e, a seguir, agenda o envio da sua resposta no instante de tempo obtido. Os valores obtidos para os temporizadores `replyDelay`, nos colaboradores 12, 13 e 14, foram, respectivamente, 1744 ms, 4087 ms e 8827 ms na rodada ilustrada na figura. As respostas são encaminhadas salto-a-salto em *broadcast* local direcionado, ou seja, todos os vizinhos diretos de um nó que tenha emitido uma resposta escutam a mensagem, mas apenas o nó que está no seu caminho de retorno a encaminha adiante na rede; os outros nós a descartam, conforme especificado no algoritmo SbV. Na Figura 37(b), as setas tracejadas representam as respostas encaminhadas aos nós que não se encontram no caminho de retorno dessas mensagens e os círculos tracejados representam o descarte das mensagens nesses nós. O caminho de retorno da resposta originada pelo nó 12 ($\texttt{CRep}_{12}$) é 12→16→15→11→7, o do nó 13 ($\texttt{CRep}_{13}$) é 13→9→5→6→7 e o do nó 14 ($\texttt{CRep}_{14}$) é 14→10→11→7. Ao escutar as mensagens de resposta dos nós 12 e 13, o nó 14 suprime a sua própria mensagem, pois a classifica como excedente, já que o valor do seu `replyDelay` é muito maior do que o dos demais; a supressão é representada na Figura 37(b) pelo círculo contornado pela linha sólida.

Nos cenários de teste realizados considerando-se a mobilidade dos dispositivos foram executadas rodadas de simulação utilizando os valores dos parâmetros definidos na Tabela 5. Foi considerado para esse cenário uma topologia determinada aleatoriamente a cada rodada pelo *script* NCTUns. A escolha dos dispositivos que irão atuar como colaboradores ativos é feita também de forma aleatória a cada rodada, considerando o valor de *p*.

Tabela 5 – Parâmetros de simulação NCTUns contemplando a mobilidade.

Parâmetro	Valores
Número de dispositivos (*N*)	10
Densidade média de dispositivos por raio de transmissão	3 dispositivos
Distância média entre dispositivos	240 m
Alcance da transmissão	250 m
Velocidade máxima de deslocamento	2 m/s
Pausa entre deslocamentos	10 s
Percentual de dispositivos colaboradores ativos (*p*)	60%
Número máximo de respostas (*R*)	2

A Figura 38 ilustra o comportamento das entidades P2PDP no processo de descoberta em uma rodada de simulação específica, de acordo com o cenário descrito pelos parâmetros da Tabela 5. O tempo máximo de espera por uma resposta (`maxReplyDelay`) é configurado para 10 s. Nesse cenário, o nó 4 atua como o iniciador do processo de descoberta, os nós 2, 5, 7, 8, 9 e 10 atuam no modo de colaboração ativa e os demais nós, no modo passivo. As setas, na figura, indicam o deslocamento dos nós durante o tempo de simulação, que foi de 53 segundos. Os círculos tracejados indicam a área que corresponde ao alcance de transmissão de cada nó no início da rodada de simulação; o círculo com a borda contínua indica a área que corresponde ao alcance de transmissão do iniciador (nó 4) após completar o seu deslocamento.

Pela observação da Figura 38, no início da colaboração, desconsiderando-se a adequação dos dispositivos, os colaboradores mais próximos do iniciador são os nós 5 e 10. Conforme os dispositivos vão se movimentando, os colaboradores 2, 5, 9 e 10 vão se aproximando do iniciador, permanecendo no seu alcance de transmissão ao final do deslocamento. Pela análise do arquivo de *trace* da simulação, é possível observar que os colaboradores 7 e 8 foram os únicos a não responder à requisição do iniciador, por se encontrarem fora do alcance de transmissão dos demais nós, durante a difusão da mensagem. Os nós 2, 5, 9 e 10 agendam o envio de suas respostas, conforme especificado no algoritmo SbV, em função do temporizador `replyDelay`, calculado de forma distribuída e independente, em cada colaborador, através do algoritmo DR. Os valores obtidos para os temporizadores `replyDelay` nos colaboradores 2, 5, 9 e 10 foram, respectivamente, 1667 ms, 3829 ms, 2884 ms e 6830 ms. Por fim, o iniciador recebe as respostas dos colaboradores 2 e 9, com as demais respostas sendo

suprimidas ao longo do seu caminho de retorno. A resposta do colaborador 5 é descartada pelo iniciador, pois ele já havia recebido o número de respostas que havia solicitado ($R = 2$). A resposta do colaborador 10 é suprimida por ele mesmo ao escutar as respostas dos colaboradores 2 e 9. Como resultado das simulações executadas, foi possível verificar a correção dos mecanismos de descoberta e seleção de recursos implementados no protocolo P2PDP, considerando-se o deslocamento dos nós. É importante ressaltar que a escolha do cenário reproduzido na Figura 38 foi simplesmente o de ilustrar o comportamento do protocolo P2PDP em uma rede sem fio *ad hoc* de saltos múltiplos, mediante a mobilidade dos nós. O número de dispositivos foi escolhido por uma questão meramente didática, com o intuito de facilitar a explanação sobre o funcionamento do protocolo P2PDP. Cenários de maior escala são retratados no Capítulo 6, onde é feita a avaliação de desempenho do protocolo P2PDP.

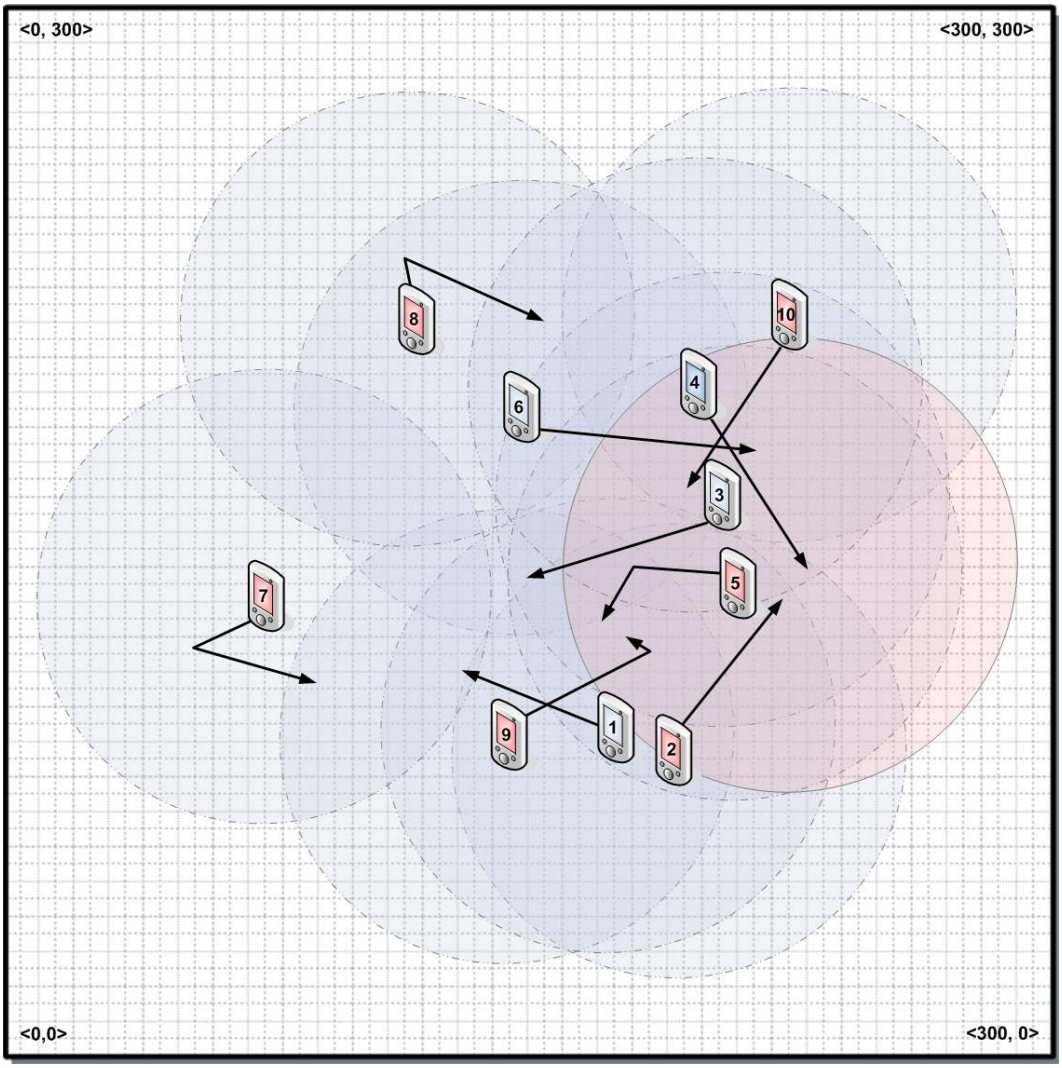

Figura 38 – Rodada de simulação NCTUns com mobilidade.