Controle de Concorrência Reconfigurável para o Ambiente Móvel

A. G. de A. L. Pierre, M. B. F. de Toledo, T. da Rocha

***Resumo*— Dentre as tecnologias emergentes, a computação móvel tem a sua posição de destaque. Apesar de atrativa, existe a limitação de recursos e nesse contexto, as transações representam um papel importante. Diversos modelos de transações têm sido apresentados na literatura, mas apesar de muitas idéias interessantes, alguns modelos não permitem que a adaptação seja realizada durante a execução de uma transação e quando permitem, eles realizam grandes reconfigurações arquiteturais. Motivado por essas questões, esse artigo propõe a configuração do mecanismo de controle de concorrência, no início da transação, e a reconfiguração da propriedade transacional, nível de isolamento, durante a execução da transação. O modelo de componentes OpenCOM foi utilizado para o desenvolvimento de um protótipo de um sistema de vendas. A configuração do controle de concorrência é uma contribuição inovadora desse artigo, pois em muitos trabalhos existentes não é possível a configuração desse mecanismo transacional.**

***Palavras chave*— Computação móvel (*Mobile computing*), sistemas de computação adaptativos *(adaptive computing systems*), reconfiguração dinâmica (*dynamic reconfiguration*), OpenCOM, controle de concorrência (*concurrency control*), nível de isolamento (*isolation level*).**

I. INTRODUÇÃO

Os dispositivos móveis estão cada vez mais presentes na vida das pessoas e contendo aplicações mais sofisticadas e semelhantes às executadas em computadores pessoais.

Um dos exemplos mais recentes é o surgimento da plataforma aberta para dispositivos móveis, chamada Android [1], criada pela empresa Google que promete revolucionar o desenvolvimento de aplicações nesse ambiente. Outro exemplo é o uso crescente da internet em celulares, permitindo que os usuários acessem diversos tipos de aplicações, tendo grande parte delas interação com banco de dados.

A computação móvel traz desafios ao desenvolvedor [2], pois ele deve considerar os recursos limitados tais como largura de banda, conectividade e o alto custo da obtenção de dados.

Nesse contexto, as transações representam um importante papel de garantir que o dinamismo do ambiente da computação móvel não comprometa a confiabilidade das aplicações. Porém, algumas aplicações não podem ser implementadas considerando o modelo de transações tradicional, pois ele não foi projetado para lidar com obstáculos de ambientes móveis – desconexões inesperadas, atrasos na comunicação e mobilidade.

Diante das limitações do modelo tradicional, diversos modelos de transações têm sido propostos na literatura. Alguns exemplos são: MobileTrans [3], CATE [4] , AMT [5], ReflecTS [6] e SGTA [7]. Esses modelos permitem que transações sejam personalizadas de acordo com os requisitos das aplicações e com as características específicas do ambiente móvel em questão. Porém, eles não permitem que essa adaptação seja realizada durante a execução da transação – característica essencial para lidar com aspectos dinâmicos do ambiente móvel.

Nesse contexto, esse artigo propõe uma arquitetura adaptável para o mecanismo de controle de concorrência de transações voltadas ao ambiente móvel. Esta arquitetura permite tanto configuração estática – para atender requisitos das aplicações – quanto dinâmica – para se adaptar às variações dinâmicas nos recursos do ambiente. O nível de isolamento é a propriedade transacional que poderá ser reconfigurada durante a transação e o controle de concorrência é o mecanismo que garantirá o isolamento entre as transações e poderá ser configurado no início da transação.

Um modelo de componentes denominado OpenCOM [8] foi o escolhido para a implementação da arquitetura proposta. Como esse modelo utiliza a técnica de reflexão computacional [9], ele provê uma facilidade de adaptação da arquitetura dos componentes diante de certas mudanças do ambiente.

A fim de verificar a arquitetura proposta, um protótipo de um sistema de vendas foi desenvolvido.

O artigo está organizado conforme descrito: A seção II apresenta a arquitetura proposta de controle de concorrência reconfigurável para dispositivos móveis. A relação entre a computação reconfigurável e a tecnologia adaptativa é mostrada na seção III. A implementação do protótipo e dos estudos de casos são apresentados na seção IV e a seção V termina o artigo apresentando as conclusões.

II. ARQUITETURA PROPOSTA

Considerando que aplicações que utilizam serviços de transações podem possuir requisitos distintos, este artigo apresenta uma nova arquitetura capaz de ser configurada com um controle de concorrência pessimista ou com um controle de concorrência otimista.

Além de permitir a configuração do tipo de controle de concorrência desejado, esta arquitetura foi projetada para prover mecanismos de adaptação dinâmicos para transações. Neste sentido, esta arquitetura permite que uma transação

Esse artigo é parte da dissertação de defesa de Mestrado da aluna A. G. de A. L. Pierre orientada pela professora M. B. F. de Toledo, do Instituto de Computação da UNICAMP e com contribuição do professor T. da Rocha.

mude o seu nível de isolamento dinamicamente como reação às mudanças ocorridas em tempo de execução.

Desta forma, optou-se por permitir que a troca de controle de concorrência ocorra somente de forma estática, ou seja, antes do processo de execução de transações. A possibilidade de reconfiguração dinâmica foi destinada somente ao âmbito das propriedades transacionais como, por exemplo, os níveis de isolamento que causam menos mudanças estruturais.

A arquitetura de serviços de transações distribuídos normalmente é estruturado em duas partes: (i) uma parte cliente - localizado na máquina onde a aplicação cliente executa; (ii) uma parte servidor – localizada em cada ponto da rede onde a transação cliente acessa dados transacionais remotos.

A Fig. 1 mostra a arquitetura do lado cliente proposta neste artigo enquanto a Fig. 2 mostra a arquitetura do lado servidor.

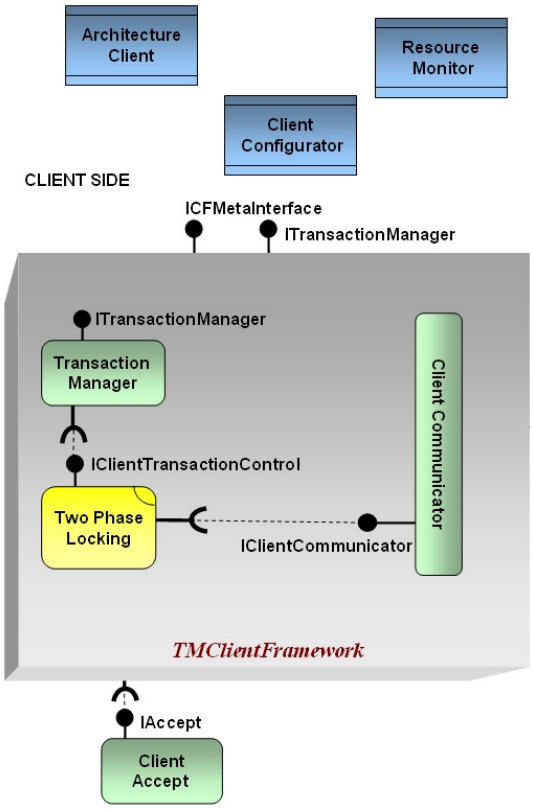

Fig. 1. Arquitetura proposta – lado Cliente

As convenções utilizadas para representar o desenho arquitetural são descritas a seguir:

 representa um framework de componentes [10] que é definido no modelo OpenCOM como uma sub-arquitetura de componentes sobre a qual pode ser verificado um conjunto de propriedades arquiteturais desejadas. Essa sub-arquitetura também pode ser vista como um componente complexo que é composto por outros componentes interconectados.

A função do framework de componentes é impor restrições ao processo de configuração e reconfiguração da plataforma de forma que a integridade da mesma possa ser mantida.

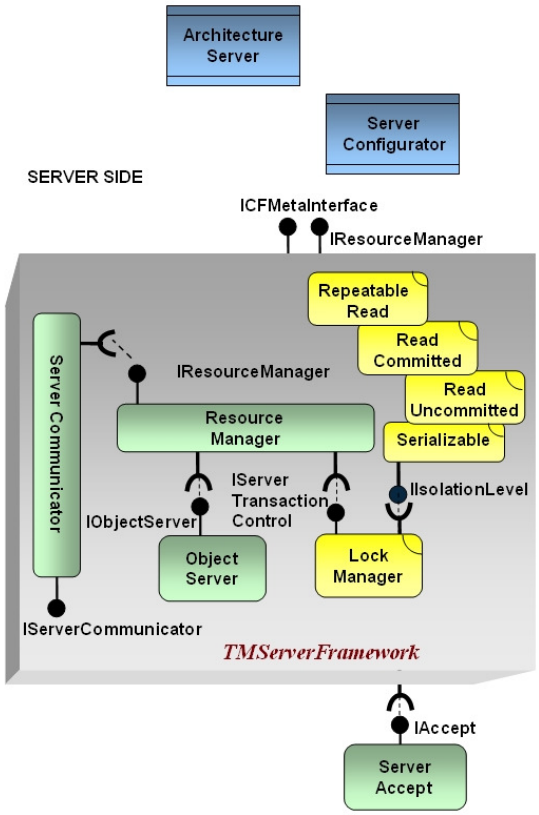

Fig. 2. Arquitetura proposta – lado Servidor

 representa uma interface: serviços providos pelo componente. Uma interface estabelece uma espécie de contrato que é obedecido por uma classe. Ela só expõe o que uma classe que a implementa deve fazer. Ela não possui atributos, nem métodos com implementação. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface.

 representa um receptáculo: serviços requeridos por um componente. Eles podem ser simples ou múltiplos. Um receptáculo simples só aceita a conexão de uma interface por vez. Já o receptáculo múltiplo permite a conexão de mais de uma interface ao mesmo tempo.

 representa uma classe para um conjunto de objetos com características comuns. Ela possui atributos e métodos. É necessário instanciá-la para criar os objetos.

 representa um componente OpenCOM. Estes são componentes fixos da arquitetura, ou seja, não podem ser mudados no processo de reconfiguração.

 também representa um componente OpenCOM.

Esses componentes estão destacados na arquitetura porque podem ser substituídos dinamicamente conforme o tipo de controle de concorrência e nível de isolamento escolhidos.

representa a conexão entre uma interface e um receptáculo. Para um que componente Y utilize serviços de outro componente Z, um receptáculo de Y precisa ser conectado a uma interface de Z.

Na arquitetura proposta foram considerados os controles de concorrência pessimista e otimista [11], [12]. Basicamente, um controle de concorrência é chamado pessimista quando ele evita previamente que ocorram conflitos entre transações em execução. Para isso ele normalmente impõe restrições de acesso aos dados às transações. Já um controle otimista, não impõe restrições de acesso aos dados durante a execução das transações, pois os conflitos só são detectados e resolvidos no final da execução das transações concorrentes. Controles pessimistas normalmente são baseados em trancas, enquanto que os otimistas são baseados em controle de versões sobre os dados acessados. Os componentes *TwoPhaseLocking*, *LockManager*, *ReadUncommitted*, *ReadCommitted*, *Repeatable Read* e *Serializable* fazem parte do controle de concorrência pessimista, conforme mostrado anteriormente nas Fig. 1 e Fig. 2.

No controle de concorrência otimista, no lado Cliente, o componente *TwoPhaseLocking* é substituído pelo componente *VersionControl*. Já no lado servidor, o componente *LockManager* é substituído pelo componente *VersionManager*. Os componentes que implementam os níveis de isolamento *Read Uncommitted*, *ReadCommitted*, *Repeatable Read* e *Serializable* foram incluídos somente na arquitetura para o controle de concorrência pessimista. Os componentes que são alterados para o controle otimista estão destacados nas Fig. 3 e Fig. 4.

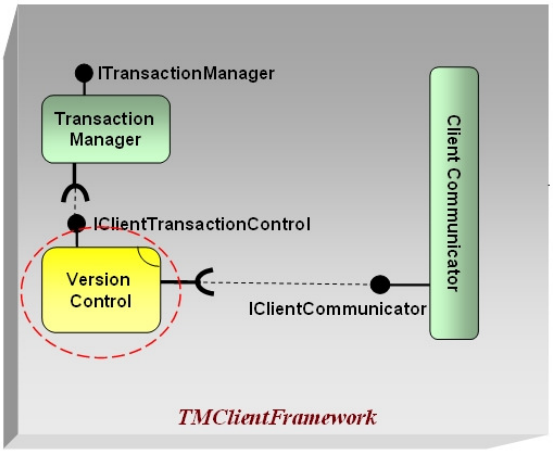

Fig. 3. Controle otimista – lado Cliente

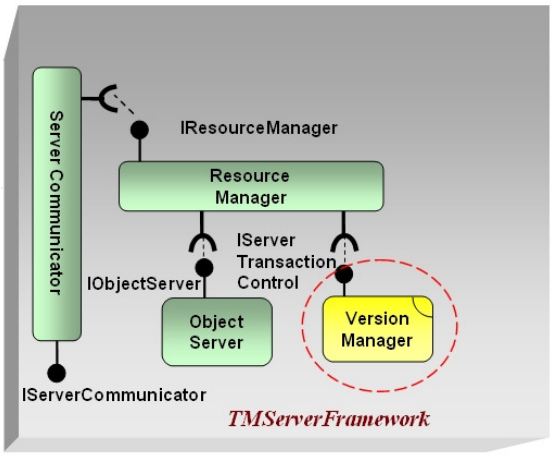

Fig. 4. Controle otimista – lado Servidor

A seguir, cada elemento da arquitetura será detalhado dentro do contexto arquitetural (eles estão listados em ordem alfabética):

- ***ArchitectureClient*** e ***ArchitectureServer***: montam a arquitetura OpenCOM (definindo os controles de concorrência e níveis de isolamento) respectivamente do lado cliente e servidor, fazendo a ligação entre os componentes.

- ***ClientAccept*** e ***ServerAccept***: componentes que implementam as regras capazes de impor as propriedades arquiteturais desejadas para a arquitetura interna dos frameworks *TMClientFramework* e *TMServerFramework*. Para evitar que estas configurações e reconfigurações comprometam a integridade da plataforma, cada mudança realizada é comparada a um conjunto de arquiteturas válidas pré-definidas. As operações de inspeção dos frameworks incluem: a recuperação de componentes internos, a recuperação da lista de conexões de um dado componente e a recuperação da lista de interfaces e da lista de receptáculos de componentes internos.

- ***ClientCommunicator***: componente responsável pela comunicação com o lado servidor através de mensagens via RMI (*Remote Method Invocation*)[13]. Implementa a interface *IClientCommunicator*.

- ***ClientConfigurator***: classe que implementa a política de adaptação do lado cliente, como por exemplo, o que acontece com o controle de concorrência e com o nível de isolamento se um determinado recurso do dispositivo móvel está limitado. Ela se comunica com a classe *ServerConfigurator* através da tecnologia RMI para notificar que uma mudança ocorreu na arquitetura do lado cliente e a mesma deve ser realizada no lado servidor.

- ***ServerCommunicator***: responsável pela comunicação com o lado cliente, através de mensagens via RMI. Implementa a interface *IServerCommunicator*.

- ***ServerConfigurator***: classe que define o controle de concorrência e o nível de isolamento do lado servidor especificados no início da aplicação ou após a notificação da classe *ClientConfigurator*.

- ***TMClientFramework***: framework de componentes OpenCOM do lado cliente.

- ***TMServerFramework*:** framework de componentes OpenCOM do lado servidor.

Os componentes e as classes que implementam o gerenciamento de transações são descritos a seguir:

- ***ObjectServer***: componente que representa o repositório de objetos acessados pelas transações.

- ***ResourceManager***: componente que gerencia o recurso (*ObjectServer*) executando as operações necessárias (inclusão, atualização e consulta).

- ***ResourceMonitor***: classe responsável por monitorar os recursos do ambiente do dispositivo móvel que possam sofrer variações como por exemplo, bateria e sinal da conexão. Ela também é responsável por notificar as transações das variações do ambiente quando estas solicitam tal monitoramento.

- ***TransactionManager***: componente que gerencia a transação do lado cliente.

Os componentes relacionados com o controle de concorrência pessimista são descritos a seguir:

- ***LockManager***: componente pertencente ao controle de concorrência pessimista do lado servidor. Ele mantém uma lista de objetos bloqueados (uma cópia desses objetos) durante uma transação.

- ***ReadCommitted***: componente que representa o nível de isolamento *Read Committed*.

- ***ReadUncommitted***: componente que representa o tipo de isolamento *Read Uncommitted*. É o nível de isolamento menos restrito.

- ***Repeatable Read***: componente que representa o tipo de isolamento *RepeatableRead*.

- ***Serializable***: componente que representa um tipo de isolamento *Serializable*. É o nível de isolamento mais restrito.

- ***TwoPhaseLocking***: componente pertencente ao controle de concorrência pessimista do lado cliente. Implementa o protocolo de bloqueio em duas fases [11].

Os componentes relacionados com o controle de concorrência otimista são descritos a seguir:

- ***VersionControl***: componente que faz parte do controle de concorrência otimista do lado cliente e é responsável por implementar as operações (inclusão, atualização e consulta) sobre os objetos da transação. As alterações são armazenadas em *cache* (memória local de acesso rápido) antes de serem efetivadas.

- ***VersionManager***: componente que faz parte do controle de concorrência otimista do lado servidor. Ele controla a versão dos objetos utilizados durante uma transação implementando as operações de validação referentes ao controle de concorrência.

A. Reconfiguração de componentes

A reconfiguração da transação é feita através da troca de componentes: componentes podem ser conectados ou desconectados de acordo com a configuração desejada, não necessitando remontar toda a arquitetura definida.

Como o foco do modelo proposto é o controle de concorrência, a idéia é que o modelo permita a configuração estática do controle de concorrência e a reconfiguração dinâmica do nível de isolamento baseado em políticas de adaptação pré-definidas.

O controle de concorrência é configurado no início da transação conforme a situação dos recursos disponíveis no ambiente móvel, como, por exemplo, bateria, largura de banda ou conectividade.

O mecanismo de controle de concorrência padrão é o pessimista com nível de isolamento *Serializable*. Se a situação de um recurso for alterada extrapolando alguns limites pré-estabelecidos pela transação, algumas propriedades podem mudar: se a transação estiver no início, o controle de concorrência poderá ser alterado, mas se ela estiver em execução, apenas o nível de isolamento poderá ser alterado.

Nesse contexto, a transação pode reagir às mudanças do ambiente conforme a política de adaptação definida para ela. A lógica dessa política é definida pelo programador. Ela é escrita utilizando uma linguagem de programação e contém as regras que deverão ser seguidas caso o monitor de recursos verifique que os valores pré-estabelecidos pela aplicação foram extrapolados.

A reconfiguração é feita pelas classes *ArchitectureServer* e *ArchitectureClient*. O monitor de recursos verifica os recursos disponíveis no ambiente do dispositivo móvel cujo monitoramento foi requisitado pela aplicação. Se o limite for atingido, ele notifica a classe *ClientConfigurator* que contém a política de adaptação do lado cliente.

Se uma das regras definidas na política de adaptação for satisfeita, a arquitetura deverá ser alterada, portanto, a classe *ArchitectureClient* será chamada para realizar a adaptação dinâmica. A reconfiguração é realizada desconectando e conectando componentes.

Como o cliente e o servidor devem estar sincronizados, ou seja, com o mesmo tipo de controle de concorrência, o cliente notifica a classe *ServerConfigurator*

indicando que a sua arquitetura também deve ser alterada já que a arquitetura do cliente foi mudada. O *ServerConfigurator* contém a política de adaptação do lado servidor e caso alguma regra inserida nele seja cumprida, a classe *ArchitectureServer* é chamada para também fazer a reconfiguração do lado servidor.

A arquitetura que foi apresentada na Fig. 2 contém quatro níveis de isolamento, porém novos níveis de isolamento poderiam ser adicionados.

III. TECNOLOGIA ADAPTATIVA E COMPUTAÇÃO RECONFIGURÁVEL

Segundo [14], a "tecnologia adaptativa" é o conjunto das aplicações práticas do conceito fundamental da adaptabilidade, sempre que esta for utilizada como instrumento na resolução de problemas oriundos das mais variadas. E a adaptabilidade é a propriedade que apresenta um sistema, dispositivo ou processo computacional, que lhe permite sem a interferência de agentes externos – mesmo do próprio operador – tomar a decisão de modificar dinamicamente, de forma autônoma, seu próprio comportamento, em resposta apenas à sua configuração corrente e ao estímulo de entrada recebido.

A adaptabilidade é uma das idéias que viabilizam a computação móvel, pois mecanismos de adaptação devem ser projetados para que os recursos disponíveis sejam melhor utilizados.

Relacionado à adaptabilidade está o conceito de reflexão computacional [9] que implica em inspeção e adaptação. O sistema inspeciona o estado corrente e a adaptação faz com que o comportamento do mesmo seja alterado para melhor se ajustar ao ambiente.

A principal vantagem da reflexão computacional é permitir que o sistema seja alterado em tempo de execução, sendo essas alterações comportamentais ou até mesmo estruturais. Na computação móvel essa técnica é essencial para prover inspeção e adaptação do sistema diante da variação dos recursos disponíveis.

Na arquitetura apresentada, a adaptabilidade do controle de concorrência e do nível de isolamento foi implementada através do modelo de componentes OpenCOM, que implementa a reflexão computacional, e de regras pré-definidas pelo programador nos componentes *ClientConfigurator* e *ServerConfigurator*, permitindo que o sistema se modificasse dinamicamente sem a interferência do usuário, adaptando-se às mudanças do ambiente.

IV. PROTÓTIPO E ESTUDOS DE CASOS

A. Protótipo

Um protótipo foi desenvolvido para comprovar a viabilidade da arquitetura proposta. Ele foi desenvolvido utilizando a versão 1.3.5 (Java) do OpenCOM [15] e a linguagem Java [16]. O motivo da escolha da linguagem Java é por ela ser orientada a objetos e independente de plataforma.

O protótipo evidencia a configuração dos controles de concorrência e a reconfiguração dos níveis de isolamento. Nele foram implementados, dois níveis de isolamento: *Serializable* (mais restrito) e *Read Uncommitted* (menos restrito).

Para avaliar a solução proposta, um estudo de caso na área de vendas foi escolhido.

B. Estudo de casos: Vendas

Considere uma aplicação de vendas destinada aos gerentes e representantes de vendas de uma distribuidora de medicamentos. Ambos utilizam PDA's (*Personal Digital Assistant*) para acessarem os módulos da aplicação e realizarem as tarefas de sua função. Um PDA é conhecido como um computador de bolso, por ter dimensões reduzidas.

A aplicação possui dois módulos: Pedido de venda de produtos e Relatório da média das vendas do dia.

O representante de vendas efetua o pedido de venda do produto através de um PDA. Ele comunica-se diretamente com o servidor em tempo real, fazendo seus pedidos através do computador, ganhando dessa forma, agilidade. O pedido do produto, após a venda efetivada, é enviado para o serviço de entrega, que já inicia o seu trabalho.

O gerente consulta a média das vendas realizadas no dia também através de um PDA. Para tomar alguma decisão com maior rapidez, ele precisa ser informado sobre as vendas realizadas no dia e o desempenho de sua equipe de vendedores.

Os dois módulos se conectam a um servidor para efetuar as vendas dos produtos e realizar as consultas necessárias.

Existem vários representantes de vendas em lugares diferentes que utilizarão a aplicação ao mesmo tempo.

Como esses usuários estão utilizando dispositivos móveis, eles encontram restrições de alguns recursos como, por exemplo, bateria e conectividade. Sendo assim, em situações de escassez de energia ou nível de bateria limitado, o gerente necessita visualizar a média das vendas considerando até os produtos que já foram registrados pelos representantes, mas cuja venda ainda não foi concluída. Essa informação o ajudará a tomar decisões como criar promoções e agilizar a entrega de alguns produtos, aumentando a margem de lucro e a satisfação do cliente.

Para esse tipo de aplicação em que existe concorrência, é necessário haver um gerenciamento de transações.

O protótipo dessa aplicação de Vendas utilizará a arquitetura mencionada anteriormente. Três casos de uso serão demonstrados para explicar o funcionamento da arquitetura, a configuração dos controles de concorrência no início da transação e a reconfiguração dos níveis de isolamento durante a transação.

1) Caso de uso: tipo de controle pessimista

Para um melhor entendimento dos casos de uso, será mostrada, primeiramente, a aplicação e posteriormente, algumas informações arquiteturais do protótipo implementado.

A aplicação de vendas contendo os dois módulos (Pedido de venda de produtos e Relatório de média das vendas do dia) representa o lado Cliente na arquitetura proposta. O lado

servidor contém o banco de dados com as informações do estoque de produtos e das vendas realizadas.

O servidor está pronto aguardando a conexão de algum cliente. O tipo de controle de concorrência padrão é o pessimista.

Um representante de vendas chamado José inicia o seu dia de trabalho. Ele está nas ruas, utilizando seu PDA.

José visita uma farmácia, verifica os medicamentos que necessitam ser solicitados à distribuidora e começa a registrar a solicitação na aplicação de vendas em seu PDA. Ele seleciona a opção "*Order*" (representada pelo ícone do carrinho) e escolhe o produto "p1", quantidade igual a 2 e seleciona a opção "*Confirm*", conforme a Fig. 5.

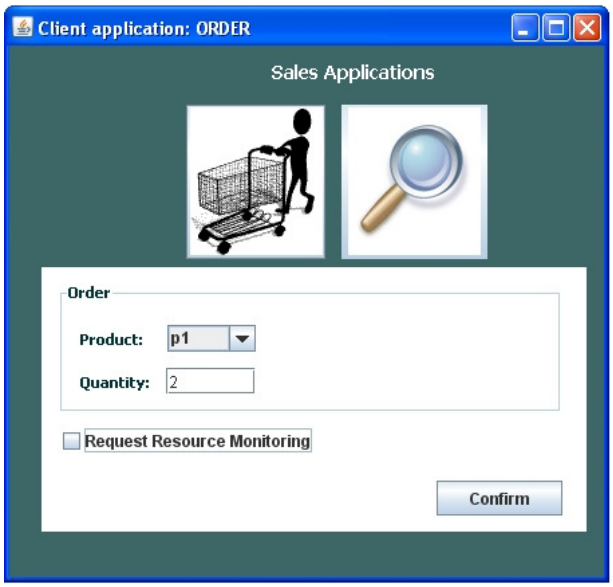

Fig. 5. Pedido solicitado pelo representante de vendas José

A transação inicia executando as operações de inserir o produto na tabela de vendas e atualizar o estoque. Antes de a transação do representante de vendas José ser efetivada, outro representante de vendas chamado Manuel estava executando, ao mesmo tempo, a aplicação em outra farmácia. Ele solicitou o mesmo produto "p1" com quantidade igual a 3. Manuel não consegue realizar a compra do produto porque a transação do representante de vendas José está bloqueando os respectivos objetos da tabela de vendas. Como o controle de concorrência é pessimista, a transação do representante de vendas Manuel segue apenas até a primeira operação de inserir o produto na tabela de vendas.

O representante de vendas José decide cancelar o pedido do produto "p1" porque a farmacêutica descobriu que havia ainda dois medicamentos "p1" no estoque, porém eles estavam armazenados no lugar incorreto. José seleciona a opção "*Cancel*" e com isso as operações de inserção do produto na tabela de vendas e a atualização da quantidade na tabela de estoque são desfeitas.

Portanto, nenhum produto foi vendido. O representante de vendas Manuel poderia ter efetivado a sua compra, mas devido ao tipo de controle de concorrência ser pessimista, ele perdeu a venda.

Diante disso, nunca deveria ser utilizado o controle de concorrência pessimista? A resposta seria: depende. O controle de concorrência pessimista é necessário quando existe a grande probabilidade de conflitos e no ambiente de computação móvel, quando os recursos como conexão e bateria não estão limitados. Mas como saber se os recursos estão limitados? Para isso, a arquitetura contém o componente *ResourceMonitor*. Na Fig. 5, José poderia ter optado por monitorar os recursos do dispositivo móvel, como por exemplo, o sinal da conexão com o servidor.

Como os representantes usam dispositivos móveis, o sinal de conexão poderia estar ruim e a conexão com o servidor cair. Sendo assim, os objetos ficariam bloqueados até a conexão voltar, impedindo novas vendas de um determinado produto. Diante desse contexto, seria apropriado alterar o tipo de controle de concorrência para otimista conforme o próximo caso de uso.

2) *Caso de Uso: alteração para o tipo de controle otimista*

Suponha que o representante de vendas José tenha optado por monitorar os recursos do dispositivo móvel no primeiro caso de uso. Conforme a Fig. 6, ele pode solicitar ao monitor de recursos que verifique o nível do sinal de conexão ou da bateria. Nesse caso, ele solicitou monitorar o nível do sinal de conexão selecionando a opção "*Request Resource Monitoring*" e "*Signal Strength*". O limite mínimo deve ser escolhido também. Entre as opções, existem: "*No Connectivity*", "*Limited*", "*Good*", "*Very Good*" e "*Excellent*", ou seja, opções indicando desde nenhum sinal até um nível de conexão excelente. Esse limite indica qual o nível mínimo do recurso monitorado será suportado pelo dispositivo móvel.

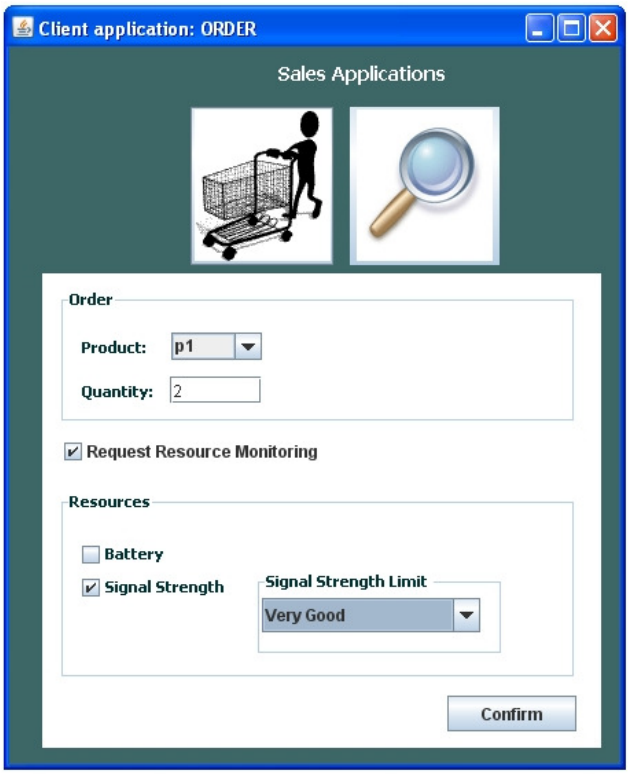

Fig. 6. Solicitação de monitoramento de recursos do dispositivo móvel

Nesse exemplo, a opção escolhida foi "*Very Good*". Isto significa que, se o nível do sinal de conexão do dispositivo estiver abaixo do limite "muito bom", o monitor de recursos notificará a transação que o limite está baixo do solicitado e a política de adaptação definida será aplicada.

Após selecionar a opção "*Confirm*", antes de iniciar a transação, o sistema verificará que a opção de monitoramento de recursos foi selecionada. O componente *ResourceMonitor* é chamado para monitorar o nível de sinal de conexão.

A transação será iniciada no dispositivo móvel de José. Como no caso de uso anterior, o controle de concorrência seria o pessimista. Porém, como ele solicitou o monitoramento do sinal de conexão, o monitor de recursos constantemente verifica se o sinal está dentro do limite pré-estabelecido. A conexão com o servidor estava excelente, porém de repente o sinal ficou limitado, antes do início da transação.

O outro representante de vendas Manuel, ao mesmo tempo, executa a aplicação cliente também solicitando o monitoramento de recursos do seu dispositivo móvel. O recurso escolhido também é o sinal de conexão e o limite tem que ser no mínimo "muito bom". O dispositivo móvel de Manuel está com uma conexão excelente com o servidor.

Antes da execução da operação de inserir o produto na tabela de vendas, a conexão com o servidor é alterada e se torna limitada.

Neste momento, o monitor de recursos percebe que o um dos recursos foi alterado. Ele compara o nível do sinal medido com o limite estabelecido pelo cliente da aplicação. Como o limite do recurso monitorado é menor que o limite esperado, o monitor de recursos notifica a transação. A transação então avalia as regras da política de adaptação para tomar alguma decisão diante da mudança.

Uma das regras definidas na política de adaptação é que, se um dos recursos monitorado estiver abaixo do limite estabelecido pelo cliente da aplicação antes do início da transação, o controle de concorrência deve ser alterado de pessimista para otimista.

No controle de concorrência otimista, os objetos não são bloqueados. Os objetos da transação são copiados para a máquina local do cliente (mantidos em *cache*) e todas as operações são efetuadas sobre a cópia local. Portanto, se a conexão com o servidor cair, a transação pode continuar.

O dispositivo móvel do representante de vendas Manuel estava com a conexão limitada. Prevendo uma possível desconexão, os objetos da transação foram copiados e as operações de inserção na tabela de vendas e de atualização na tabela de estoque foram executadas localmente no dispositivo móvel. No final da transação, as alterações são validadas para verificar se a versão de cada objeto foi alterada. As versões dos objetos de vendas e estoque permaneceram a mesma, portanto a venda do produto foi efetivada e o estoque atualizado.

O controle de concorrência do dispositivo móvel de José também foi alterado para otimista já que o nível do sinal estava limitado e ele havia estabelecido um limite de nível de conexão "muito bom". Ele conseguiu executar localmente as operações sobre os objetos. Porém, quando ele foi efetivar a transação do lado servidor, como o representante de vendas Manuel já havia alterado os objetos de vendas e estoque, as versões estavam diferentes (a versão dos objetos do repositório com a versão desses objetos do cache do lado cliente). Dessa forma, José não conseguiu efetivar o seu pedido.

No caso de uso acima, os dois clientes iniciaram a transação utilizando o controle de concorrência otimista, devido ao sinal de conexão limitado. Supondo que a transação do dispositivo de José inicie utilizando o controle pessimista (o sinal de conexão não está limitado) e que antes de a transação ser efetivada, Manuel também solicita o mesmo produto. O nível de conexão do dispositivo móvel dele está limitado e, portanto, o controle é alterado para otimista do lado cliente. Quando o servidor receber uma solicitação para alterar o controle de concorrência, ele deve verificar se existe alguma transação iniciada com algum cliente em outro modo de concorrência. Se existir, ele mantém o modo de concorrência e recusa a nova solicitação. Nesse caso, José conseguiria efetivar a transação, Manuel não conseguiria mudar o modo de controle de concorrência e teria que prosseguir no modo pessimista ou abortar a transação. O motivo disso é evitar que cliente e servidor tenham tipos de controles de concorrência diferentes.

3) Caso de Uso: reconfiguração do nível de isolamento

Os casos de uso anteriores demonstraram a configuração do controle de concorrência antes do início da transação. Esse caso de uso demonstrará a reconfiguração do nível de isolamento durante uma transação.

Suponha que os seguintes pedidos já tenham sido efetivados:

Produto = "p2" com quantidade = 1
Produto = "p2" com quantidade = 2
Produto = "p2" com quantidade = 3

Porém, um outro pedido é solicitado, mas não é efetivado (as operações de inserir o produto na tabela de vendas e atualizar o estoque são executadas, mas os dados não são gravados no banco de dados). Esse pedido é descrito abaixo:

Produto = "p1" com quantidade = 50

Ao mesmo tempo, o gerente da equipe de vendas, para tomar uma decisão, necessita consultar a média das vendas realizadas no dia. Como ele está fora do escritório, ele utiliza o seu PDA para obter esse resultado. Ele seleciona a opção "*Sales report*" e solicita o monitoramento de recursos do dispositivo móvel. O recurso a ser monitorado será a bateria. A aplicação é exibida na Fig. 7.

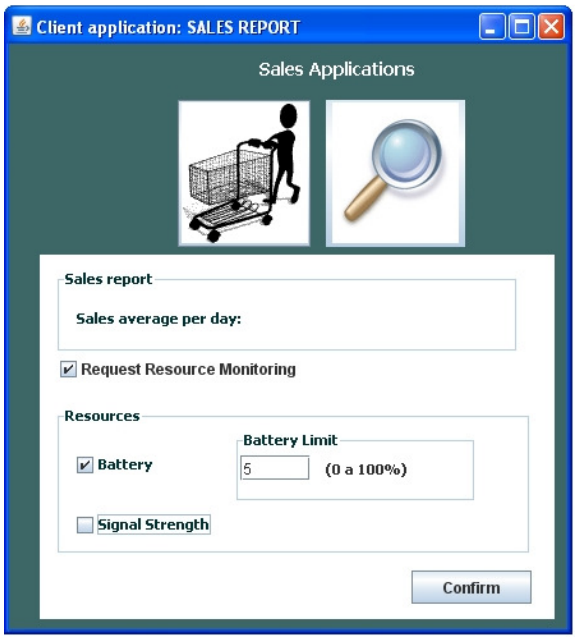

Fig.7. Opção selecionada: relatório das vendas do dia

Supondo que o nível da bateria do dispositivo móvel esteja 100%, a aplicação exibirá apenas a média das vendas efetivadas. Portanto, no exemplo acima, a média é igual a 10 considerando-se que:

Produto = "p2" com quantidade = 1 (1 x 10.00) = 10.00
Produto = "p2" com quantidade = 2 (2 x 10.00) = 20.00
Produto = "p2" com quantidade = 3 (3 x 10.00) = 30.00

No cálculo acima, o produto "p1", com quantidade = 50, não foi considerado porque o controle de concorrência definido é o pessimista e por padrão o nível de isolamento é *Serializable*. Esse nível não permite que os resultados de transações não efetivadas sejam visualizados.

Porém, conforme o gerente utiliza o dispositivo, ele percebe que o nível da bateria está diminuindo rapidamente. Como ele optou por um limite de 5%, se o nível da bateria estiver abaixo, o monitor de recursos notificará a transação e a política de adaptação será executada. A transação iniciada tem a política de adaptação de alterar o nível de isolamento de *Serializable* para *Read Uncommitted* se o nível de bateria estiver abaixo do limite estabelecido.

Como a transação já foi iniciada, apenas o nível de isolamento pode ser alterado. Se alguma transação estiver em andamento com outro nível de isolamento, ela não será afetada. O servidor permite a execução de diferentes níveis de isolamento em paralelo.

Enquanto isso, a transação do gerente de vendas continua e como o nível de isolamento é menos restrito, as vendas não efetivadas serão visualizadas pelo gerente. Portanto, a última venda não concluída será considerada no cálculo, logo o resultado mostrado ao gerente não será mais 10.00 e sim 179.64. Foram consideradas as seguintes vendas:

Produto = "p2" com quantidade = 1 (1 x 10.00) = 10.00
Produto = "p2" com quantidade = 2 (2 x 10.00) = 20.00
Produto = "p2" com quantidade = 3 (3 x 10.00) = 30.00
Produto = "p1" com quantidade = 50 (50 x 200.00) =10000.00

Se a última venda for concluída pelo representante de vendas, a média de vendas realizadas exibida para o gerente estará correta. Porém, o representante de vendas desistiu do produto, ou seja, a última venda não foi efetivada. Sendo assim, a média das vendas seria 10.00 e não 179.64. Mas o gerente tinha necessidade de ver a informação rapidamente, pois o nível da bateria estava diminuindo e logo o dispositivo móvel poderia deixar de funcionar. Além disso, o resultado exibe uma média geral. A preocupação nesse tipo de aplicação não é a consistência dos dados e sim o desempenho. O gerente sabe que algumas vendas podem não ser efetivadas, mas diante da situação do ambiente, ele opta por considerar essas informações para tomar uma decisão.

C. Detalhes da arquitetura do protótipo implementado

No primeiro caso de uso, o servidor aciona a classe *ArchitectureServer* para montar a arquitetura com os componentes OpenCOM do lado servidor. Do lado cliente, a classe *ArchitectureClient* é a responsável pela montagem.

Como o controle de concorrência definido como padrão é o pessimista, o componente *LockManager* é conectado ao componente *ResourceManager*. Os componentes referentes ao nível de isolamento são conectados ao *ResourceManager* também. O nível de isolamento padrão é o *Serializable*. O outro nível isolamento (*ReadUncommitted*) permanece desconectado da arquitetura. O componente *TwoPhaseLocking* é conectado ao componente *TransactionManager*.

Como a interface exposta pelo framework do cliente é a *ITransactionManager*, todas as operações pertencentes a ela podem ser utilizadas pelo cliente da aplicação. Seguindo o modelo de transação, as operações chamadas são: *begin, execute*, *commit* e *rollback*. A primeira operação chamada do componente *TransactionManager* é *begin* que indica o início da transação. Seguindo a arquitetura, a operação chama os componentes *TwoPhaseLocking* e *ClientCommunicator*. No *ClientCommunicator*, o servidor é procurado no servidor de nomes RMI, realizando dessa forma a conexão cliente-servidor.

Do lado servidor, o componente *ServerCommunicator* recebe a requisição do cliente e chama o componente *ResourceManager*. Na operação de *begin* dentro desse componente, uma cópia dos objetos a serem utilizados pela transação (tabela de vendas e tabela de estoque) é inserida na lista de objetos bloqueados. O responsável por esse controle é o componente *LockManager*.

A próxima operação da transação é o *execute*. Nesse exemplo, duas operações de execução são realizadas: inserir o pedido solicitado na tabela de vendas e atualizar a tabela de estoque.

As duas operações são executadas de forma atômica, ou seja, se o produto for inserido na tabela de vendas, a tabela

de estoque deve ser atualizada. Se uma das operações falhar, a outra deve ser desfeita.

Na execução das operações, o componente *ResourceManager* pede para o componente *LockManager* verificar se o objeto está bloqueado por outra transação. Caso exista outra transação utilizando o mesmo objeto, a transação corrente não pode continuar. Outra possibilidade seria bloquear a transação corrente, mas só a primeira opção foi implementada. Em seguida, se a requisição de bloqueio teve sucesso, o componente inclui o pedido na tabela de vendas e atualiza o estoque.

Para efetivar a transação, a operação *commit* precisa ser executada.

Para que o controle de concorrência seja alterado dentro da arquitetura proposta, o monitor de recursos notifica a classe *ClientConfigurator* que contém a política de adaptação. Essa classe solicita para o componente *ArchitectureClient* que ele altere o controle de concorrência de pessimista para otimista. Esse componente, por sua vez, desconecta o componente *TwoPhaseLocking* do componente *TransactionManager* e conecta o componente *VersionControl*.

Até esse momento, o controle de concorrência foi alterado apenas do lado cliente. O mesmo deve acontecer do lado servidor. Para isso, a classe *ClientConfigurator* após solicitar a alteração do controle de concorrência do lado cliente, manda uma notificação para a classe ServerConfigurator através da tecnologia RMI. A classe *ServerConfigurator* ao receber a notificação, solicita ao componente *ArchitectureServer* a alteração do controle de concorrência de pessimista para otimista. Esse componente, por sua vez, desconecta o componente *LockManager* do componente *ResourceManager* e conecta o componente *VersionManager*.

No controle de concorrência otimista, os objetos não são bloqueados. Os objetos da transação são copiados para a máquina local do cliente (mantidos em *cache*) e todas as operações são efetuadas sobre a cópia local. Portanto, se a conexão com o servidor cair, a transação pode continuar. Essa é a fase de preparação. O responsável por esse processo é o componente *VersionControl*. Apenas no final da transação, nas fases de validação e escrita, o cliente necessita se conectar ao servidor.

Por último, no terceiro caso de uso, o monitor de recursos notifica a classe *ClientConfigurator* que verifica se a transação já começou. Como a transação já foi iniciada, apenas o nível de isolamento pode ser alterado. Se alguma transação estiver em andamento com outro nível de isolamento, ela não será afetada. Como do lado cliente não existe alteração, a classe *ClientConfigurator* notifica a classe *ServerConfigurator* através da tecnologia RMI. A classe *ServerConfigurator* solicita para o componente *ArchitectureServer* a alteração do nível de isolamento. Esse componente, por sua vez, desconecta o componente *Serializable* do componente *ResourceManager* e conecta o componente *Read Uncommitted*.

V. CONCLUSÕES

O propósito desse artigo foi demonstrar como algumas propriedades transacionais e alguns mecanismos para garanti-las, podem ser configurados e reconfigurados dinamicamente utilizando um modelo de componentes. Neste contexto, a configuração do tipo de controle de concorrência foi uma proposta inovadora em comparação aos modelos de transações existentes, permitindo a demonstração do conceito de adaptabilidade.

O modelo proposto baseou-se na idéia do modelo SGTA [7], porém utilizando o modelo de componentes OpenCOM, ao invés de CORBA [17] e também utilizando o controle de concorrência como foco principal da arquitetura. Além disso, foi implementado um protótipo para analisar os impactos da reconfiguração durante uma transação.

Dois motivos fizeram o nível de isolamento ter sido escolhido como mecanismo de adaptação: permitir pequenas reconfigurações arquiteturais durante a transação e propiciar ganhos de desempenho. Quanto mais baixo o nível de isolamento, maior o ganho de desempenho. Como esse ganho pode muitas vezes ocasionar perda de consistência, apenas em algumas situações isso seria desejável. Portanto, essa é uma boa propriedade para ser alterada durante a execução da transação. Além disso, como mostrado num dos casos de uso da seção IV, em algumas aplicações e em situações de recursos limitados, pode-se optar por melhor desempenho ao invés de nível de consistência restrito como a seriabilidade.

Notou-se que cada controle de concorrência tem a sua vantagem. Em determinadas situações, como por exemplo, quando a conexão está limitada, o controle otimista pode ser mais adequado, pois o cliente pode trabalhar com os dados localmente. Já em situações em que largura de banda e a conexão são satisfatórias, o controle pessimista pode ser utilizado, pois evita a cópia de dados o que acarretaria em largura de banda e custo de comunicação excessivos.

Porém o tipo de controle de concorrência só pôde ser alterado antes de a transação começar. A razão disso é porque quando a transação já iniciou com um determinado controle de concorrência, algumas ações já foram executadas tanto no cliente quanto no servidor. Por exemplo, se o cliente inicia a transação utilizando o controle pessimista, os objetos utilizados na transação ficam bloqueados até o final da mesma. Já o controle otimista exige que sejam feitas cópias locais dos objetos envolvidos na transação. Se durante a transação o controle de concorrência for alterado para otimista, uma cópia dos objetos será feito no cliente e o processo de validação será realizado como determinado nesse tipo de controle, porém os objetos não serão desbloqueados.

REFERÊNCIAS

[1] Android. http://www.android.com. Acessado em 01/2010.
[2] E. A. Nassu. Consultas sobre "aqui" em Sistemas de Bancos de Dados em Ambientes de Computação Nômade. Tese de Doutorado - IME-USP, Orientador: Prof. Dr. Marcelo Finger, Junho de 2003.
[3] N. Santos and P. Ferreira. Making Distributed Transactions Resilient to Intermittent Network Connections. Instituto Superior Técnico de Lisboa, 2006.
[4] R. Rouvoy , P. S. Alvarado and M. Philippe. Towards Context-Aware Transaction Services, 2006.
[5] P. S. Alvarado, C. Roncancio, M. Adiba and C. Labbé. Adaptable Mobile Transactions and Enviromnet Awareness, 2003.

[6] A. Arntsen and R. Karlsen. ReflecTS; A Flexible Transaction Service Framework. University of Tromsoe, 2005.
[7] T. Rocha and M. B. F. Toledo. Um Sistema de Transações Adaptável para o Ambiente de Computação Móvel. Simpósio Brasileiro de Redes de Computadores, 2003.
[8] OpenCOM. Computer Departament of Lancaster University. http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/opencom.php. Acessado em 01/2010.
[9] B. Smith. Reflection and Semantics in a Procedural Language. PhD thesis, Cambridge, Massachusetts, 1982.
[10] F. Backmann et. al. Volume II: Technical Concepts of Component-Based Software Engineering, 2ª edição, 2002.
[11] P. Bernstein, V. Hadzilacos and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.
[12] G. Coulouris, J. Dollimore and T. Kindberg. Distributed Systems: Concepts and Design. Third edition. Addison-Wesley, 2001.
[13] Remote Method Invocation Home. Java Sun Microsystems. Acessado em 01/2010. http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[14] http://www.pcs.usp.br/~lta/ Página web do LTA – Laboratório de Linguagens e Técnicas Adaptativas.
[15] The OpenCOMJ Handbook. Lancaster Univerty. Febreary, 2007. http://www.mirrorservice.org/sites/download.sourceforge.net/pub/sourceforge/g/gr/gridkit/OpenCOMJHandbook.pdf
[16] Java Sun Microsystems. http://java.sun.com/. Acessado em 01/2010.
[17] CORBA - Common Object Request Broker Architecture. http://www.corba.org. Acessado em 01/2010.

Allyn Grey de A. Lima Pierre é Engenheira de Computação pela Pontíficia Universidade Católica de Campinas (2002) e Mestre em Ciência da Computação pela Universidade Estadual de Campinas (2009). Atualmente trabalha no Centro de Inovação Tecnológica.

Maria Beatriz Felgar de Toledo possui graduação e mestrado em Ciência da Computação pela Universidade Estadual de Campinas e doutorado em Sistemas Distribuídos pela Universidade de Lancaster, Inglaterra (1992). Atualmente é professora associada na Universidade Estadual de Campinas. Tem interesse nas áreas de gestão de processos de negócio, serviços Web, Web semântica, bibliotecas digitais e museus.

Tarcisio da Rocha possui doutorado em Ciência da Computação pela Universidade Estadual de Campinas (2008) com um período sanduíche na Universidade de Tromso (Noruega). Atualmente é professor Adjunto da Universidade Federal de Sergipe. Suas áreas de interesse incluem sistemas distribuídos, middleware, computação móvel e transações.