

PAULO HENRIQUE LOSSIO BARROS

**UMA PROPOSTA PARA SINCRONIZAÇÃO DE DADOS  
EM DISPOSITIVOS UBÍQUOS**

Belo Horizonte  
Novembro de 2008

UNIVERSIDADE FEDERAL DE MINAS GERAIS  
CENTRO DE PESQUISA E DESENVOLVIMENTO EM ENGENHARIA ELÉTRICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

# UMA PROPOSTA PARA SINCRONIZAÇÃO DE DADOS EM DISPOSITIVOS UBÍQUOS

Proposta de dissertação apresentada ao Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Engenharia Elétrica.

PAULO HENRIQUE LOSSIO BARROS

Belo Horizonte  
Novembro de 2008

# Resumo

A onipresença de telefones inteligentes, PDAs, cartões inteligentes, sensores e outros dispositivos ubíquos, junto com o desenvolvimento de diferentes tipos de redes de dados sem fio, tem levado a uma grande mudança no uso, desenho e desenvolvimento dos sistemas de informação. Cada vez mais os dispositivos ubíquos de acesso a dados são utilizados como ferramentas de trabalho. Executivos, vendedores, trabalhadores de campo em geral precisam acessar e atualizar informações corporativas aonde quer que estejam.

Mesmo com os avanços na tecnologia de *hardware* e redes de comunicação sem fio, as desconexões são comuns. Dispositivos podem ficar desconectados por razões diversas, como ausência de cobertura de rede ou simplesmente para economizar energia. Para simular um ambiente de conexão permanente e manter a aparência de ubiquidade mesmo quando desconectadas, as aplicações podem se antecipar à desconexão e copiar parte das informações disponíveis em servidores conectados à rede fixa para os dispositivos, onde poderão ser acessadas e modificadas durante os períodos sem conexão. A funcionalidade chave deste novo paradigma da computação é a habilidade de armazenar dados localmente nos clientes móveis e sincronizar as modificações de volta para o servidor corporativo. Este recurso desempenha um papel essencial na visão da computação ubíqua e pervasiva, onde as informações estão disponíveis a todo momento e em qualquer lugar.

Neste trabalho foi proposto e implementado um *framework*, chamado mSync (*mobile Synchronization*), para sincronização de dados em dispositivos ubíquos. O *framework* trata problemas relacionados a consulta e modificações de dados em um ambiente distribuído, permitindo que desenvolvedores utilizem a sincronização de dados em suas aplicações como se fosse um serviço, podendo concentrar esforços em questões específicas da aplicação. O uso do *framework* agiliza o desenvolvimento de soluções, provendo um nível elevado de abstração que esconde dos programadores a complexidade introduzida pela distribuição.

Foram apresentados dois estudos de caso para ilustrar a eficiência e utilidade do *framework*. O primeiro, uma aplicação simples de cadastro de clientes e pedidos, foi desenvolvido com o objetivo didático de demonstrar a utilização do *framework*. O segundo estudo de caso apresentou uma aplicação real utilizada pelo Crea-MG para fiscalização de obras, empreendimentos e serviços.

# Abstract

The omnipresence of smart phones, PDAs, smart cards, sensors and other ubiquitous devices, along with the development of different types of networks for wireless data, have led to a major shift in the use, design and development of information systems. Increasingly ubiquitous devices to access data are used as tools for work. Executives, vendors, workers in general update corporate information wherever they are.

Even with the advances in technology of hardware and wireless communications networks, the disconnections are common. Devices can be disconnected for various reasons such as lack of network coverage or simply to save energy. To simulate an environment of continuous connection and maintain the appearance of ubiquity even when disconnected, the applications can anticipate the disconnection and copy some of the information available on servers connected to fixed network for the devices, which can be accessed and modified during the periods without connection. The key feature of this new paradigm of computing is the ability to store data locally on mobile clients and synchronize the changes back to the corporate server. This feature plays a key role in the vision of pervasive and ubiquitous computing, where information is available at any time and anywhere.

This work proposed and implemented a framework, called mSync (mobile Synchronization), for data synchronization in ubiquitous devices. The framework deals with problems related to the query and modification of data in a distributed environment, allowing developers to use the data synchronization in their applications as a service and to focus on specific issues of implementation. The framework speeds up the development of solutions that use data replication and synchronization, providing a high level of abstraction for developers that hide the complexity introduced by distribution.

We present two case studies to illustrate the effectiveness and usefulness of the framework. The first, a single application for registration of customers and sales, was developed to demonstrate the use of framework. The second case study presents a real application of supervision of works, projects and services, used by Crea-MG, which uses the mSync as a tool to allow consultation and modification of data in disconnected mode.

*Ao meu Pai, Marcos, meu maior parceiro, incentivador e inspirador.*

*À minha Mãe, Cida, pelo carinho e apoio em todos os momentos da minha vida.*

*Aos meus irmãos, Camilla e Lucas, pela amizade.*

*À minha noiva, Flávia Jardim, pelo amor e incentivo.*

# Agradecimentos

Aos meus professores do curso de graduação e mestrado em engenharia elétrica, pelos ensinamentos e valiosa contribuição na minha formação pessoal e profissional.

Ao meu Pai, pelo incentivo e apoio incondicional. À minha mãe e pequeno grande irmão Lucas, pela compreensão em me cederem o computador nas intermináveis horas de estudo e trabalho duro. A minha irmã Camilla que, mesmo distante, continua sendo minha maior fã e admiradora.

À minha noiva, Flávia Jardim, por me ajudar na revisão do texto, compreender os meus momentos de ausência e aceitar em organizar os preparativos para o casamento quase que sozinha.

À professora Linnyer Beatrys Ruiz pela orientação.

À Velp Soluções em Tecnologia, pelo apoio financeiro e liberdade, sem os quais não seria possível desenvolver este trabalho. À toda equipe da Velp, em especial ao Fabricio e André, pelas importantes contribuições e sugestões no desenvolvimento desta dissertação.

À UFMG, pela oportunidade de desenvolver esta dissertação de mestrado.

E a Deus por possibilitar tudo isto.

À todos o meu **Muito Obrigado!**

# Sumário

|                                                                           |             |
|---------------------------------------------------------------------------|-------------|
| <b>Lista de Figuras</b>                                                   | <b>viii</b> |
| <b>Lista de Tabelas</b>                                                   | <b>xi</b>   |
| <b>Lista de Abreviaturas</b>                                              | <b>xii</b>  |
| <b>1 Introdução</b>                                                       | <b>1</b>    |
| 1.1 Motivação . . . . .                                                   | 3           |
| 1.2 Contribuições . . . . .                                               | 3           |
| 1.3 Organização do trabalho . . . . .                                     | 4           |
| <b>2 Computação Ubíqua</b>                                                | <b>5</b>    |
| 2.1 Computação Ubíqua . . . . .                                           | 5           |
| 2.1.1 Definições . . . . .                                                | 6           |
| 2.1.2 Princípios da Computação Ubíqua . . . . .                           | 7           |
| 2.2 Dispositivos Ubíquos . . . . .                                        | 8           |
| 2.2.1 Controles Inteligentes ( <i>Smart Controls</i> ) . . . . .          | 9           |
| 2.2.2 Utensílios Inteligentes ( <i>Intelligent Appliances</i> ) . . . . . | 10          |
| 2.2.3 Sistemas de Entretenimento . . . . .                                | 11          |
| 2.2.4 Dispositivos de Acesso à Informação . . . . .                       | 11          |
| 2.3 Sistemas Operacionais . . . . .                                       | 13          |
| 2.3.1 <i>Symbian OS</i> . . . . .                                         | 14          |
| 2.3.2 <i>Windows CE</i> . . . . .                                         | 14          |
| 2.3.3 <i>Windows Mobile</i> . . . . .                                     | 16          |
| 2.3.4 <i>BlackBerry OS</i> . . . . .                                      | 16          |
| 2.3.5 <i>Linux Embarcado</i> . . . . .                                    | 16          |
| 2.3.6 <i>iPhone OS</i> . . . . .                                          | 17          |
| 2.3.7 <i>Palm OS</i> . . . . .                                            | 17          |
| 2.4 Conclusões . . . . .                                                  | 18          |
| <b>3 Replicação e Sincronização de Dados</b>                              | <b>20</b>   |
| 3.1 Consistência em bancos de dados móveis . . . . .                      | 21          |
| 3.2 Conflitos . . . . .                                                   | 22          |

|          |                                                                             |           |
|----------|-----------------------------------------------------------------------------|-----------|
| 3.3      | Protocolos para troca de dados entre os dispositivos e o servidor . . . . . | 23        |
| 3.4      | Âncoras de sincronização . . . . .                                          | 25        |
| 3.5      | Carga inicial e particionamento de dados para execução em modo desconectado | 26        |
| 3.6      | Rastreamento de modificações . . . . .                                      | 26        |
| 3.7      | Geração de identificadores únicos em um ambiente distribuído . . . . .      | 27        |
| 3.8      | Soluções comerciais de bancos de dados móveis . . . . .                     | 30        |
| 3.8.1    | <i>SQL Anywhere</i> . . . . .                                               | 30        |
| 3.8.2    | <i>Oracle Lite</i> . . . . .                                                | 31        |
| 3.8.3    | <i>DB2 Everyplace</i> . . . . .                                             | 32        |
| 3.8.4    | <i>SQL Server Mobile</i> . . . . .                                          | 33        |
| 3.9      | Trabalhos relacionados . . . . .                                            | 34        |
| 3.10     | Conclusões . . . . .                                                        | 38        |
| <b>4</b> | <b><i>Framework mSync</i></b>                                               | <b>40</b> |
| 4.1      | Âncoras de Sincronização . . . . .                                          | 41        |
| 4.2      | Erros e Conflitos . . . . .                                                 | 43        |
| 4.2.1    | Detecção de Erros e Conflitos . . . . .                                     | 43        |
| 4.2.2    | Resolução de Erros e Conflitos . . . . .                                    | 44        |
| 4.3      | Rastreamento de modificações . . . . .                                      | 44        |
| 4.3.1    | Rastreamento de modificações no dispositivo . . . . .                       | 44        |
| 4.3.2    | Rastreamento de modificações no banco de dados central . . . . .            | 45        |
| 4.4      | Geração de identificadores únicos no <i>framework mSync</i> . . . . .       | 48        |
| 4.5      | Componentes da arquitetura . . . . .                                        | 50        |
| 4.5.1    | SyncAgent . . . . .                                                         | 52        |
| 4.5.2    | ISyncClientMetaDataProvider e SyncClientMetaDataProvider . . . . .          | 52        |
| 4.5.3    | SyncElement . . . . .                                                       | 52        |
| 4.5.4    | ISyncClient . . . . .                                                       | 52        |
| 4.5.5    | ClientSyncDAO . . . . .                                                     | 53        |
| 4.5.6    | Client Database . . . . .                                                   | 53        |
| 4.5.7    | ISyncServer . . . . .                                                       | 53        |
| 4.5.8    | GenericServerAdapter . . . . .                                              | 54        |
| 4.5.9    | SyncWebService . . . . .                                                    | 54        |
| 4.5.10   | IServerSyncDAO . . . . .                                                    | 54        |
| 4.5.11   | ServerSyncDAO . . . . .                                                     | 54        |
| 4.5.12   | ServerDatabase . . . . .                                                    | 54        |
| 4.6      | Modelo de reuso . . . . .                                                   | 54        |
| 4.7      | Sincronização de dados . . . . .                                            | 56        |
| 4.7.1    | Sincronização sentido dispositivo móvel para servidor fixo . . . . .        | 57        |
| 4.7.2    | Sincronização sentido servidor fixo para dispositivo móvel . . . . .        | 61        |
| 4.8      | Conclusões . . . . .                                                        | 63        |
| <b>5</b> | <b>Estudos de Caso</b>                                                      | <b>64</b> |

|          |                                                                                                     |            |
|----------|-----------------------------------------------------------------------------------------------------|------------|
| 5.1      | Estudo de caso 1: Cadastro de Clientes e Pedidos . . . . .                                          | 64         |
| 5.1.1    | Preparação do Banco de Dados Central e Servidor de Sincronização . . . . .                          | 64         |
| 5.1.2    | Preparação do Dispositivo Móvel . . . . .                                                           | 69         |
| 5.2      | Estudo de caso 2: Sistema de fiscalização de obras, empreendimentos e serviços do Crea-MG . . . . . | 73         |
| 5.2.1    | GeFisc Web . . . . .                                                                                | 75         |
| 5.2.2    | GeFisc Móvel . . . . .                                                                              | 75         |
| 5.2.3    | Sincronização de Dados . . . . .                                                                    | 77         |
| 5.3      | Resultados . . . . .                                                                                | 81         |
| 5.4      | Conclusão . . . . .                                                                                 | 88         |
| <b>6</b> | <b>Conclusão</b>                                                                                    | <b>92</b>  |
| <b>A</b> | <b>Implementação do <i>Framework</i> mSync</b>                                                      | <b>96</b>  |
| A.1      | mSyncCommon . . . . .                                                                               | 96         |
| A.2      | mSyncClient . . . . .                                                                               | 98         |
| A.2.1    | Interface ISyncClient . . . . .                                                                     | 98         |
| A.2.2    | ClientSyncDAO . . . . .                                                                             | 101        |
| A.2.3    | Interface ISyncServer . . . . .                                                                     | 102        |
| A.2.4    | Server Adapter . . . . .                                                                            | 105        |
| A.2.5    | SyncElement . . . . .                                                                               | 106        |
| A.2.6    | Interface ISyncClientMetaDataProvider . . . . .                                                     | 106        |
| A.2.7    | BaseSyncClientMetaDataProvider . . . . .                                                            | 107        |
| A.2.8    | SyncAgent . . . . .                                                                                 | 108        |
| A.3      | mSyncServer . . . . .                                                                               | 109        |
| A.3.1    | BaseSyncWebService . . . . .                                                                        | 110        |
| A.3.2    | Interface ISyncServerDAO . . . . .                                                                  | 111        |
| A.3.3    | ServerSyncDAO . . . . .                                                                             | 111        |
| A.4      | Requisitos mínimos de <i>hardware</i> e <i>software</i> . . . . .                                   | 112        |
| A.5      | Flexibilidade com o mSync . . . . .                                                                 | 113        |
|          | <b>Referências Bibliográficas</b>                                                                   | <b>115</b> |

# Lista de Figuras

|      |                                                                                            |    |
|------|--------------------------------------------------------------------------------------------|----|
| 1.1  | <i>Framework</i> mSync . . . . .                                                           | 2  |
| 2.1  | Relação entre computação ubíqua, pervasiva e móvel . . . . .                               | 7  |
| 2.2  | Uma classificação dos dispositivos ubíquos. [Adaptado de Hansmann et al. (2001)] . . . . . | 9  |
| 2.3  | Sensores sem fio . . . . .                                                                 | 10 |
| 2.4  | Exemplos de PDAs . . . . .                                                                 | 12 |
| 2.5  | Exemplos de coletores de dados . . . . .                                                   | 12 |
| 2.6  | Exemplos de telefones inteligentes . . . . .                                               | 13 |
| 2.7  | Exemplo de <i>tabletPC</i> . . . . .                                                       | 13 |
| 2.8  | Exemplos de interfaces Symbian OS . . . . .                                                | 14 |
| 2.9  | iPhone da Apple . . . . .                                                                  | 18 |
| 3.1  | Ambiente cliente / servidor de banco de dados móvel . . . . .                              | 21 |
| 3.2  | Exemplo de mapeamento de identificadores no SyncML . . . . .                               | 28 |
| 3.3  | Funcionamento do método CPISync . . . . .                                                  | 35 |
| 3.4  | <i>Framework</i> SyncML . . . . .                                                          | 37 |
| 3.5  | Exemplo de mapeamento objeto relacional utilizando o GDA . . . . .                         | 38 |
| 3.6  | Exemplo de persistência de dados utilizando o GDA . . . . .                                | 39 |
| 4.1  | Cenário de aplicação do <i>framework</i> mSync . . . . .                                   | 40 |
| 4.2  | Fluxo para operação em modo desconectado . . . . .                                         | 41 |
| 4.3  | Estrutura da tabela appTables . . . . .                                                    | 42 |
| 4.4  | Estrutura da tabela tombStone . . . . .                                                    | 46 |
| 4.5  | Exemplo do uso de âncoras no rastreamento de modificações . . . . .                        | 46 |
| 4.6  | Estrutura da tabela de controle de alterações . . . . .                                    | 47 |
| 4.7  | Estrutura da tabela idMapping . . . . .                                                    | 50 |
| 4.8  | Exemplo de conversão de LUID para GUID . . . . .                                           | 51 |
| 4.9  | Componentes do <i>framework</i> mSync . . . . .                                            | 51 |
| 4.10 | Diagrama de seqüência do início do processo de sincronização . . . . .                     | 58 |
| 4.11 | Diagrama de seqüência do processo de sincronização sentido DM -> SF . . . . .              | 59 |
| 4.12 | Diagrama de seqüência do processo de sincronização sentido SF -> DM . . . . .              | 62 |
| 5.1  | Caso 1: Tabelas do servidor fixo replicadas nos dispositivos . . . . .                     | 65 |

|      |                                                                                                                               |     |
|------|-------------------------------------------------------------------------------------------------------------------------------|-----|
| 5.2  | Caso 1: Povoamento inicial da tabela AppTables . . . . .                                                                      | 65  |
| 5.3  | Caso 1: <i>Script</i> de criação da tabela Clientes . . . . .                                                                 | 66  |
| 5.4  | Caso 1: <i>Script</i> de criação da função <i>getIdSync</i> . . . . .                                                         | 66  |
| 5.5  | Caso 1: Gatilho de atualização da tabela Clientes . . . . .                                                                   | 66  |
| 5.6  | Caso 1: Gatilho de exclusão da tabela Clientes . . . . .                                                                      | 67  |
| 5.7  | <i>Script</i> de criação da função <i>getCodTable</i> . . . . .                                                               | 67  |
| 5.8  | <i>Script</i> de criação da função <i>newIdSync</i> . . . . .                                                                 | 67  |
| 5.9  | Caso 1: Classe de modelo da tabela Clientes utilizada no servidor . . . . .                                                   | 68  |
| 5.10 | Caso 1: DAO da tabela clientes utilizada no servidor . . . . .                                                                | 68  |
| 5.11 | Web service utilizado no estudo de caso 1 . . . . .                                                                           | 69  |
| 5.12 | <i>DAO Factory</i> utilizado no estudo de caso 1 . . . . .                                                                    | 70  |
| 5.13 | Caso 1: Tabelas utilizadas no banco de dados móvel . . . . .                                                                  | 71  |
| 5.14 | Caso 1: Instanciação do <i>SyncAgent</i> . . . . .                                                                            | 72  |
| 5.15 | Caso 1: Instanciação do <i>GenericWebServiceAdapter</i> . . . . .                                                             | 72  |
| 5.16 | Caso 1: Classe <i>WebServiceC1</i> . . . . .                                                                                  | 73  |
| 5.17 | Caso 1: Instanciação dos elementos de sincronização . . . . .                                                                 | 74  |
| 5.18 | Caso 1: Configuração dos eventos do <i>SyncAgent</i> . . . . .                                                                | 74  |
| 5.19 | Dispositivos utilizados no sistema de fiscalização do Crea-MG . . . . .                                                       | 75  |
| 5.20 | Tela de visualização da rota percorrida pelo fiscal no GeFisc Web . . . . .                                                   | 76  |
| 5.21 | Tela de objetos de fiscalização do GeFisc Web . . . . .                                                                       | 76  |
| 5.22 | Telas do GeFisc Móvel . . . . .                                                                                               | 78  |
| 5.23 | Caso 2: Exemplo de utilização do evento <i>AfterInsertServer</i> . . . . .                                                    | 79  |
| 5.24 | Caso 2: Exemplo de utilização da propriedade <i>SetClientMsgPacketSize</i> . . . . .                                          | 80  |
| 5.25 | Caso 2: Classe <i>ObjFiscDAO</i> . . . . .                                                                                    | 81  |
| 5.26 | Cenário utilizado nos testes de performance . . . . .                                                                         | 83  |
| 5.27 | Dispositivo ubíquo utilizado nos testes de performance . . . . .                                                              | 84  |
| 5.28 | Tabela TBPESSOA . . . . .                                                                                                     | 85  |
| 5.29 | Tempos de sincronização sentido dispositivo para servidor . . . . .                                                           | 86  |
| 5.30 | Comparação entre os tempos das operações no sentido dispositivo para servidor .                                               | 87  |
| 5.31 | Tempos de sincronização sentido servidor para dispositivo . . . . .                                                           | 89  |
| 5.32 | Comparação entre os tempos das operações no sentido servidor para dispositivo .                                               | 90  |
| 5.33 | Comparação entre os tempos de sincronização nos sentidos dispositivo para servi-<br>dor e servidor para dispositivo . . . . . | 91  |
| A.1  | Classe <i>SyncSession</i> . . . . .                                                                                           | 97  |
| A.2  | Objetos de retorno das funções de exclusão, atualização e inserção de registros no<br>servidor . . . . .                      | 97  |
| A.3  | Enumerações utilizadas no mSync . . . . .                                                                                     | 98  |
| A.4  | Classe <i>BaseDAO</i> . . . . .                                                                                               | 99  |
| A.5  | Interface <i>ISyncClient</i> . . . . .                                                                                        | 99  |
| A.6  | Classe <i>ClientBaseSyncDAO</i> . . . . .                                                                                     | 102 |

|      |                                              |     |
|------|----------------------------------------------|-----|
| A.7  | Classe <i>ClientBaseSimpleSyncDAO</i>        | 103 |
| A.8  | Interface <i>ISyncServer</i>                 | 103 |
| A.9  | <i>GenericWebserviceAdapter</i>              | 105 |
| A.10 | <i>SyncElement</i>                           | 106 |
| A.11 | Interface <i>ISyncClientMetaDataProvider</i> | 107 |
| A.12 | Classe <i>BaseSyncClientMetaDataProvider</i> | 107 |
| A.13 | Estrutura da tabela appTables                | 108 |
| A.14 | Classe <i>SyncAgent</i>                      | 108 |
| A.15 | Classe <i>BaseSyncWebService</i>             | 111 |
| A.16 | Interface <i>ISyncServerDAO</i>              | 111 |
| A.17 | Classe <i>BaseSyncDAO</i>                    | 112 |
| A.18 | Classe <i>BaseSimpleSyncDAO</i>              | 113 |

# Lista de Tabelas

|     |                                                                  |    |
|-----|------------------------------------------------------------------|----|
| 2.1 | Ondas da computação segundo Weiser e Brown (1997) . . . . .      | 5  |
| 4.1 | Exemplo de registros da tabela appTables . . . . .               | 42 |
| 5.1 | Dispositivo Ubíquo (figura 5.27) . . . . .                       | 82 |
| 5.2 | Servidor de Sincronização . . . . .                              | 82 |
| 5.3 | Servidor de Banco de Dados . . . . .                             | 83 |
| 5.4 | Tempos de Sincronização (s): Dispositivo para Servidor . . . . . | 84 |
| 5.5 | Tempos de Sincronização (s): Servidor para Dispositivo . . . . . | 88 |

# Lista de Abreviaturas

- **ACID:** Sigla utilizada para caracterizar as propriedades desejáveis em transações tradicionais. A-Atomicidade, C-Consistência, I-Isolamento e D-Durabilidade.
- **ART:** Anotação de Responsabilidade Técnica.
- **BDC:** Banco de dados central.
- **BDM:** Banco de dados móvel.
- **CPF:** Cadastro de Pessoa Física.
- **CNPJ:** Cadastro Nacional de Pessoa Jurídica.
- **DM:** Dispositivo Móvel.
- **DAO:** *Data Access Object*.
- **EGDE:** *Enhanced Data rates for GSM Evolution*.
- **GDA:** *Generic Data Access. Middleware* utilizado para acesso e persistência de dados.
- **GPRS:** *General Packet Radio Service* ou Serviço Geral de Pacotes por Rádio.
- **GSM:** *Global System Mobile* ou Sistema Móvel Global.
- **GUID:** *Globally Unique ID*. Identificador único global que identifica cada linha no banco de dados central.
- **HTML:** *Hypertext Transfer Protocol*.
- **idSync:** Identificador de Sincronizações. Âncora utilizada na arquitetura mSync.
- **LUID:** *Locally Unique ID*. Identificador único local que identifica cada linha no banco de dados local.
- **SF:** Servidor da Rede Fixa.
- **SGBD:** Sistema Gerenciador de Banco de Dados.
- **SOAP:** *Simple Object Access Protocol*.

- 
- **WAP:** *Wireless Application Protocol* ou Protocolo para Aplicação sem Fio.
  - **WBXML:** WAP Binary XML.
  - **WSNs:** *Wireless Sensor Networks*.
  - **XML:** eXtensible Markup Language.

# Capítulo 1

## Introdução

Computação ubíqua, como o termo implica, refere-se à capacidade de estar conectado à rede e fazer uso da conexão constantemente, a todo momento, e em todo lugar (Weiser (1999)).

A onipresença de telefones inteligentes, PDAs, cartões inteligentes, sensores e outros dispositivos ubíquos, junto com o desenvolvimento de diferentes tipos de redes de dados sem fio, têm levado a uma grande mudança no uso, desenho e desenvolvimento dos atuais e futuros sistemas de informação. Cada vez mais os dispositivos ubíquos de acesso a dados são utilizados como ferramentas de trabalho. Executivos, vendedores, trabalhadores de campo em geral precisam acessar e atualizar informações corporativas aonde quer que estejam.

Mesmo com os avanços na tecnologia de *hardware* e redes de comunicação sem fio, as desconexões são comuns. Dispositivos podem ficar desconectados por razões diversas, como ausência de cobertura de rede ou simplesmente para economizar energia. Para simular um ambiente de conexão permanente e manter a aparência de ubiquidade mesmo quando desconectadas, as aplicações podem se antecipar à desconexão e copiar parte das informações disponíveis em servidores conectados à rede fixa para os dispositivos, onde poderão ser acessadas e modificadas durante os períodos sem conexão. A funcionalidade chave deste novo paradigma da computação é a habilidade de armazenar dados localmente nos clientes móveis e sincronizar as modificações de volta para o servidor corporativo. Este recurso desempenha um papel essencial na visão da computação ubíqua e pervasiva, onde as informações estão disponíveis a todo momento e em qualquer lugar (Choi et al. (2005)).

Sincronização de dados envolve diversas áreas de pesquisa, como bancos de dados, computação distribuída, comunicações móveis e muitas outras, e tem despertado grande interesse de pesquisadores e empresas. Muitos esforços estão sendo despendidos para melhorar o gerenciamento de dados em ambientes móveis, e soluções têm sido propostas em diversas áreas (Serrano-Alvarado et al. (2004)).

Alguns bancos de dados comerciais, como Oracle Lite e Microsoft SQL Server Mobile, oferecem suporte para sincronização de dados, mas são soluções proprietárias, limitadas aos sistemas de gerenciamento de bancos de dados fornecidos pelas empresas desenvolvedoras. Na literatura científica são encontrados vários trabalhos relacionados ao tema. Muitos destes, como Tarkoma et al. (2006), Ratner et al. (2004) e Zhang et al. (2003), se ocupam da sincro-

nização de arquivos. Outros, como Helal et al. (2001), Phatak e Nath (2004) e Citro et al. (2007), estão focados em pontos específicos do processo de sincronização, como carga antecipada de dados e resolução de conflitos.

O desenvolvimento de soluções computacionais que permitam que dispositivos ubíquos acessem e modifiquem dados em modo desconectado não é uma tarefa simples. O desenvolvedor precisa compreender e se preocupar com questões como carga antecipada de dados, rastreamento de modificações nos bancos de dados móveis e fixo, geração de identificadores únicos em um ambiente distribuído, identificação / resolução de conflitos, dentre outras. Todos estes fatores geram um dificultador para o avanço na proliferação de soluções ubíquas e concretização da visão da computação pervasiva.

Este trabalho apresenta o mSync (*mobile Synchronization*), um *framework* aberto para sincronização de dados em dispositivos ubíquos. Conforme apresentado na figura 1.1, o *framework* se situa entre a aplicação e o banco de dados, oferecendo a infra-estrutura necessária para o desenvolvimento de soluções que utilizem sincronização de dados, conforme será detalhado no decorrer do texto. O *framework* esconde a complexidade do processo de sincronização, permitindo que programadores sem conhecimentos específicos relacionados à replicação e sincronização de dados utilizem a infra-estrutura disponibilizada pela arquitetura para integrar estas funcionalidades às aplicações, podendo, assim, se concentrar nas questões específicas da aplicação desenvolvida.

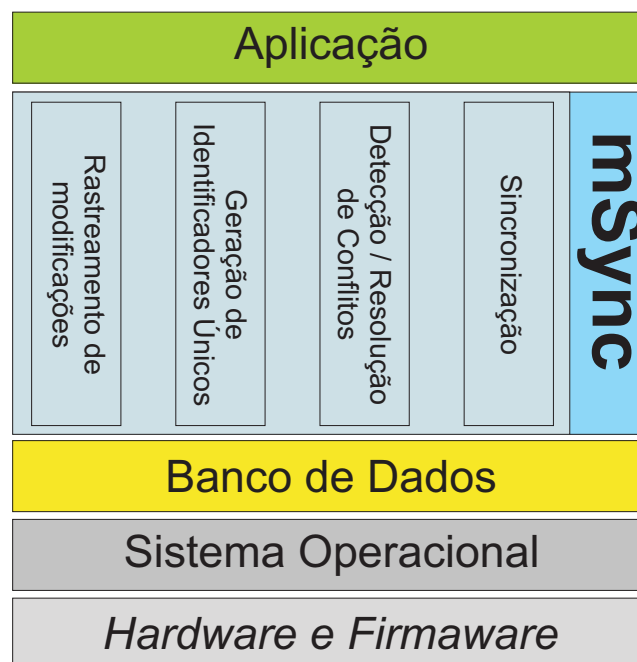


Figura 1.1: *Framework* mSync

Espera-se, com este trabalho, contribuir para o avanço da área de computação ubíqua, permitindo que mais pessoas desenvolvam soluções de mobilidade. Como prova de conceito, um estudo de caso real é apresentado. O *framework* está sendo utilizado em uma aplicação de fiscalização de obras, empreendimentos e serviços do Conselho Regional de Engenharia,

Arquitetura e Agronomia do Estado de Minas Gerais (Crea-MG), o que ajuda a demonstrar a utilidade e eficácia do trabalho proposto.

## 1.1 Motivação

As principais soluções de sincronização de dados disponíveis no mercado são proprietárias e específicas para determinadas plataformas ou bancos de dados. Um programador interessado em desenvolver aplicações de mobilidade, que não queira utilizar uma solução proprietária ou gastar dinheiro com licenças, tem que implementar sua própria rotina de sincronização, o que é uma tarefa complexa, exigindo muito tempo e conhecimento.

Com o objetivo de simplificar e agilizar o desenvolvimento de soluções ubíquas, este trabalho propõe o mSync, um *framework* aberto para replicação e sincronização de dados. O *framework* trata problemas relacionados à consulta e modificação de dados em um ambiente distribuído, oferecendo a infra-estrutura necessária para que programadores utilizem a sincronização de dados em suas aplicações como se fosse um serviço, podendo concentrar esforços em questões específicas da aplicação.

## 1.2 Contribuições

Este trabalho detalha os conceitos, processos e boas práticas relacionadas à replicação e sincronização de dados. A principal contribuição é a proposta e implementação de um *framework* aberto, baseado em padrões como XML e SOAP, direcionado para o desenvolvimento de soluções ubíquas. O uso do *framework* agiliza o desenvolvimento de soluções que utilizem replicação e sincronização de dados, provendo um nível elevado de abstração que esconde dos programadores a complexidade introduzida pela distribuição.

Os benefícios oferecidos pelo *framework* incluem:

- Permitir que uma aplicação continue útil mesmo quando desconectada, por meio do acesso e atualização de réplicas de dados armazenadas localmente nos dispositivos;
- Permitir que vários usuários compartilhem informações e que as modificações realizadas por um usuário sejam disponibilizadas para os demais;
- Prover mecanismos para detecção e resolução de conflitos, permitindo que desenvolvedores especifiquem regras baseadas na estrutura e semântica das aplicações;
- Propor um novo esquema para geração de identificadores de registros únicos em um ambiente distribuído;
- Prover funcionalidades que permitam o rastreamento das modificações realizadas nas bases de dados, tanto nos dispositivos quanto no servidor;
- Manter o volume de dados transmitido proporcional à quantidade de modificações recebidas ou enviadas para o servidor;

- Suportar sincronização em modo assíncrono, permitindo que o processo ocorra em *background*, sem intervenção dos usuários;
- Garantir a integridade das informações, evitando perda e duplicação de dados na ocorrência de falhas de comunicação durante o processo de sincronização de dados;
- Prover mecanismos que permitam que os desenvolvedores personalizem o processo de sincronização para atender a necessidades específicas das aplicações.

### 1.3 Organização do trabalho

Esta dissertação está dividida em seis Capítulos. O Capítulo 2 apresenta os conceitos e definições referentes à computação móvel, pervasiva e ubíqua. Esse Capítulo também apresenta as principais classes de dispositivos ubíquos, com ênfase especial nos dispositivos de acesso à informação, que são os mais relevantes para a dissertação. O texto revisa ainda os principais sistemas operacionais utilizados pelos dispositivos ubíquos.

O Capítulo 3 trata das questões relacionadas à replicação e sincronização de dados, apresentando os conceitos referentes à consistência em bancos de dados móveis, conflitos, protocolos para troca de dados entre os dispositivos móveis e o servidor de sincronização, carga inicial e particionamento de dados, rastreamento de modificações nos bancos de dados móveis e fixo e geração de identificadores únicos em ambientes distribuídos. Esse Capítulo também apresenta algumas soluções comerciais para bancos de dados móveis e revisa os principais trabalhos relacionados.

O Capítulo 4 apresenta o *framework* mSync, detalhando como as questões levantadas no Capítulo 3 são resolvidas no *framework*. O Capítulo apresenta os componentes da arquitetura e detalha o fluxo do processo de sincronização proposto, tanto no sentido dispositivo para servidor, quanto servidor para dispositivo.

O Capítulo 5 apresenta dois estudos de caso. O primeiro é uma aplicação simples de cadastro de clientes e pedidos, desenvolvida com o objetivo didático de demonstrar a utilização do *framework*. O segundo é uma aplicação real de fiscalização de obras, empreendimentos e serviços utilizada pelo Crea-MG, que demonstra o funcionamento e eficácia do trabalho proposto.

O Capítulo 6 conclui o trabalho, apresentando os comentários finais e sugestões para trabalhos futuros.

## Capítulo 2

# Computação Ubíqua

Este Capítulo trata das questões relacionadas à computação ubíqua. A Seção 2.1 define a terminologia, conceitos e princípios da computação ubíqua, móvel e pervasiva. A Seção 2.2 apresenta uma classificação e definição das principais categorias de dispositivo ubíquos, com ênfase especial aos dispositivos de acesso à informação, que são os mais relevantes para a dissertação. A Seção 2.3 apresenta os principais sistemas operacionais utilizados em dispositivos ubíquos de acesso a dados. As conclusões do Capítulo são apresentadas na Seção 2.4, que finaliza o texto.

### 2.1 Computação Ubíqua

O minidicionário da língua portuguesa, Rocha (2006), define o verbete ubíquo como “Que está em todos os lugares ao mesmo tempo; onipresente”. Weiser (1999), considerado o pai da computação ubíqua, utilizou este termo para se referir à terceira onda da computação, onde a computação move-se para fora das estações de trabalho e computadores pessoais (PCs) e passa a habitar os objetos triviais da vida cotidiana, tais como interruptores de luz, etiquetas de roupas, xícaras de café, canetas, etc, de forma invisível para o usuário.

Tabela 2.1: Ondas da computação segundo Weiser e Brown (1997)

|                                                        |                                            |
|--------------------------------------------------------|--------------------------------------------|
| Mainframe                                              | Muitas pessoas compartilham um processador |
| Computador Pessoal (PCs)                               | Um processador, uma pessoa                 |
| <b>Crescimento da Internet e sistemas distribuídos</b> |                                            |
| Computação Ubíqua                                      | Muitos processadores, uma pessoa           |

A essência da computação ubíqua é criação de ambientes repletos de dispositivos computacionais onde o foco seja o ser humano, ou a tarefa que ele deseja realizar, permitindo assim ao usuário dedicar-se às questões de maior interesse, deixando o ambiente pervasivo responsável pela execução das tarefas secundárias. Esses ambientes de computação são espaços físicos quaisquer (salas de aula, escritórios, edifícios, etc), e os dispositivos podem ser sensores, computadores e outros, capazes de realizar alguma computação útil aos frequentadores do ambiente.

Uma das principais características desses ambientes é que a interação entre os usuários e os dispositivos ocorre de forma tranqüila, o que é conhecido na literatura pelo termo tecnologia calma, do inglês *Calm Technology*. Os usuários não precisam se lembrar ou até mesmo saber que estão interagindo com máquinas, mas apenas usufruir dos benefícios da computação que ocorre em *background*. Esta forma transparente de interação entre pessoas e elementos computacionais é conhecida como computação invisível (Weiser e Brown (1997)).

Ainda segundo Weiser (1999), a computação ubíqua seria o inverso da realidade virtual. Enquanto a realidade virtual coloca as pessoas dentro de um mundo gerado pelo computador, a computação ubíqua força os computadores a viverem no mundo real junto com as pessoas.

### 2.1.1 Definições

Por ser uma área emergente de pesquisa, termos como computação ubíqua, computação pervasiva, computação móvel e outros tantos, têm sido usados muitas vezes como sinônimos, embora sejam diferentes conceitualmente e empreguem diferentes idéias de organização e gerenciamento dos serviços computacionais. Na medida em que a área evolui, esses conceitos vão sendo melhor compreendidos e suas definições se tornam mais claras. A definição e diferenciação entre estes conceitos, segundo de Araujo (2003), são mostradas a seguir.

#### 2.1.1.1 O que é Computação Móvel?

A computação móvel baseia-se no aumento da nossa capacidade de mover fisicamente serviços computacionais conosco, ou seja, o computador torna-se um dispositivo sempre presente que expande a capacidade de um usuário utilizar os serviços que um computador oferece, independentemente de sua localização. Combinada com a capacidade de acesso, a computação móvel tem transformado a computação numa atividade que pode ser carregada para qualquer lugar.

#### 2.1.1.2 O que é Computação Pervasiva?

O conceito de computação pervasiva implica que o computador está embarcado no ambiente de forma invisível para o usuário. Nesta concepção, o computador tem a capacidade de obter informação do ambiente no qual ele está embarcado e utilizá-la para dinamicamente construir modelos computacionais, ou seja, controlar, configurar e ajustar a aplicação para melhor atender as necessidades do dispositivo ou usuário. O ambiente também pode e deve ser capaz de detectar outros dispositivos que venham a fazer parte dele. Desta interação surge a capacidade de computadores agirem de forma “inteligente” no ambiente no qual nos movemos, um ambiente povoado por sensores e serviços computacionais.

#### 2.1.1.3 O que é Computação Ubíqua?

A computação ubíqua beneficia-se dos avanços da computação móvel e da computação pervasiva. A computação ubíqua surge da necessidade de se integrar mobilidade com a funcionalidade da computação pervasiva, ou seja, qualquer dispositivo computacional, enquanto em

movimento conosco, pode construir, dinamicamente, modelos computacionais dos ambientes nos quais nos movemos e configurar seus serviços dependendo da necessidade.

#### 2.1.1.4 Como se relacionam as computações ubíqua, pervasiva e móvel?

Tendo em vista todas as definições mencionadas acima, o termo computação ubíqua será usado aqui como uma junção da computação pervasiva e da computação móvel. A justificativa de se realizar uma diferenciação desses termos é que um dispositivo que está embutido em um ambiente não necessariamente é móvel. Devido a isso, quando for utilizado o termo computação ubíqua, considerar-se-ão o alto grau de dispositivos embarcados da computação pervasiva, juntamente com o alto grau de mobilidade da computação móvel, conforme mostra a figura 2.1.

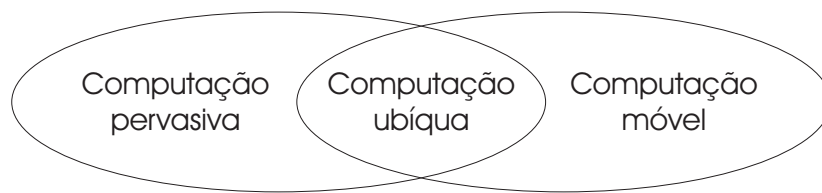


Figura 2.1: Relação entre computação ubíqua, pervasiva e móvel

#### 2.1.2 Princípios da Computação Ubíqua

Alguns princípios essenciais da computação ubíqua são destacados a seguir (adaptado de de Araujo (2003) e Hansmann et al. (2001)).

- **Simplicidade:** Weiser e Brown (1997) utilizou o termo tecnologia calma para se referir à tecnologia “que informa, mas não demanda nosso foco ou atenção”. Na computação ubíqua o computador não deve ser o centro das atenções. O objetivo da computação é ajudar uma pessoa a realizar uma atividade sem que ela precise se preocupar com a utilização da computação em si. Para atingir esta meta, os dispositivos computacionais devem exigir o mínimo possível de intervenção humana, o que vem ao encontro da definição de computação pervasiva. O *framework* mSync, proposto neste trabalho, está de acordo com o princípio da simplicidade, uma vez que ele fornece uma infra-estrutura computacional que possibilita que o processo de sincronização de dados seja realizado em pano de fundo, de forma assíncrona, sem necessidade de intervenção do usuário.
- **Diversidade:** Ao contrário do PC, que é um dispositivo de propósito geral que atende a várias necessidades distintas do usuário, tais como: edição de texto, contabilidade, navegação na web, etc., os dispositivos ubíquos acenam com uma nova visão da funcionalidade do computador, que é a de propósito específico, que atende necessidades específicas de usuários particulares. Apesar de vários dispositivos poderem oferecer funcionalidades que se sobrepõem, um pode ser mais apropriado para uma função do que outro. Os princípios de sincronização de dados propostos neste trabalho e utilizados

no *framework* mSync, seguindo o princípio da diversidade, são aplicáveis a diferentes tipos de dispositivos, desde que os mesmos possuam memória e capacidade computacional suficientes para permitir o armazenamento e processamento local de dados.

- **Descentralização:** Na computação ubíqua as responsabilidades são distribuídas entre vários dispositivos pequenos que assumem e executam certas tarefas e funções. Estes dispositivos cooperam entre si para a construção de inteligência no ambiente, que é refletida nas aplicações. Para isso, uma rede dinâmica de relações é formada entre os dispositivos e entre dispositivos e servidores do ambiente, caracterizando um sistema distribuído. O *framework* mSync contribui para a concretização do princípio da descentralização, possibilitando que diferentes dispositivos sincronizem informações com um servidor centralizado.
- **Conectividade:** Na computação ubíqua tem-se a visão da conectividade sem fronteiras, em que dispositivos e as aplicações que neles são executadas movem-se juntamente com o usuário, de forma transparente, entre diversas redes heterogêneas, tais como as redes sem fio de longa distância e redes de média e curta distância. O *framework* proposto nesta dissertação é independente da camada de enlace, podendo ser utilizado em diferentes tipos de redes, como *bluetooth*, 802.11 e GPRS (Tanenbaum (2003)). Além disto, por permitir que os dados continuem disponíveis mesmo na ausência de conexão, o mSync contribui para visão de conectividade sem fronteiras, simulando um ambiente onde a conectividade é onipresente.

## 2.2 Dispositivos Ubíquos

A computação ubíqua implica que dispositivos computacionais estarão presentes em todos os lugares, como residências, meios de transporte, escritórios e até no corpo do usuário, oferecendo serviços relacionados à informação, segurança, comodidade, entretenimento e muitos outros. Para prover estes serviços, pode-se fazer uso de vários tipos de dispositivos, tais como celulares, relógios, controles, sensores, atuadores para residências e automóveis, eletrodomésticos, ar-condicionado, aquecedores, etiquetas inteligentes, TVs, PDAs, etc...

A existência de múltiplos dispositivos, seja para acesso à informação, ao entretenimento, embutidos em utensílios domésticos, embarcados em ambientes inteligentes, etc., constitui-se em um dos desafios da computação ubíqua [de Araujo (2003)].

A figura 2.2 apresenta uma possibilidade de classificação dos vários dispositivos em quatro grandes categorias. Esta classificação é útil por simplificar o estudo das diferentes tecnologias, mas não deve ser considerada absoluta, pois muitos dispositivos, como, por exemplo, celulares que tocam MP3, se encontram em áreas intermediárias entre diferentes grupos.

A Seção 2.2.1 trata do grupo dos controles inteligentes. A Seção 2.2.2 apresenta os utensílios inteligentes. A Seção 2.2.3 introduz os dispositivos dedicados ao entretenimento. A Seção 2.2.4 trata dos dispositivos de acesso à informação, cuja função principal é disponibi-



Figura 2.2: Uma classificação dos dispositivos ubíquos. [Adaptado de Hansmann et al. (2001)]

lizar informações para os usuários e, por este motivo, são os mais relevantes para o trabalho proposto.

### 2.2.1 Controles Inteligentes (*Smart Controls*)

Os controles inteligentes são pequenos dispositivos computacionais que realizam funções específicas e dedicadas, e, normalmente, são integrados a outros equipamentos, tais como lâmpadas, interruptores, aquecedores, refrigeradores, sensores, atuadores, etc.

Controles inteligentes são utilizados em diversas áreas, como automação residencial, onde sensores e atuadores espalhados por uma residência controlam a temperatura dos sistemas de aquecimento/resfriamento e o acender/apagar de luzes, aplicações de segurança, onde sensores monitoram a entrada de intrusos, e aplicações de automação industrial, controlando os processos de manufatura.

Estes dispositivos também podem comunicar-se entre si, por exemplo, em uma aplicação onde um relógio com alarme envia dados para ajustar um aquecedor em um banheiro. São exemplos de controles inteligentes:

- Controles de processos de manufatura;

- Controles residenciais utilizados em aplicações de domótica;
- Termostatos eletrônicos;
- Etiquetas inteligentes;
- Controles de bombas de ar, água, gás, etc.;
- Cartões inteligentes.

### Redes de Sensores Sem Fio

Uma área de pesquisa bastante ativa são as redes de sensores sem fio (*Wireless Sensor Networks* - WSNs) (Loureiro et al. (2003)). Essas redes são formadas por uma grande quantidade de dispositivos sensores, interconectados por um canal sem fio, com a finalidade de realizar tarefas de sensoriamento de forma distribuída e cooperativa. Redes de sensores sem fio podem ser utilizadas em muitas aplicações, como monitoramento ambiental em florestas e oceanos, monitoramento de tráfego de veículos em rodovias, monitoramento de órgãos do corpo humano, dentre outras. A figura 2.3 apresenta alguns exemplos de nós sensores.

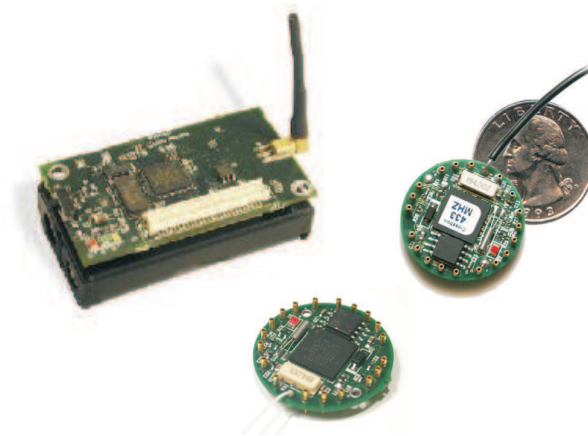


Figura 2.3: Sensores sem fio

#### 2.2.2 Utensílios Inteligentes (*Intelligent Appliances*)

É difícil estabelecer uma linha divisória clara entre as classes de dispositivos, em especial os controles inteligentes e os utensílios inteligentes. Os controles inteligentes podem estar embutidos em utensílios que são controlados por um ou mais microprocessadores. Desta forma, pode-se considerar os utensílios como sendo mais inteligentes e complexos do que os controles inteligentes.

Os utensílios inteligentes, em ambientes de computação ubíqua, interagem entre si para aumentar o conforto dos usuários, seja em casa, no carro, no escritório, no banco, no hospital, nas ruas, nos *shoppings*, etc [de Araujo (2003)]. Exemplos de aplicação de utensílios inteligentes incluem:

- Centrais de telecomunicação;
- Utensílios da linha branca, como geladeiras e máquinas de lavar, dotados de capacidade computacional, que se adaptam as condições do ambiente e reagem de acordo com as circunstâncias;
- Dispositivos computacionais integrados a automóveis disponibilizando para os usuários serviços como sistemas de navegação, informações sobre falhas em componentes do automóvel, monitoramento dos dados sobre os veículos pelas montadoras, acesso à informação e entretenimento;
- Instrumentos de monitoramento médico;
- Máquinas de venda automática.

### 2.2.3 Sistemas de Entretenimento

Os dispositivos ubíquos pertencentes a este grupo são aqueles cujo uso é voltado principalmente para o lazer. Exemplos incluem reprodutores de música (MP3/MP4), TVs, *Set-top-boxes* (caixa sobre a TV que promove a interface entre os provedores do serviço de difusão e a TV do consumidor), consoles de jogos eletrônicos (*Playstation* da *Sony*, *X-box* da *Microsoft*), câmeras digitais, brinquedos inteligentes, dentre outros.

### 2.2.4 Dispositivos de Acesso à Informação

Os dispositivos de acesso à informação, como o próprio nome indica, possuem como principal objetivo disponibilizar para os usuários serviços de comunicação e acesso à informação. Este grupo de dispositivos é o mais relevante para esta dissertação, pois o *framework* proposto se refere à sincronização de dados, que é um processo aplicável a esta classe de dispositivos ubíquos.

A seguir são apresentadas algumas das principais categorias de dispositivos de acesso à informação disponíveis no mercado.

#### Assistentes pessoais digitais

Um dos tipos de dispositivos de acesso a dados mais conhecidos são os assistentes pessoais digitais, do inglês *Personal Data Assistant*, ou PDAs. Eles caracterizam-se pela tela sensível ao toque. A interação com o dispositivo é baseada em apontador, ou *stylus*, e a entrada de texto normalmente é feita por reconhecimento de escrita manual ou teclado virtual.

A empresa que popularizou estes dispositivos se chamava *Palm* ([www.palmone.com](http://www.palmone.com)) e, por algum tempo, o seu nome foi amplamente conhecido como sinônimo dos aparelhos.

Os serviços oferecidos incluem agenda de endereços e telefones, calendário, atividades a fazer, editor de texto, calculadora, despertador, e-mail, jogos, navegadores, relógios globais, planilhas, aplicações financeiras, reprodutores de mídia, recursos de desenho e outros.

Os PDAs vêm perdendo espaço para os telefones inteligentes que, além de oferecerem as mesmas funcionalidades que os PDAs, possuem telefone celular integrado. A figura 2.4 exhibe alguns exemplos de PDAs.



Figura 2.4: Exemplos de PDAs

### Coletores de dados

Outra classe de dispositivos de acesso à informação é a dos coletores de dados. Eles, normalmente, são maiores e mais robustos do que os PDAs, sendo utilizados em aplicações corporativas, como serviços de leitura de medidores de energia, leitura de hidrômetros de água e controle de almoxarifados. As características dos coletores de dados podem variar muito de um modelo para outro, sendo que alguns possuem câmara digital, GPS e celular integrados. A figura 2.5 exhibe alguns exemplos de coletores de dados.



Figura 2.5: Exemplos de coletores de dados

### Telefones inteligentes

Os telefones inteligentes, ou *smartphones*, integram em um só equipamento todas as funcionalidades de um PDA e de um telefone celular. Hoje eles são a classe de dispositivos de acesso a dados mais vendidos, ganhando o mercado que antes pertencia aos PDAs. A figura 2.6 exhibe alguns exemplos de telefones inteligentes.



Figura 2.6: Exemplos de telefones inteligentes

### *Tablet PCs*

Um *tablet PC* se parece com *notebook* pequeno, com a tela sensível ao toque. O nome *tablet PC* advém do fato da tela sensível ao toque “girar” para o lado de fora do equipamento, de modo que ele se transforme em uma espécie de prancheta. Com dimensões similares às de um caderno universitário, a tela possibilita o reconhecimento da escrita do usuário com o auxílio de canetas do tipo *stylus*, semelhante a um PDA.

Estes equipamentos possuem capacidade de processamento e configuração semelhantes às de um computador *desktop*, possuindo mais recursos computacionais que as outras classes de dispositivos ubíquos de acesso à informação.

A maior parte dos *tablet PCs* utilizam os mesmos sistemas operacionais utilizados em computadores de mesa, acrescidos de alguns recursos particulares, o que permite ao usuário executar no equipamento os mesmos programas utilizados em computadores normais. A figura 2.7 exibe um exemplo de um *tablet PC*.

Figura 2.7: Exemplo de *tabletPC*

## 2.3 Sistemas Operacionais

Nesta Seção são apresentados os sistemas operacionais mais comuns utilizados por dispositivos ubíquos, em especial os destinados aos dispositivos de acesso à informação.

### 2.3.1 Symbian OS

*Symbian* (Symbian (2008)) é um avançado sistema operacional de padrão aberto para telefones celulares. Possui um núcleo multi-tarefas e *multithread*, um *framework* para interface gráfica, componentes para serviços de transferência e processamento de dados, *engines* para aplicações e gerenciamento de dados pessoais e comunicação sem fio.

O *Symbian OS* é desenvolvido exclusivamente para dispositivos móveis por fabricantes de celulares como *Nokia*, *Siemens*, *Samsung* e *Motorola*, que se uniram para formar a *joint venture* denominada *Symbian*. A meta era desenvolver um sistema operacional para *smartphones* e PDAs baseado no *Psion's EPOC operating system*. O *Symbian* suporta vários protocolos de comunicação, tais como TCP/IP, WAP, GSM e *Bluetooth*, e permite que as aplicações sejam escritas em linguagens como C++ e Java.

Segundo Canalys (2008), o *Symbian* possui a maior participação mundial nas vendas de *smartphones*, atingindo 65% de *Market Share* em 2007, mas está atrás de outras companhias no mercado Norte Americano, que é relativamente pequeno, mas com grande visibilidade.

O *Symbian* é um sistema totalmente modular, permitindo que cada empresa crie sua própria interface. Portanto, este sistema não tem uma interface definida, podendo ser um simples sistema de textos em telas monocromáticas ou um completo sistema operacional tão potente quanto o *PalmOS* ou *Windows Mobile*. O *Symbian OS* pode ser encontrado nos *smartphones* da *Nokia*, *SonyEricsson*, *Siemens*, *Motorola*, dentre outras. A figura 2.8, extraída de Symbian Brasil (2008), exhibe exemplos da interface *Symbian OS*.



Figura 2.8: Exemplos de interfaces Symbian OS

### 2.3.2 Windows CE

O *Microsoft Windows CE* (Microsoft (2008a)) é um sistema operacional embutido que pode ser utilizado em dispositivos com capacidade computacional limitada, como robôs, contro-

ladores industriais, bombas de gás, urnas eletrônicas, máquinas de venda, terminais POS, consoles para entretenimento e multimídia, equipamentos médicos, televisores interativos, câmeras digitais, etc.

Com a utilização da ferramenta *Platform Builder*, também desenvolvida pela *Microsoft*, fabricantes de dispositivos podem personalizar o sistema operacional de forma que ele melhor se adapte as características de cada equipamento. Esta funcionalidade é essencial para permitir que ele seja utilizado em dispositivos tão diferentes como os citados anteriormente.

O *Windows CE* é um sistema operacional compacto, podendo ser executado em apenas um *megabyte* de memória. Ele também é compatível com a definição de sistema operacional de tempo real, possuindo uma latência de interrupção determinística.

Um telefone celular baseado no *Windows CE*, dependendo dos componentes do sistema operacional utilizados, pode ser extremamente simples, com apenas as funções de voz e, talvez, SMS, ou bastante poderoso, possuindo todas as funcionalidades comuns a um *smartphone*.

O desenvolvimento de aplicações para este sistema operacional, em um primeiro momento, só podia ser feito em C e C++, por meio de um subconjunto da *Win 32 API*, seguindo o mesmo estilo de programação das aplicações nativas para os sistemas operacionais para computadores pessoais desenvolvidos pela *Microsoft*.

Com a evolução do *Windows CE*, este foi ganhando poder de processamento e estabilidade, e novos componentes e funcionalidades foram incluídas, como a possibilidade de desenvolvimento de aplicações baseadas no modelo de componentes COM. Atualmente o desenvolvimento de aplicações para este sistema também pode ser feito com o *.NET Compact Framework* (*.NET Compact Framework* (2008)).

Muitas plataformas da *Microsoft* foram baseadas no núcleo do *Windows CE*, incluindo o *Pocket PC 2000*, *Pocket PC 2002*, *Windows Mobile 2003*, *Windows Mobile 2003 SE*, *Windows Mobile 5.0*, *Windows Mobile 6* e outras.

### ***.NET Compact Framework***

O *.NET Compact Framework* (CF) (*.NET Compact Framework* (2008)) é uma versão do *.NET Framework* projetada para ser executada em dispositivos cujo sistema operacional seja baseado no *Windows CE*, como PDAs, telefones inteligentes, controladores industriais e outros. Por ser um subconjunto do *.NET framework*, o *.NET CF* herdou muitas de suas características e formas de execução.

É possível desenvolver aplicações com o *.NET Compact Framework* utilizando o Visual Studio (VS) *.NET*, que é um ambiente de desenvolvimento fornecido pela *Microsoft*, em linguagens como C# ou Visual Basic *.NET*.

Para ser capaz de executar aplicações construídas com o *.NET Compact Framework*, a plataforma precisa suportar o *runtime* do *.NET CF*, o que é possível para a maioria dos sistemas operacionais da família *Windows*. Aplicações desenvolvidas em *.NET CF* também podem ser executadas em computadores de mesa com a versão completa do *.NET Framework*, desde que estas aplicações acessem apenas as partes compartilhadas entre os dois *frameworks*.

### 2.3.3 *Windows Mobile*

O *Windows Mobile* (Microsoft (2008b)), da *Microsoft*, é um sistema operacional baseado no *Windows CE*, porém com alguns componentes padronizados, de forma que, enquanto dois equipamentos que utilizam o *Windows CE* podem ser extremamente diferentes entre si, os dispositivos baseados no *Windows Mobile* possuem basicamente as mesmas funcionalidades do ponto de vista do sistema operacional.

O *Windows Mobile* é um sistema operacional compacto, desenvolvido para rodar em dispositivos móveis como *PDA*s, *smartphones* e aparelhos de multimídia em geral. Projetado para ser capaz de realizar boa parte do que é possível em uma versão do *Windows* para PCs de mesa, o sistema vem com um conjunto de aplicações básicas bem conhecidas, tais como o *Word*, *Excel*, *PowerPoint* e *Windows Media Player Pocket*.

Existem várias empresas que manufaturam dispositivos com o sistema *Windows Mobile*, como, por exemplo *HTC*, *Qtek*, *i-mate*, *BenQ*, *HP*, *Siemens*, *Eten*, *Sony Ericksson*, dentre outras.

### 2.3.4 *BlackBerry OS*

É o sistema operacional proprietário utilizado nos *smartphones BlackBerry*, fabricados pela empresa canadense *Research in Motion (RIM)* (Black Berry Developers (2008)). Segundo Canalys (2008), este sistema operacional obteve 11% do mercado mundial de sistemas operacionais para *smartphones* no ano de 2007. A principal característica do *BlackBerry* é permitir que seus usuários possam receber e-mails sempre que o dispositivo estiver conectado a uma rede de dados.

O *BlackBerry OS* é focado na facilidade de operação e foi originalmente projetado para o mercado corporativo. Recentemente foi liberado o SDK, do inglês *Software Development Kit*, que permite que desenvolvedores implementem aplicações para este sistema operacional.

### 2.3.5 *Linux Embarcado*

Segundo de Araujo (2003), o *Linux* embarcado é na realidade um *Linux* simplificado com suporte especial para dispositivos ubíquos. Este sistema operacional apresenta as seguintes características:

- Arquitetura *microkernel* - serviços e características podem ser compiladas no núcleo ou carregadas como módulos dinamicamente ligados em tempo de execução;
- Suporte multiusuário;
- Sistema multitarefa, preemptivo, com escalonadores de tempo-real opcionais;
- Suporte básico para múltiplos processadores;
- Tamanho do núcleo pode variar de 200KB a vários MB;
- Suporta memória virtual com paginação;

- Suporta versões “enxutas” do *X-Window* para interface com o usuário;
- Possui versões para diversos processadores, como MIPS, ARM, Motorola e Intel.

### 2.3.6 *iPhone OS*

O *iPhone OS* (Apple (2008)), da *Apple Inc.*, é derivado do sistema operacional *Mac OS X*, que equipa os computadores *Macintosh*, porém com algumas adaptações, como interface *multi touch* e suporte para acelerômetro. Ele é utilizado nos dispositivos de tela sensível ao toque *iPhone* e *iPode*. A *Apple* entrou no mercado de *smartphones* em 2007, com o *iPhone* sendo disponibilizado em apenas alguns países e, segundo Canalys (2008), mesmo assim obteve, em um ano, 7% do mercado mundial e 28% do mercado Norte Americano.

Uma das grandes inovações do *iPhone* é a tela sensível ao toque especialmente criada para ser usada com um dedo, ou múltiplos dedos, para sensibilidade *multi touch*. Não é necessária uma caneta *stylus*, e nenhuma pode ser usada, visto que o sistema é sensível apenas ao calor emitido pela pele humana. Por esse motivo, as vezes pode ser incômodo manusear o aparelho, como em regiões frias em que os usuários terão que tirar suas luvas para usar o *touchpad*.

O *multi touch*, diferente do *touch screen* disponível nos PDAs e *smartphones* convencionais, permite não só executar comandos tocando na tela, mas ir além, rolando listas com o deslizar do dedo e ampliando e reduzindo imagens com o movimento de pinça do polegar e do indicador. A tela é capaz de reconhecer toques simultâneos, possibilitando ao telefone reconhecer gestos e oferecer uma interação mais rica, permitindo que o usuário “manipule” os objetos.

A tela responde a três sensores: um primeiro sensor de presença que desliga a tela e o *touch screen* quando o *iPhone* é aproximado do rosto, para economizar bateria e prevenir toques indesejados na tela por meio da orelha ou rosto do usuário. Um segundo sensor de luz ambiente controla a luminosidade do visor para economizar bateria, e um acelerômetro que sente a orientação do telefone e muda o conteúdo da tela de acordo. A figura 2.9 exibe um exemplar do *iPhone*.

Recentemente a *Apple* disponibilizou um SDK que permite que desenvolvedores implementem aplicações para o sistema *iPhone OS* utilizando a linguagem C e *Objective-C*, que é uma extensão do padrão ANSI C.

### 2.3.7 *Palm OS*

O *Palm OS*, também conhecido como *Garnet OS* (Access Developer Network (2008)), é um sistema operacional embutido inicialmente desenvolvido pela *U.S. Robotics*, então proprietária da *Palm Computing*, para ser utilizado em PDAs no ano de 1996.

O *PalmOS* foi projetado para ser fácil de ser utilizado por meio de uma interface gráfica sensível ao toque, do tipo *touch screen*. Ele é disponibilizado com um conjunto de aplicações básicas, como agenda de telefones, calendário, calculadora e outras.

Além da *Palm*, outras empresas, como a *Handspring* e *Symbol*, já manufacturam dispositivos com o sistema operacional *PalmOS*. A versão atualmente licenciada pela empresa



Figura 2.9: iPhone da Apple

*ACCESS* é chamada *Garnet OS*, pois a marca *Palm* pertence à empresa Palm Inc. As principais características do *Garnet OS* são:

- Simplicidade, com um ambiente mono-tarefa;
- Possui um sistema de reconhecimento de escrita;
- Suporte para acesso a redes TCP/IP;
- Suporte para conexões USB, infravermelho, *bluetooth* e *Wi-Fi*;
- Suporte para cartões de expansão de memória.

O *PalmOS* foi um dos responsáveis pela popularização dos PDAs no final da década de 90, quando obteve liderança absoluta no mercado de computadores de mão. Contudo, nos últimos anos, o sistema operacional vem enfrentando forte concorrência, perdendo uma grande parte do seu mercado, principalmente para o sistema operacional *Windows Mobile*, da *Microsoft*.

## 2.4 Conclusões

Este Capítulo tratou das questões relacionadas à computação ubíqua necessárias para compreensão do trabalho proposto. Primeiro foram definidos a terminologia, conceitos e princípios da computação ubíqua, móvel e pervasiva. Foi visto que o mSync vai ao encontro dos princípios de simplicidade, diversidade, descentralização e conectividade da computação ubíqua. Em seguida, foi apresentada uma classificação que divide os dispositivos ubíquos em quatro grandes grupos: controles inteligentes, utensílios inteligentes, sistema de entretenimento e dispositivos de acesso à informação. A classificação é útil para simplificar o estudo, mas não deve ser considerada absoluta, pois, em muitas situações, um dispositivo pode possuir características pertencentes a dois ou mais grupos. Foi dada ênfase especial ao grupo de dispositivos de acesso a dados, pois estes representam o grupo mais relevante para o trabalho

---

proposto. Depois foi apresentada uma revisão dos principais sistemas operacionais utilizados em dispositivos ubíquos de acesso a dados. O próximo Capítulo descreve os conceitos e trabalhos relacionados à replicação e sincronização de dados.

## Capítulo 3

# Replicação e Sincronização de Dados

O uso de dispositivos ubíquos de acesso a dados tem crescido em ritmo acelerado. Uma importante comodidade oferecida por estes dispositivos é a conveniência de permitir que os usuários transportem cópias das informações necessárias ao longo do dia e as acessem aonde quer que eles estejam. Para permitir esta funcionalidade é necessário que os dados sejam replicados e depois sincronizados entre os dispositivos móveis e as fontes de dados que residem em servidores fixos.

Segundo Saito e Shapiro (2005), a replicação de dados consiste em manter múltiplas cópias de dados, chamadas réplicas, em computadores separados. O processo responsável por manter estas várias cópias consistentes entre si é conhecido como sincronização de dados. As técnicas de replicação e sincronização de dados são fundamentais em ambientes ubíquos, pois permitem que dispositivos móveis acessem e modifiquem dados mesmo quando desconectados.

Em geral, informações em servidores corporativos são acessadas somente sob demanda enquanto os usuários estão conectados. Entretanto, devido às desconexões freqüentes em redes sem fio, um cliente móvel pode ficar desconectado do servidor por longos períodos de tempo. Realizar uma cópia local dos dados potencialmente úteis para execuções futuras, antecipando-se a desconexão, é o único modo para suportar operações móveis desconectadas. Um cliente móvel pode executar uma transação utilizando os dados armazenados localmente enquanto estiver sem conexão. A funcionalidade chave deste novo paradigma da computação é a habilidade de armazenar dados localmente nos clientes móveis e sincronizar as modificações de volta para o servidor corporativo. Este recurso desempenha um papel essencial na visão da computação pervasiva, onde as informações estão disponíveis a todo momento e em qualquer lugar (Choi et al. (2005)).

O texto do Capítulo está assim organizado. A Seção 3.1 trata das questões relacionadas à consistência de dados em bancos de dados móveis. A Seção 3.2 introduz o conceito de conflitos em sincronizações de dados. A Seção 3.3 apresenta os protocolos utilizados para intercâmbio de dados entre os dispositivos e o servidor durante o processo de sincronização. A Seção 3.4 introduz o conceito de âncoras de sincronização. A Seção 3.5 apresenta as técnicas

relacionadas ao particionamento de dados para trabalho em modo desconectado. A Seção 3.6 trata do rastreamento de modificações nas bases de dados móveis e fixa. A Seção 3.7 discute os problemas relacionados à geração de identificadores únicos de registros em um ambiente distribuído. A Seção 3.8 apresenta os principais bancos de dados móveis comerciais existentes no mercado. A Seção 3.9 apresenta os principais trabalhos relacionados a esta dissertação e, por fim, a Seção 3.10 encerra o Capítulo com os comentários e observações finais.

### 3.1 Consistência em bancos de dados móveis

Segundo Gollmick (2003), um ambiente cliente / servidor de banco de dados móvel é constituído por um servidor de banco de dados central, residindo em um local fixo, e um ou mais bancos de dados locais armazenados em clientes móveis, conforme exibido na figura 3.1.

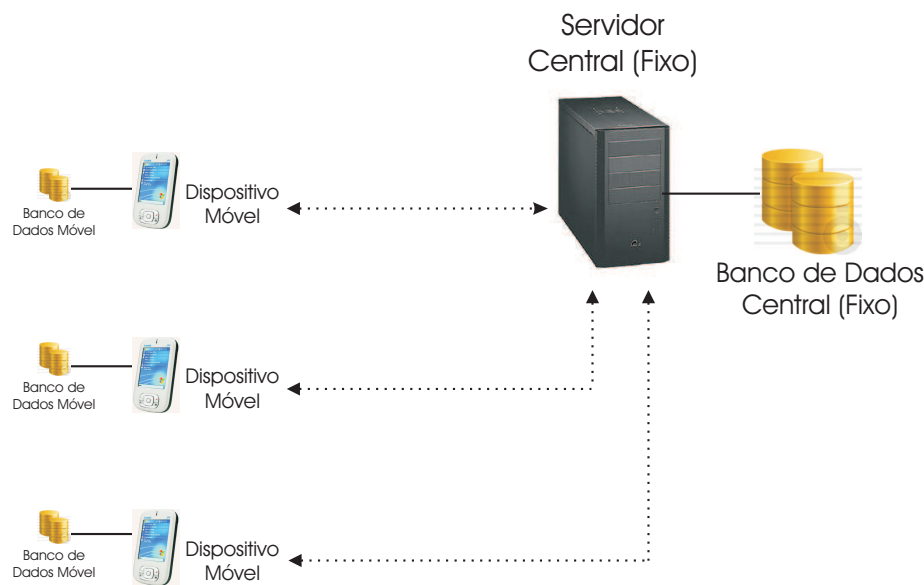


Figura 3.1: Ambiente cliente / servidor de banco de dados móvel

Uma vez que os dispositivos ubíquos de acesso a dados estão sujeitos a desconexão de rede uma grande parte do tempo, as técnicas tradicionais de transações em bancos de dados não podem ser aplicadas para manter as informações consistentes entre os bancos de dados móveis, armazenados nos dispositivos, e o banco de dados central, armazenado em um servidor fixo. Muitos trabalhos discutem novas técnicas de transações aplicáveis a estes ambientes.

Davidson et al. (1984) classifica as estratégias de consistência em bancos de dados em pessimistas e otimistas. A estratégia pessimista evita a ocorrência de conflitos por meio da limitação da disponibilidade das informações. Nesta abordagem, cada partição trabalha assumindo o pior cenário, fazendo a consideração pessimista de que se um conflito pode ocorrer ele vai ocorrer.

A estratégia otimista não limita a disponibilidade de informações, permitindo que as réplicas sejam modificadas independentemente, detectando e resolvendo os possíveis conflitos resultantes. Na literatura existe uma clara tendência em indicar que a abordagem otimista é

mais apropriada do que a pessimista para uma rede constituída principalmente por dispositivos móveis (Cohen (2000); Leonard Kawell et al. (1988); Lu e Satyanarayanan (1994); Petersen et al. (1996); Wang et al. (1999); Terry et al. (1995)).

Um axioma comum referente a consistência de um sistema de banco de dados é que as transações devem ser serializáveis. A serialização é uma propriedade que garante que, independente da ordem em que os dados são acessados dentro de uma transação, o resultado final será sempre o mesmo. Fischer e Michael (1982) observa que existe um conflito inerente entre a propriedade de serialização e a disponibilidade de dados em ambientes distribuídos. Porém, o trabalho afirma que a disponibilidade é a principal razão para a utilização da replicação de dados e que, em muitas aplicações, como apontamentos de agenda, e-mails e sistemas de arquivos distribuídos, a disponibilidade é mais importante do que a serialidade.

Segundo Butrico et al. (2000), modelos de consistência de dados tradicionais não são práticos para uso em ambientes distribuídos, pois eles exigem comunicação com outras máquinas até mesmo para acesso aos dados locais. A arquitetura mSync, proposta neste trabalho, relaxa os modelos de consistência, utilizando uma abordagem otimista, sacrifica algumas propriedades ACID (Serrano-Alvarado et al. (2004)) em troca da disponibilidade de dados. Só existe garantia que um banco de dados móvel estará consistente com o banco de dados central depois da execução do processo de sincronização e antes que alguma aplicação modifique uma das duas bases de dados. Modificações efetuadas nos dados e interrupções no processo de sincronização devido a falhas de comunicação, podem levar os dados para um estado de inconsistência temporária até a próxima sincronização.

Muitas aplicações encontradas no mundo real, como soluções de automação comercial, fiscalização, vistorias, levantamento de dados e outras, não exigem uma garantida rígida de consistência de dados, sendo aceitável que as informações disponíveis nos bancos de dados móveis, armazenados nos dispositivos, fiquem temporariamente inconsistentes com o banco de dados central, armazenado em um servidor fixo, em troca de uma maior disponibilidade de dados. A arquitetura mSync se baseia nesta característica para propor uma solução de sincronização de dados fim a fim que permite uma grande disponibilidade dos dados, tanto para consulta quanto para modificações e, ao mesmo tempo, mantém uma estrutura simples e eficiente.

## 3.2 Conflitos

Mesmo com os avanços em *hardware* e tecnologia de comunicação, a conectividade em dispositivos móveis é intermitente devido a fatores físicos, econômicos e energéticos. Esta característica torna interessante o suporte para operações independentes, isto é, os usuários precisam conseguir acessar e modificar um banco de dados compartilhado mesmo sem nenhum tipo de conexão remota. Mesmo com conectividade disponível, operações desconectadas podem ser utilizadas para superar problemas relativos à latência e largura de banda (Preguiça et al. (2003); Tolia et al. (2007)).

A replicação otimista tem sido utilizada para suportar operações em modo desconectado

(Wang et al. (1999)). Nesta abordagem, os dispositivos móveis mantêm uma cópia local dos dados que podem ser modificados independentemente. Em algum momento, as modificações realizadas devem ser aplicadas no banco de dados central e os possíveis conflitos gerados por modificações concorrentes resolvidos. Esta abordagem aumenta a disponibilidade dos dados mas, em contrapartida, pode comprometer a consistência das informações.

As transações podem produzir resultados incorretos, por exemplo, em uma aplicação de venda de passagens aéreas onde são reservados mais assentos do que os disponíveis fisicamente. Em muitas aplicações estes resultados incorretos são aceitáveis para se obter uma maior disponibilidade dos dados. Quando o dispositivo é reconectado a rede, os conflitos devem ser detectados e resolvidos. Caso os conflitos não tenham efeito no mundo real, eles podem ser resolvidos através de métodos padrões de recuperação de bancos de dados, como, por exemplo, desfazendo-se algumas operações ou definindo-se uma regra de prioridade para determinar qual informação será válida. Mas se, em contrapartida, o conflito gerou efeitos no mundo real, devem ser executadas operações de compensação onde não só a consistência do banco de dados é restaurada, mas os efeitos sob o mundo real sejam anulados como, por exemplo, cancelando certas reservas de passagem e enviando uma mensagem para os passageiros afetados.

Um registro está em conflito quando ele é alterado por mais de um nó entre sincronizações. As aplicações devem ser projetadas para minimizar a ocorrência de conflitos, pois sua detecção e resolução introduzem complexidade, processamento e tráfego de rede adicionais. Uma das formas de se evitar conflitos é garantir que apenas um nó modifique um determinado registro de cada vez, o que pode ser feito em algumas aplicações, dividindo-se as informações carregadas em cada dispositivo de modo que um mesmo registro não esteja disponível no banco de dados local de mais de um dispositivo ao mesmo tempo. Um exemplo dessa estratégia pode ser encontrada em aplicações de automação de força vendas onde cada vendedor é responsável por uma determinada região e, desta forma, são carregados localmente em seu dispositivo móvel apenas as informações referentes aos clientes da sua região.

A arquitetura mSync oferece suporte para detecção e resolução de conflitos. Uma vez identificados, os conflitos podem ser resolvidos por meio de regras pré-definidas como, por exemplo, mantendo-se válida a informação mais recente ou, caso seja necessário, definindo-se políticas que levem em consideração as regras de negócio de uma aplicação específica. A forma como os erros e conflitos são tratados na arquitetura é apresentada na Seção 4.2.

### 3.3 Protocolos para troca de dados entre os dispositivos e o servidor

Um dos pilares da computação pervasiva é que diversos dispositivos ubíquos, com diferentes características, devem se comunicar para oferecer serviços aos usuários. Neste contexto, é importante que o protocolo utilizado para realizar a troca de dados entre os dispositivos seja suportado por diferentes plataformas. O XML (W3C (2008a)) possui esta característica e, segundo Kangasharju et al. (2007), ele tem se apresentado como uma tendência para comunicação entre aplicações. Vários trabalhos utilizam protocolos baseados no XML para inter-

câmbio de informações entre dispositivos móveis e um servidor central (Butrico et al. (2000); Dong e Bingjun (2005); Lam et al. (2002); Miller et al. (2004); Lindholm (2003); Lam et al. (2002)).

O XML é um padrão aberto, que provê uma estrutura de dados padronizada que pode ser alterada e adaptada para suportar a maior parte dos tipos de arquivos e aplicativos. Um dos benefícios do XML é o fato dele ser independente da camada de transporte, podendo ser utilizado sobre diversos protocolos como HTTP ou TCP/IP. O XML também é independente de plataforma, podendo ser utilizado em dispositivos com diferentes características (W3C (2008a)).

Mesmo sendo muito utilizado, o XML possui características que o tornam ineficiente em um ambiente de computação móvel. Os principais problemas estão relacionados ao tamanho das mensagens e ao tempo gasto para processá-las (Tarkoma et al. (2006)). Muitos esforços têm sido despendidos com o objetivo de se encontrar soluções que tornem o XML mais apropriado para o ambiente de computação ubíqua.

Técnicas de compressão de dados podem ser utilizadas para diminuir o tamanho dos arquivos XML, mas esta abordagem nem sempre é viável, uma vez que ela aumenta o tempo de processamento e não funciona bem para mensagens pequenas (Kangasharju et al. (2003)). Butrico et al. (2000) recomenda que sejam utilizados nomes curtos para os *tags* e atributos, pois este artifício ajuda a diminuir o tamanho das mensagens.

Outra limitação imposta pelo XML é o fato dele não transmitir dados binários. Caso seja necessário, este tipo de informação pode ser transmitido como texto codificado, por exemplo utilizando a base 64, porém esta alternativa faz com que o volume de dados transmitidos seja maior do que o necessário caso a informação fosse transmitida diretamente no formato binário. Uma outra possibilidade para transmitir dados binários é utilizar uma extensão do WBXML (W3C (2008b)).

O *WAP Binary XML* (WBXML W3C (2008b)) é uma representação binária do XML, desenvolvida pela *Open Mobile Alliance*, como um padrão para permitir que documentos XML sejam transmitidos de uma maneira compacta em ambientes de computação móvel. Basicamente, o WBXML utiliza uma tabela de nomes (*tokens*) para converter nós e atributos em valores de um *byte* e, com isto, evita que um grande volume de informações desnecessárias sejam transmitidas.

Muitos trabalhos utilizam *Web services* (Web Services Activity (2008)) como suporte para a integração entre os dispositivos móveis e os servidores fixos (Oh e Fox (2005); Oh et al. (2005)). SOAP (SOAP Specifications (2008)), do inglês *Simple Object Access Protocol*, é o protocolo utilizado pelos *Web services*. Sendo baseado em XML, o SOAP compartilha muitas das vantagens e problemas já citados para o XML.

O *framework* mSync utiliza *Web services* como meio para troca de informações entre os dispositivos móveis e o servidor central. Este padrão foi escolhido por ser independente da plataforma, permitindo que o *framework* evolua para suportar diferentes arquiteturas de dispositivos e servidores. Outra característica que contribuiu para a escolha deste padrão foi o fato de, por poder ser utilizado sobre o protocolo HTTP, largamente utilizado na Internet,

as mensagens SOAP normalmente não serem bloqueadas em *firewalls* corporativos. Além disto, as principais ferramentas de desenvolvimento disponíveis no mercado, como Eclipse e Visual Studio .NET, oferecem suporte para utilização e geração de *Web services*, o que facilita a utilização do padrão. Uma sugestão de trabalho futuro é a pesquisa e adaptação do *framework* para suportar o intercâmbio de dados entre os dispositivos e o servidor utilizando um padrão mais otimizado para o ambiente de redes ubíquas, como, por exemplo, XML Binário (WBXML).

### 3.4 Âncoras de sincronização

O termo âncora de sincronização é utilizado neste trabalho para representar uma marca que o dispositivo armazena no fim de cada sessão de sincronização bem sucedida. Este termo não é novo na literatura, tendo sido utilizado em alguns trabalhos, como no protocolo SyncML (SyncML Protocol (2001)) e na arquitetura *Sync Service*, da *Microsoft* (MicrosoftSyncServices (2008)).

O protocolo SyncML utiliza esse termo para designar uma *string* que representa um evento de sincronização. Ele utiliza duas âncoras, *Last* e *Next*, que são enviadas no início do processo de sincronização. A âncora *last* descreve o último evento em que o banco de dados foi sincronizado, e a âncora *next* descreve o evento de sincronização atual, ambos do ponto de vista do dispositivo que está transmitindo os dados. O servidor precisa armazenar cada âncora *next* recebida e, no início do processo, verificar se o valor da âncora *last* transmitida pelo dispositivo é igual à última âncora *next* armazenada. Caso os valores não sejam iguais, significa que o último processo de sincronização não foi concluído com sucesso e o servidor pode requisitar uma função especial como, por exemplo, realizar uma sincronização lenta, onde todos os registros do banco de dados do cliente são transmitidos para o servidor.

Na arquitetura *Sync Service*, da *Microsoft*, o significado do termo âncora de sincronização é semelhante ao utilizado neste trabalho. O cliente armazena um novo valor de âncora no fim de cada sessão de sincronização. No início do processo, o servidor gera um novo valor de âncora e apenas as modificações realizadas entre a velha e nova âncora são listadas e enviadas para o cliente.

No mSync, proposto nesta dissertação, semelhante ao *Sync Service*, as âncoras são armazenadas pelos dispositivos no final de cada sessão de sincronização e, na próxima sincronização, esta marca é enviada para o servidor para que ele possa determinar quais informações já foram enviadas para o cliente no último processo de sincronização, conforme será detalhado na Seção 4.1. Além disto, diferente das outras abordagens, o mSync também utiliza as âncoras no processo conversão dos identificadores únicos locais (LUID), gerados nos dispositivos quando um novo registro é inserido no banco de dados local, em identificadores únicos globais (GUID), gerados no banco de dados central. O método de mapeamento de identificadores de registros proposto neste trabalho possui vantagens em relação às abordagens tradicionais, conforme será apresentado na Seção 4.4.

### 3.5 Carga inicial e particionamento de dados para execução em modo desconectado

Em muitas aplicações os usuários precisam ter acesso a apenas um subconjunto das informações disponíveis no servidor, exigindo que apenas uma porção dos dados sejam replicados nos dispositivos. Um exemplo são aplicações de automação de força de vendas, onde cada vendedor é responsável por uma determinada região e necessita acessar somente as informações referentes aos clientes sob sua responsabilidade. Ainda é uma área aberta de pesquisa, uma forma segura de determinar se os dados armazenados localmente são suficientes para atender todas as requisições durante a desconexão (Castro et al. (2004)). Alguns trabalhos, como Helal et al. (2001), utilizam algoritmos de predição para definir quais informações devem ser replicadas. O principal problema destas abordagens é que as predições podem estar erradas. O *framework* mSync optou por uma solução menos geral que trata a carga inicial de dados como sendo parte da lógica de negócio, oferecendo suporte para que o desenvolvedor, baseado nas características da solução desenvolvida, defina a regra utilizada para definir quais informações devem ser disponibilizadas no banco de dados local dos dispositivos.

Com o particionamento e armazenamento local de apenas um subconjunto das informações disponíveis no servidor, é possível obter os seguintes benefícios:

- Redução do volume de dados transmitidos através da rede;
- Redução da quantidade de espaço em disco necessária para armazenar o banco de dados local;
- Redução ou até mesmo eliminação da possibilidade de ocorrência de conflitos, uma vez que diferentes partições do banco de dados central podem ser enviadas para os dispositivos, evitando que mais de um dispositivo modifique o mesmo conjunto de registros;
- Diminuição da quantidade de processamento necessário no dispositivo, devido à redução da quantidade de informações armazenadas no banco de dados local.

O particionamento de dados pode ser tanto horizontal, reduzindo o número de linhas enviadas para os dispositivos, quanto vertical, diminuindo a quantidade de colunas, de modo que apenas os campos necessários sejam armazenados na memória local do dispositivo. O mSync permite que as regras utilizadas para carga inicial e particionamento de dados sejam implementadas sobrescrevendo-se as funções pertinentes da classe *ServerSyncDAO*, conforme será detalhado nas Seções 4.5.11 e 5.2.3.3.

### 3.6 Rastreamento de modificações

Uma tarefa importante dentro do processo de sincronização de dados é identificar quais informações foram modificadas durante o período de desconexão. Esta atividade permite a realização de uma sincronização eficiente, onde apenas os registros modificados são sincronizados entre os dispositivos e o servidor. Petersen et al. (1996); Castro e Pakkala (2007)

utilizam técnicas baseadas em vetores para determinar quais informações devem ser sincronizadas entre duas réplicas. Nesta abordagem, cada dispositivo armazena um conjunto de vetores associado a cada réplica ativa, o que faz com que o volume de dados armazenados localmente seja proporcional ao número de réplicas, o que não é escalável para aplicações utilizadas por uma grande quantidade de dispositivos. Em contrapartida, esta abordagem permite que a sincronização seja realizada entre dois dispositivos quaisquer, em uma topologia ponto a ponto, onde qualquer dispositivo pode atuar ora como cliente, ora com servidor.

O mSync é aplicável em cenários onde vários dispositivos sincronizam dados com um único servidor, em uma topologia estrela, conforme apresentado na figura 3.1. Aproveitando-se desta característica, é utilizado um esquema de rastreamento de modificações onde, nos dispositivos, é usado um *flag* para determinar o estado de cada registro, que pode ser *Novo*, *Atualizado*, *Sincronizado* ou *Excluído*. Já do lado do servidor, as modificações são rastreadas com o uso das âncoras de sincronização. Sempre que um registro é inserido ou modificado no banco de dados central, o valor da âncora associada a este registro é atualizado.

As exclusões de registros são rastreadas por meio de uma tabela especial chamada *TombStone*. Outros trabalhos, como MicrosoftSyncServices (2008), também utilizam uma tabela semelhante para este fim. A forma como o rastreamento de modificações, tanto nos dispositivos quanto no servidor, é tratada pela arquitetura proposta é detalhada na Seção 4.3.

### 3.7 Geração de identificadores únicos em um ambiente distribuído

A necessidade de se identificar unicamente cada registro é uma característica básica presente em muitos sistemas de informação (Blaha et al. (1988)). Um dos grandes desafios introduzidos pelo caráter distribuído das soluções de replicação e sincronização de dados está na geração de identificadores únicos locais, criados de forma independente nos dispositivos, que não sejam coincidentes com identificadores únicos criados no servidor ou em outros dispositivos, evitando um problema conhecido como colisão de chave primária (Oracle FAQ's (2008)). Considere o seguinte exemplo:

1. Dispositivo A, executando uma aplicação de automação de força de vendas, cadastra um novo pedido cujo código de identificação possui o valor 1004. Dispositivo A sincroniza e o pedido é aplicado no servidor com o identificador único 1004.
2. Dispositivo B, executando a mesma aplicação de automação de força de vendas, cadastra outro pedido cujo código de identificação único também é 1004. Como o dispositivo B não sincronizou com o servidor depois que o dispositivo A enviou o pedido, ele não sabe que o número 1004 já foi utilizado.
3. Dispositivo B sincroniza com servidor. Neste momento ocorrerá um conflito de colisão de chave primária, pois o dispositivo B tentará cadastrar no banco de dados central um

pedido com o código 1004, sendo que já existe no banco de dados um outro pedido com este mesmo código cadastrado pelo dispositivo A.

Algumas soluções de sincronização encontradas no mercado, como o *SyncML* (SyncML Protocol (2001)), propõem uma arquitetura onde o cliente e o servidor administram seus próprios identificadores únicos independentemente. O identificador do cliente é conhecido como LUID, do inglês *Locally Unique ID*, e o identificador do servidor é conhecido como GUID, do inglês *Globally Unique ID*. Como os identificadores podem ser diferentes, o servidor precisa manter um tabela de mapeamento de identificadores. Por exemplo, para um item chamado “Carro” o LUID do cliente pode ser 11, enquanto o GUID do servidor pode ser 1010101. A tabela de mapeamento do servidor deve mostrar que, para este cliente, o LUID 11 é mapeado para o GUID 1010101. Nessa arquitetura os LUIDs são sempre atribuídos pelo cliente. Se o servidor enviar um novo item para o cliente, o cliente irá atribuir a este item um LUID e enviar o identificador local gerado de volta para o servidor para que o mapeamento seja registrado. A figura 3.2 apresenta um exemplo do mapeamento de identificadores únicos proposto pelo *SyncML*.

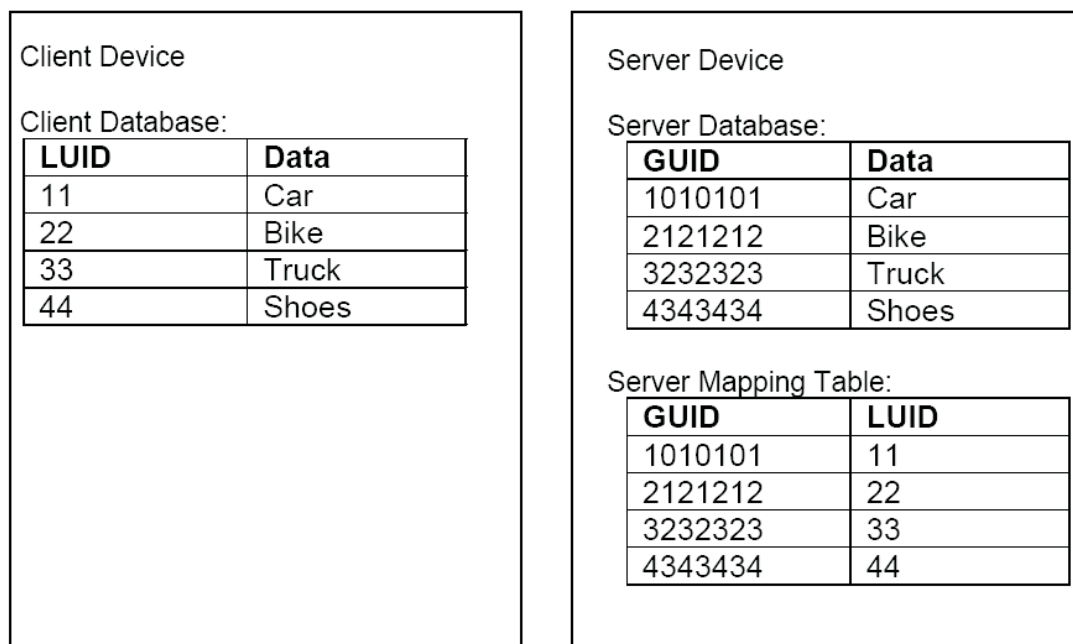


Figura 3.2: Exemplo de mapeamento de identificadores no SyncML

Este procedimento é interessante em situações onde um mesmo dispositivo precisa sincronizar dados com mais de um servidor, pois, nestes casos, cada servidor poderia ter um GUID independente. Porém, no cenário de aplicação do mSync, onde vários dispositivos sincronizam com um único servidor, esta solução não é indicada, pois a medida que o número de dispositivos e o volume de dados aumenta, o tamanho da tabela de mapeamento de identificadores também aumenta, e, considerando-se que a tabela de mapeamento precisa ser acessada para realizar qualquer operação de atualização, inserção ou exclusão de registros durante a sincronização, esse processo pode gerar um aumento significativo de processamento.

Outras soluções, como o *Microsoft Synchronization* (MicrosoftSyncServices (2008)), sugerem que a aplicação se encarregue de gerar identificadores únicos locais que também sejam únicos globalmente. Uma das sugestões é a utilização de um tipo de dado especial do *Microsoft SQL Server*, chamado GUID (Vieira (2007)), como identificador único. Esse tipo de dado garante que os valores gerados como identificadores serão únicos independente do número de clientes. Essa abordagem apresenta os seguintes problemas:

- O tipo de dado utilizado é muito grande (16 bytes), o que aumenta o tamanho dos agrupamentos de índices e afeta negativamente a performance de operações comuns, como *joins*.
- A geração dos GUID é feita de maneira desordenada, o que faz com que os registros sejam inseridos em posições aleatórias no agrupamento de índices. Isso pode causar a fragmentação dos agrupamentos de índices, o que aumenta a quantidade de operações de entrada e saída necessárias para a realização de consultas nas tabelas.

O *framework* mSync propõe uma nova solução para o problema de geração de identificadores únicos em um ambiente distribuído. Quando um dispositivo insere um novo registro em sua base de dados local, é utilizado um LUID (*Locally Unique ID*), único localmente, como identificador do registro. Durante a sincronização, quando o registro é inserido no banco de dados central, o servidor gera um GUID (*Globally Unique ID*) para o registro, que é único globalmente. O GUID gerado é enviado de volta para o dispositivo para que o LUID temporário seja substituído pelo GUID definitivo, fazendo com que os identificadores nas duas arquiteturas, móvel e fixa, fiquem com o mesmo valor.

Essa abordagem não apresenta os problemas discutidos nas outras soluções. Diferente do que acontece no *SyncML*, a tabela de mapeamento de identificadores não possui uma quantidade excessiva de registros, pois o mapeamento só é necessário no momento em que os registros são inseridos no banco de dados central durante a sincronização, podendo depois serem descartados. Na solução proposta os identificadores globais são gerados apenas pelo servidor, o que permite a utilização de identificadores de tipos menores, resultando em agrupamentos de índices mais compactos, *joins* mais eficientes e menos processamento de entrada e saída na consulta às tabelas. A Seção 4.4 detalha o esquema de geração de identificadores únicos proposto nesta dissertação.

### Geração de identificadores únicos globais nos dispositivos

Algumas aplicações exigem que os identificadores globais sejam gerados no próprio dispositivo ubíquo. Um exemplo são aplicações de fiscalização, onde o fiscal precisa imprimir, em campo, autos de infração que contenham o número de identificação do auto gerado.

Nesses casos é preciso utilizar alguma técnica de geração de identificadores que garanta que o identificador gerado no banco de dados local seja único globalmente, não coincidindo com nenhum identificador gerado pelo servidor ou outro dispositivo.

Uma solução é utilizar como identificador a combinação de um valor que seja único localmente, por exemplo, um campo auto-incremental, e um valor que identifique unicamente o dispositivo. Outra solução é utilizar como identificador uma chave natural, como, por exemplo, um número de CPF ou CNPJ como identificador de um cliente.

A desvantagem destas abordagens é que, conforme discutido nesta Seção, elas geram identificadores pouco compactos, que influenciam negativamente no tamanho dos agrupamentos de índices, eficiência dos *joins* e requerimentos de entrada e saída.

### 3.8 Soluções comerciais de bancos de dados móveis

O mercado de computação móvel tem atraído o interesse de grandes empresas. Os maiores fabricantes de SGBDs convencionais já atuam neste segmento, oferecendo *softwares* para desenvolvimento de aplicações e SGBDs adaptados para o ambiente de mobilidade. Para permitir que os usuários tenham acesso à informações mesmo quando desconectados, as aplicações ubíquas precisam, de alguma forma, armazenar na memória local parte das informações disponíveis em servidores da rede fixa. Várias aplicações, incluindo o *framework* mSync, utilizam bancos de dados móveis para este fim. De modo geral, as características desejáveis para SGBDs móveis incluem:

- Suporte para diferentes plataformas de dispositivos, como *Windows Mobile*, *Palm OS* e *Symbian*;
- Estrutura compacta e eficiente, permitindo que sejam utilizados em dispositivos com poucos recursos computacionais, como memória e capacidade de processamento;
- Suporte para criptografia e compactação de dados;
- Fácil integração entre o banco de dados e aplicações desenvolvidas por terceiros;
- Ser livre ou possuir um valor de licença não restritivo.

Esta Seção apresenta uma visão geral sobre os principais bancos de dados móveis disponíveis no mercado.

#### 3.8.1 *SQL Anywhere*

O *SQL Anywhere*, da Sybase (*SQL Anywhere* (2008)), é um pacote de *software* direcionado para o mercado de mobilidade que, dentre outras coisas, inclui um banco de dados compacto, chamado *UltraLite*, projetado para ser utilizado em dispositivos móveis. O *UltraLite* possui as seguintes características:

- *Kernel* pequeno, com aproximadamente 300 KB, adaptado para dispositivos com poucos recursos computacionais;
- Suporte para gerenciamento de transações e integridade referencial;

- Suporte para comandos padrão SQL;
- Suporte para campos do tipo BLOB (Vieira (2007));
- Provê interfaces para utilização em conjunto com as ferramentas de desenvolvimento *Microsoft Visual Studio* e *Metrowerks Code Warrior*;
- Compatível com as plataformas *Windows Mobile*, *Windows Mobile Smartphone Edition*, *Windows CE (Pocket PC, Handheld PC)*, *Windows XP* e *Windows Vista*, *Windows XP Embedded*, *Windows Tablet PC*, *Palm OS* e *Symbian OS*.

O *MobiLink*, que também acompanha o *SQL Anywhere*, é uma ferramenta que permite a sincronização entre o banco de dados *UltraLite* e diversos bancos de dados corporativos, como *Oracle*, *IBM DB2* e *Microsoft SQL Server*.

Semelhante ao mSync, o *MobiLink* suporta operações como rastreamento de modificações no dispositivo e no servidor, geração de identificadores únicos e detecção / resolução de conflitos. Porém, as estratégias propostas no mSync e as utilizadas no *MobiLink* são diferentes. Por exemplo, quando um cliente *MobiLink* envia para o servidor uma atualização, ele transmite não apenas os novos valores, mas também os valores antigos, que foram recebidos na última sincronização. O servidor verifica se o valor antigo não corresponde ao valor corrente no banco de dados central para detectar a ocorrência de conflitos. A estratégia utilizada pelo mSync, apresentada na Seção 4.2, é mais eficiente, pois não exige que sejam transmitidas duas versões de cada registro atualizado.

Para resolver o problema de geração de identificadores únicos em um ambiente distribuído, o *MobiLink* propõe soluções como utilização de chaves compostas e campos do tipo GUID (Vieira (2007)), cujas desvantagens em relação à alternativa proposta nesta dissertação são apresentadas nas Seções 3.7 e 4.4. Outra solução proposta pelo *MobiLink* é a utilização de reserva de faixas ou de um *pool* de identificadores por dispositivo, porém, esta abordagem não é escalável e o gerenciamento das faixas e números reservados introduz complexidade extra ao processo de sincronização de dados.

### 3.8.2 Oracle Lite

A *Oracle* é uma das líderes mundiais no mercado de bancos de dados. A empresa possui o produto *Oracle Lite*, direcionado para o mercado de mobilidade. O *Oracle Database Lite* (Oracle Database Lite (2008)) é uma solução completa, cujo objetivo é permitir o acesso móvel a informações armazenadas em bancos de dados corporativos da própria *Oracle*. O *Oracle Lite* é constituído por três componentes principais, que são:

1. *Client Stack*, que é o módulo utilizado nos dispositivos. Este componente inclui o banco de dados móvel propriamente dito, sendo suportado por diferentes plataformas, como *Windows 32-bit*, *Windows Mobile*, *Linux* e *Symbian OS*.
2. *Mobile Server*, utilizado para sincronização de dados entre o banco de dados móvel, armazenado no dispositivo, e o banco de dados corporativo *Oracle*.

3. Ferramentas de desenvolvimento que podem ser utilizadas para desenvolver aplicações que utilizem este banco de dados.

As principais características do *Oracle Database Lite* são:

- *Kernel* pequeno, adaptado para dispositivos com poucos recursos computacionais;
- Suporte para comandos padrão SQL;
- Suporte para gatilhos e procedimentos armazenados;
- Possui funcionalidade integrada para rastreamento de modificações, o que pode ser utilizado na sincronização de dados;
- Tamanho máximo do banco de dados de 4 GB;
- Suporte para compactação de dados;
- Suporte para criptografia;
- Suporte para conexões ODBC, JDBC e ADO.NET.

O *Oracle Lite* possui funcionalidades que permitem a replicação e sincronização de dados entre os dispositivos e o servidor, porém, diferente do mSync, elas só podem ser utilizadas se o banco de dados de retaguarda também for fornecido pela *Oracle*. A documentação disponibilizada pelo fabricante não detalha o funcionamento e implementação do processo de sincronização de dados.

### 3.8.3 *DB2 Everyplace*

O *IBM DB2 Everyplace* (DB2 Everyplace (2008); Karlsson et al. (2001)) é a solução da IBM para banco de dados móvel. O *DB2 Everyplace Database* possui as seguintes características:

- Compatível com vários sistemas operacionais, como PalmOS, *embedded Linux*, Symbian e Windows Mobile;
- Suporte para comandos padrão SQL;
- *Kernel* pequeno, adaptado para dispositivos com poucos recursos computacionais.

Além do banco de dados *DB2 Everyplace Database*, o produto da IBM inclui uma solução para sincronização de dados, composta pelo *Everyplace Sync Server*, executado no servidor, o *Everyplace Sync Client*, executado no cliente, e o banco de dados espelho *DB2 Everyplace*.

Diferente do mSync, a IBM adotou uma estratégia onde os dispositivos não acessam diretamente o banco central, mas sim um banco de dados espelho, que replica todas as informações do banco de dados corporativo que precisam ser sincronizadas. O banco de dados espelho é um DB2, enquanto o banco de dados central, chamado pela IBM de banco de dados de origem, pode ser fornecido por outras empresas, como *Oracle* e *Microsoft*.

Durante a sincronização de dados os dispositivos recebem e enviam as modificações realizadas durante o período de desconexão para o banco de dados espelho. Em outro momento, previamente agendado ou iniciado pelo administrador, é realizado um processo de replicação de dados, onde as informações do banco de dados espelho são sincronizadas com o banco de dados de origem. Alguns conflitos podem ser detectados nesta etapa e o resultado final da operação só é transmitido para os dispositivos durante o próximo ciclo de sincronização.

A estratégia adotada pela IBM isola o banco de dados de origem da carga de processamento gerada pela sincronização de muitos dispositivos simultaneamente, mas, em contrapartida, exige que os dados sincronizados sejam duplicados e mantidos sincronizados com o banco de dados espelho. Além disto, esta abordagem impede que os usuários móveis tenham acesso às informações mais recentes e as aplicações corporativas recebam as informações atualizadas pelos dispositivos imediatamente após a sincronização.

#### 3.8.4 *SQL Server Mobile*

O *SQL Server Mobile* (SQL Server Mobile (2008)) é o banco de dados móvel desenvolvido pela *Microsoft* para ser utilizado nos dispositivos móveis que utilizam os sistemas operacionais da família *Windows*. As funcionalidades do *SQL Server Mobile* incluem:

- *Kernel* compacto;
- Suporte para acesso multiusuário;
- Possui integração com o ambiente de desenvolvimento *Visual Studio* (VisualStudio (2008));
- Suporte para comandos padrão SQL;
- Suporte para integridade referencial;
- Compatível com os sistemas operacionais da família *Windows*.

O *SQL Server Mobile* oferece duas possibilidades para sincronização de dados com um banco de dados corporativo *Microsoft SQL Server*: *remote data access* (RDA) e *merge replication*. O RDA é uma ferramenta simples que permite que os dispositivos enviem e recebam dados de um banco de dados *SQL Server* armazenado em um servidor. Ele não oferece suporte para operações como rastreamento de modificações, detecção de conflitos ou geração de identificadores, deixando o controle do processo de sincronização sob responsabilidade da aplicação.

O *merge replication* é uma ferramenta direcionada para administradores de bancos de dados. Por meio de uma API própria, a ferramenta permite a configuração de várias funcionalidades, incluindo rastreamento de modificações no dispositivo e no servidor, detecção / resolução de conflitos e gerenciamento de faixas de identificadores únicos de registros por dispositivo.

As duas opções para sincronização de dados disponibilizadas em conjunto com o *SQL Server Mobile* só podem ser utilizadas para sincronizar informações entre o banco de dados *SQL Server Mobile* e o *SQL Server*, desenvolvidos pela própria *Microsoft*.

### 3.9 Trabalhos relacionados

Vários trabalhos encontrados na literatura tratam de questões relacionadas à replicação e sincronização de dados. Tarkoma et al. (2006), Ratner et al. (2004) e Zhang et al. (2003), diferente do mSync, que se ocupa da sincronização de informações corporativas, apresentam arquiteturas focadas na replicação e sincronização de arquivos.

Li et al. (2004) propõe a utilização de regras semânticas definidas pelos usuários para definir o momento em que o processo de sincronização deve ser iniciado. Este trabalho é complementar a arquitetura proposta nesta dissertação e, as regras definidas no trabalho podem ser utilizadas para definir a condição necessária para que o serviço de sincronização fornecido pelo mSync seja iniciado.

Ye et al. (2005) apresenta uma ferramenta de geração automática de código para *conduits* na plataforma PalmOS. *Conduit* é um programa que o *HotSync Manager* executa para sincronizar informações entre um PDA PalmOS e um computador *desktop*. O artigo é um exemplo do esforço despendido com o objetivo de facilitar o desenvolvimento de soluções de sincronização de dados. Porém, a solução proposta é limitada, pois utiliza como base um *conduit* para o *HotSync Manager*, o que não é praticável para uso em ambientes de redes ubíquas. Além disto, na implementação apresentada, a solução suporta apenas sincronizações do tipo espelho, onde é realizada uma cópia integral dos dados em cada sincronização e, apenas em um sentido (*one-way*), do dispositivo para o servidor ou do servidor para o dispositivo.

*SodaSync*, descrito em Castro et al. (2004), é um *framework* de programação que provê um modelo de sincronização genérico para aplicações móveis corporativas. *SodaSync* foca na interface entre um modelo de dados genérico idealizado e uma fonte de dados real. Os dados são modelados utilizando um serviço de objeto de dados, do inglês *Service Data Object* (SDO) (Service Data Objects Specification (2008)). Esta abordagem limita o uso do *SodaSync* a aplicações Java baseados no *framework* SDO. Diferente do mSync, o *SodaSync* não está concluído e não pode ser utilizado como suporte para desenvolvimento de uma solução fim a fim de sincronização de dados.

Trachtenberg et al. (2002); Agarwal et al. (2002); Chauhan (2006) apresentam um algoritmo de sincronização de dados chamado CPISync, que é uma abreviação de “*Characteristic Polynomial Interpolation Synchronization*”. A estratégia básica realizada pelo CPISync é calcular um *hashing* dos dados dentro de um certo tipo de polinômio, conhecido como polinômio característico. O cliente e o servidor mantêm seus próprios polinômios. Durante a sincronização, o cliente envia para o servidor o valor calculado do seu polinômio um número aleatório de vezes. Este número precisa exceder o número de entradas diferentes nas duas bases de dados. Estas amostras são suficientes para que o servidor determine todas as diferenças, através de interpolação e, então, envie para o dispositivo os registros existentes em

sua base de dados que não existem no dispositivo, assim como solicite ao dispositivo o envio dos registros presentes na base de dados móvel, mas não no servidor. A figura 3.3, obtida de Agarwal (2008), detalha o funcionamento do CPISync sendo utilizado para sincronizar um PDA e um computador *desktop* executando uma aplicação de bloco de notas.

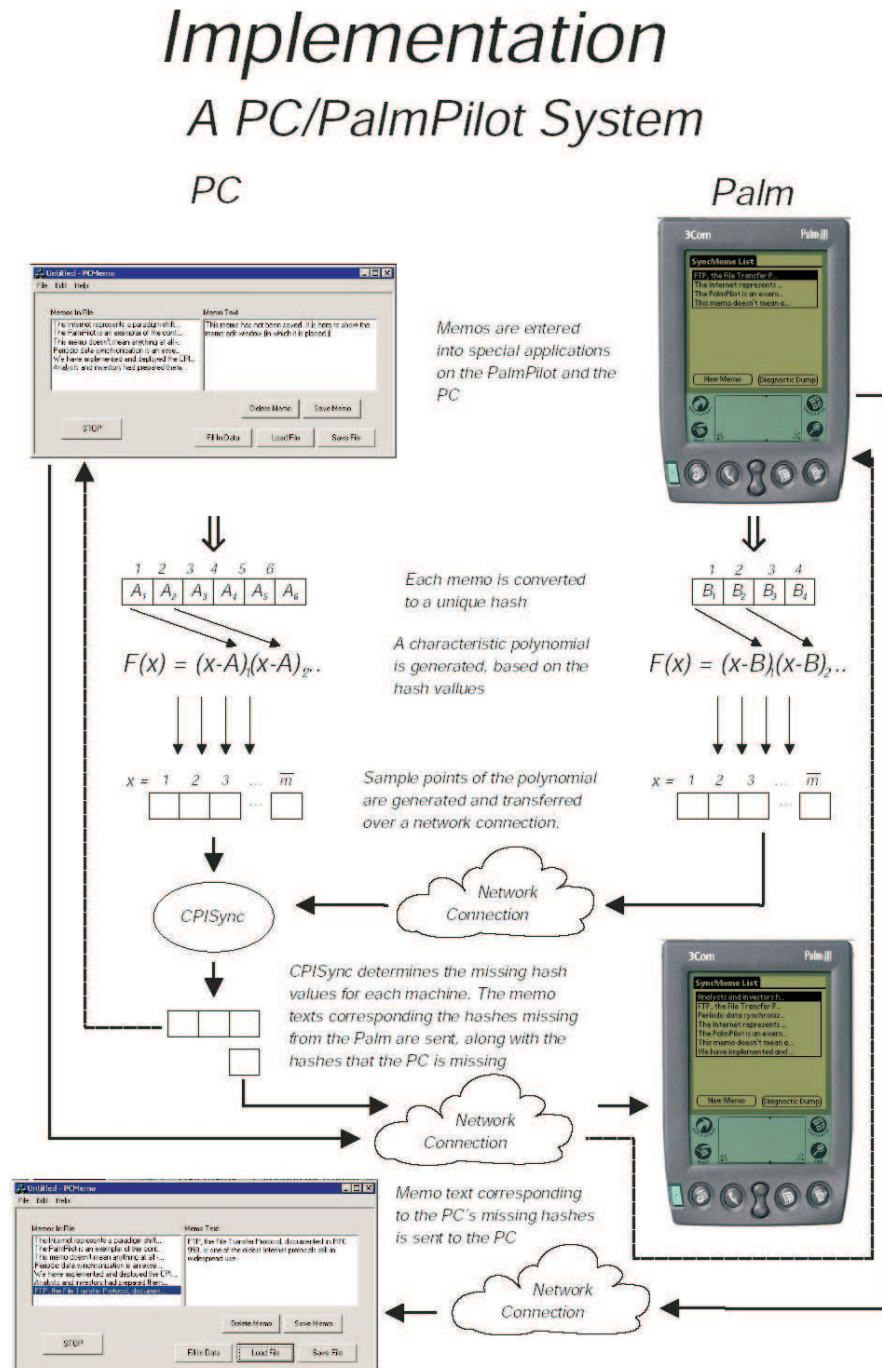


Figura 3.3: Funcionamento do método CPISync

A propriedade notável deste método é que a complexidade da comunicação, durante o processo de sincronização, depende principalmente da quantidade de itens diferentes nos

bancos de dados móvel e fixo, sendo praticamente independente do tamanho do conjunto de dados a ser sincronizado. Além disto, este método não exige que seja mantido um log para rastrear as modificações realizadas. O algoritmo, porém, tem suas limitações. Se existir uma quantidade significativa de diferenças entre as duas bases de dados sincronizadas, o algoritmo acaba transmitindo a mesma quantidade de dados como se todos os registros fossem transmitidos, incluindo aqueles já existentes nos dois *hosts*. Um exemplo de número onde isto ocorre são 175 diferenças ou mais em 250 registros, ou 1.117 diferenças em 10.000 registros. A medida em que o número total de registros aumenta, a razão entre a diferença e o número de registros diminui. Além disto, a medida em que a complexidade dos dados aumenta, o tamanho da variável necessária para armazenar o valor de *hash* único aumenta, assim como o tempo de processamento exigido para cálculo do próprio *hash* e do polinômio característico, o que torna este método inviável para ser utilizado em grande parte das aplicações existentes no mundo real.

O *Microsoft Synchronization Services* (MicrosoftSyncServices (2008)) é um *framework* de sincronização de dados desenvolvido pela *Microsoft* e disponibilizado em conjunto com o *Visual Studio 2008*. Inicialmente ele permitia apenas a sincronização entre computadores *desktop* e um banco de dados centralizado, não sendo compatível com dispositivos. Em março de 2008 foi liberada a atualização da solução que permite que a mesma seja utilizada para sincronização de dados entre dispositivos equipados com os sistemas operacionais *Windows Mobile 5* ou *Windows Mobile 6*, executando um banco de dados *SQL Server Mobile*, e um servidor de retaguarda utilizando qualquer banco de dados compatível com a tecnologia *ADO .NET* (ADO .NET (2008)).

O *Sync Service* possui algumas características semelhantes à arquitetura mSync, como ser focado na sincronização de informações corporativas e ser aplicável a situações onde vários dispositivos sincronizam informações com um único servidor. Além disto, da mesma forma que o mSync, o *Sync Service*, da *Microsoft*, também é uma arquitetura que permite a construção de uma solução completa de sincronização de dados. Porém, por ser uma solução comercial, o *Sync Service* não detalha como os processos são executados em *background*, impedindo que seja realizado um estudo comparativo minucioso entre as abordagens adotadas no mSync e no *Sync Service*. O produto da *Microsoft* é restrito a plataforma *Windows Mobile* e banco de dados *SQL Server Mobile*, enquanto o mSync, embora na atual implementação seja compatível apenas com o *Windows Mobile*, pode ser expandido para suportar outras plataformas. Os esquemas de geração e conversão de identificadores únicos propostos no mSync, conforme detalhado nas Seções 3.7 e 4.4, possuem vantagens em relação ao sugerido pelo *Sync Service*.

Com o objetivo de padronizar o processo de sincronização de dados e permitir que dispositivos desenvolvidos por diferentes fabricantes possam trocar informações entre si, várias empresas, incluindo *Ericsson*, *IBM*, *Lotus*, *Matsushita*, *Motorola*, *Nokia*, *Palm*, *Psion* e *Starfish*, se uniram para propor um padrão, baseado no XML, chamado SyncML (*Synchronization Markup Language*) (SyncML Protocol (2001); Ren e Song (2002); Lee et al. (2002); Ren et al. (2004); Lee et al. (2007)).

O SyncML hoje é suportado por vários dispositivos de acesso a informação, como, por

exemplo, telefones inteligentes da *Nokia* e *SonyEricsson*, sendo utilizado principalmente para sincronização de informações de gerenciamento pessoal, do inglês *Personal Information Manager* (PIM), como agenda de contatos e calendário. O SyncML, por ser um padrão comum, permite, por exemplo, que contatos registrados em um telefone com o sistema operacional *Symbian* sejam sincronizados com um PDA com o sistema operacional *Windows Mobile*, desde que ambos suportem o protocolo.

O SyncML especifica as interfaces necessárias e o resultado final do processo de sincronização, mas não define a forma como o padrão deve ser implementado, de modo que cada empresa pode implementar o protocolo da forma que achar mais apropriada. A figura 3.4 apresenta a estrutura do padrão SyncML.

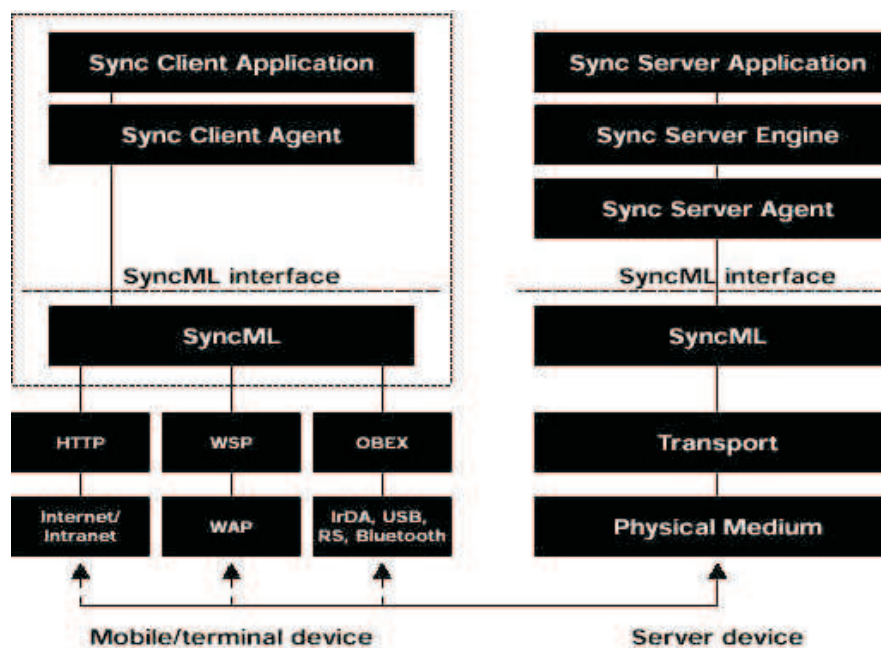


Figura 3.4: *Framework SyncML*

O *SyncML* padroniza o formato das mensagens utilizadas para sincronização de dados entre dois dispositivos, concentrando-se, principalmente, em informações pessoais, como contatos, calendários e listas de tarefas. Uma sugestão de trabalho futuro, é modificar o mSync para suportar o SyncML como protocolo para troca de informações entre os dispositivos e o servidor.

O GDA (Sales (2008)) é um *framework* para mapeamento objeto relacional, escrito na linguagem C# .NET, que realiza a persistência de objetos C# em bancos de dados relacionais. Mapeamento objeto-relacional, ou, do inglês *object relational mapping* (ORM), é uma técnica de desenvolvimento utilizada para diminuir a complexidade da programação orientada a objetos utilizando bancos de dados relacionais (Keller (1998)). As tabelas do banco de dados são representadas por meio de classes e os registros de cada tabela são representados como instâncias das classes correspondentes. Com esta técnica, o programador não precisa se preocupar em escrever comandos em linguagem SQL, pois o *framework* disponibiliza uma

interface de programação simples que realiza todo o trabalho de persistência de dados.

Não é necessária uma correspondência direta entre as tabelas de dados e as classes do programa. A relação entre as tabelas onde se originam os dados e o objeto que os disponibiliza é configurada pelo programador, isolando o código do programa da estrutura física dos dados nas tabelas do banco de dados.

O GDA permite este mapeamento por meio de *metadados* definidos nas classes do modelo de dados. A figura 3.5 apresenta um exemplo de classe de modelo de dados onde é possível observar o mapeamento sendo realizado por meio dos atributos *PersistenceClass*, para a tabela no banco de dados, e *PersistenceProperty*, para os campos.

```
[PersistenceClass("TBPessoa")]
public class Pessoa
{
    private int m_codPessoa;
    [PersistenceProperty("codPessoa", PersistenceParameterType.IdentityKey)]
    public int CodPessoa
    {
        get { return m_codPessoa; }
        set { m_codPessoa = value; }
    }

    private string m_nome;
    [PersistenceProperty("nome")]
    public string Nome
    {
        get { return m_nome; }
        set { m_nome = value; }
    }

    private string m_apelido;
    [PersistenceProperty("apelido")]
    public string Apellido
    {
        get { return m_apelido; }
        set { m_apelido = value; }
    }
}
```

Figura 3.5: Exemplo de mapeamento objeto relacional utilizando o GDA

No GDA a persistência e acesso aos dados é efetuado por meio de objetos de acesso a dados, ou, do inglês *data access object* (DAO). O DAO fornece a interface utilizada pelo programador para interagir com o banco de dados, obtendo informações e persistindo dados. A figura 3.6 apresenta um exemplo de inclusão de registro em um banco de dados utilizando o GDA. O GDA é um *framework* complementar ao mSync, onde é utilizado como suporte para acesso e persistência de dados.

## 3.10 Conclusões

Este Capítulo apresentou os principais conceitos relacionados à replicação e sincronização de dados. Foram apresentadas as duas principais estratégias relacionadas à consistência em

```
Pessoa pessoa = new Pessoa();  
pessoa.Nome = "JOAO ALVES DA SILVA";  
pessoa.Apelido = "JAO";  
//Insere registro no banco de dados  
PessoaDAO dao = new PessoaDAO();  
dao.Insert(pessoa);
```

Figura 3.6: Exemplo de persistência de dados utilizando o GDA

bancos de dados móveis, a pessimista e a otimista. A abordagem pessimista, como foi visto, garante a consistência do banco de dados limitando a disponibilidade das informações. A abordagem otimista, em contrapartida, aumenta a disponibilidade dos dados flexibilizando algumas propriedades de consistência. O *framework* proposto vai ao encontro dos principais trabalhos disponíveis na literatura e adota a estratégia otimista, que é considerada mais apropriada para um ambiente de computação ubíqua.

Quando a estratégia otimista é utilizada, é comum a existência de conflitos, que ocorrem quando dois ou mais dispositivos modificam a mesma informação. Este Capítulo detalhou o conceito de conflito e apresentou algumas sugestões de como eles podem ser resolvidos. Depois foram apresentadas as principais abordagens para troca de dados entre os dispositivos e o servidor, assim como os motivos que levaram a escolha do padrão SOAP para intercâmbio de dados na arquitetura mSync.

O Capítulo apresentou ainda os conceitos referentes a âncoras de sincronização, carga inicial e particionamento de dados para execução em modo desconectado, rastreamento de modificações, sendo concluído com uma revisão dos principais trabalhos relacionados. O próximo Capítulo detalha a arquitetura e funcionamento do *framework* mSync.

## Capítulo 4

### *Framework* mSync

Este Capítulo apresenta o mSync (*mobile Synchronization*), um *framework* focado na sincronização de dados em dispositivos ubíquos, detalhando o funcionamento e principais componentes da solução proposta.

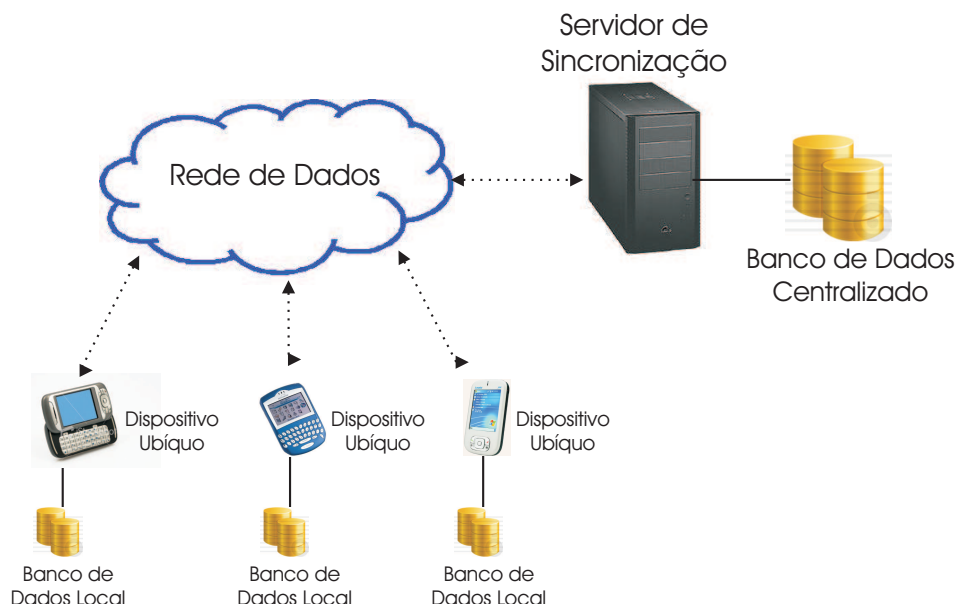


Figura 4.1: Cenário de aplicação do *framework* mSync

O *framework* mSync é aplicável a situações onde vários dispositivos ubíquos realizam a sincronização de dados com um único servidor central, em uma topologia do tipo estrela, conforme apresentado na figura 4.1. Para permitir que os usuários acessem e modifiquem informações mesmo sem uma conexão de rede disponível, os dispositivos precisam armazenar na memória local uma cópia parcial do banco de dados central.

A figura 4.2 apresenta o fluxo para operação em modo desconectado. Em um primeiro momento, com a conexão ativa, é realizada uma cópia parcial dos dados do servidor no dispositivo. Quando a conexão é interrompida, o dispositivo continua acessando as informações em modo *off line*, permitindo a consulta, modificação, inclusão e exclusão dos registros. Depois, com a conexão restabelecida, o banco de dados local do dispositivo é sincronizado com

o banco de dados central da rede fixa, de forma que os dados das duas plataformas fiquem consistentes entre si.

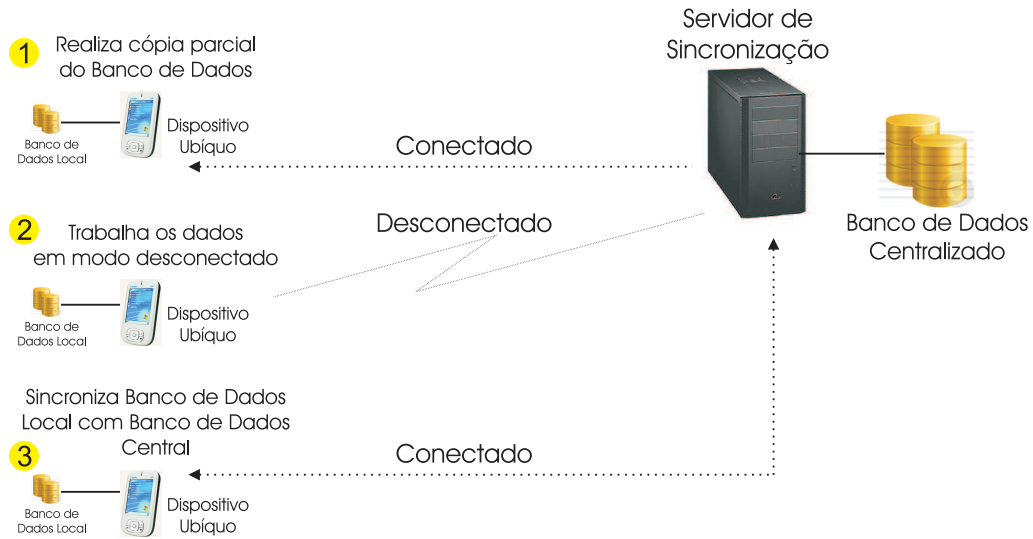


Figura 4.2: Fluxo para operação em modo desconectado

O Capítulo está organizado da seguinte forma: a Seção 4.1 detalha como as âncoras de sincronização, apresentadas na Seção 3.4, são utilizadas no *framework*. A Seção 4.2 apresenta a forma com que o mSync trata os erros e conflitos que podem ocorrer durante a sincronização. A Seção 4.3 apresenta a forma como o rastreamento das modificações realizadas nas plataformas móvel e fixa é realizado na arquitetura. A Seção 4.4 se dedica ao processo de geração de identificadores únicos em um ambiente distribuído. A Seção 4.5 apresenta os principais componentes da arquitetura mSync. A Seção 4.6 apresenta questões relacionadas ao modelo de reuso e classificação do *framework* mSync. A Seção 4.7 detalha os diagramas de sequência referentes ao processo de sincronização e, por fim, a Seção 4.8 encerra o Capítulo com as conclusões e comentários finais.

## 4.1 Âncoras de Sincronização

Uma âncora de sincronização (ver Seção 3.4) representa um ponto de referência utilizado para determinar quais modificações precisam ou não serem atualizadas durante o processo de sincronização de dados. Por meio dela, é possível enviar para cada dispositivo apenas os registros que foram modificados no banco de dados central durante o período de desconexão, verificando-se quais são aqueles cujo o valor da âncora no momento em que foram modificados era maior ou igual a âncora da última sincronização e menor do que a âncora da sincronização atual do dispositivo.

Os registros cujo valor da âncora no momento em que foram modificados (*idSyncChange*) era menor do que o valor da âncora da última sincronização do dispositivo, não devem ser sincronizados, pois essa situação indica que o dispositivo já recebeu estas modificações em alguma sincronização anterior. Já os registros, cujo *idSyncChange* é igual ou superior ao valor

da âncora da sincronização atual, também não devem ser sincronizados, pois essa condição indica que esses registros foram modificados após a sincronização atual ter sido iniciada, não sendo possível determinar se a modificação ocorreu antes ou depois das alterações serem transmitidas para o dispositivo.

Algumas arquiteturas, como a *Microsoft Synchronization Service* (MicrosoftSyncServices (2008)), sugerem que sejam utilizados valores do tipo estampa de tempo (*timestamp*) (Vieira (2007)) ou funções de alguns bancos de dados, que retornam um valor que é incrementado cada vez que uma alteração é realizada, como âncora de sincronização. Essa abordagem é apropriada quando o objetivo é apenas determinar quais registros foram modificados no banco de dados central durante o período de desconexão, utilizando a regra explicada anteriormente.

O mSync, diferente da abordagem da *Microsoft*, utiliza as âncoras de sincronização para, além de determinar quais modificações precisam ser sincronizadas entre o servidor e os dispositivos, fazer a conversão dos identificadores únicos locais (LUID), gerados nos dispositivos quando um novo registro é inserido no banco de dados local, em identificadores únicos globais (GUID), gerados no banco de dados central. O processo de conversão de identificadores únicos utilizando âncoras de sincronização possui vantagens em relação aos métodos tradicionais, conforme será apresentado na Seção 4.4.

Para realizar o processo de conversão de identificadores, é necessário que cada dispositivo, em cada sincronização, possua um valor de âncora diferente e único, que é chamado na arquitetura de identificador de sincronização ou *idSync*.

As tabelas do banco de dados central, que suportam o modo de sincronização incremental (ver Seção 4.7), estão associadas a um *idSync* independente, que é incrementado em uma unidade cada vez que algum dispositivo inicia a sincronização com a tabela em questão.

A figura 4.3 e tabela 4.1 apresentam um exemplo de estrutura que pode ser utilizada para armazenar o valor das âncoras de sincronização.

| appTables |          |      |
|-----------|----------|------|
| codTable  | int      | <pk> |
| tableName | char(30) |      |
| idSync    | int      |      |

Figura 4.3: Estrutura da tabela appTables

| codTable | tableName | idSync |
|----------|-----------|--------|
| 1        | Clientes  | 85     |
| 2        | Produtos  | 89     |
| 3        | Vendas    | 87     |

Tabela 4.1: Exemplo de registros da tabela appTables

## 4.2 Erros e Conflitos

Conforme visto na Seção 3.2, os processos de replicação e sincronização de dados que adotam a abordagem otimista, como o mSync, estão sujeitos a ocorrência de erros e conflitos, que devem ser detectados e resolvidos. Erros durante a sincronização são essencialmente um tipo de conflito e, normalmente, envolvem violação de restrições, como chaves estrangeiras ou valores válidos para campos específicos. Um registro está em conflito quando é modificado por mais de um nó entre sincronizações. As soluções de mobilidade devem ser projetadas para minimizar a ocorrência de conflitos, pois a sua detecção e resolução introduzem complexidade, processamento e tráfego de rede adicionais. Uma das formas de se evitar conflitos é garantir que apenas um nó modifique um determinado registro de cada vez, o que pode ser feito dividindo-se as informações carregadas em cada dispositivo de modo que um mesmo registro não esteja disponível no banco de dados local de mais de um dispositivo ao mesmo tempo. Um exemplo dessa estratégia pode ser encontrada em aplicações de automação de força de vendas, onde cada vendedor é responsável por uma determinada região e, desta forma, são carregados localmente em seu dispositivo móvel apenas as informações referentes aos clientes da sua região. Entretanto, nem sempre a estratégia de dividir os registros entre os dispositivos para evitar conflitos pode ser utilizada, sendo comum a existência de aplicações onde vários dispositivos precisam compartilhar as mesmas informações.

Os conflitos podem ocorrer em qualquer cenário onde mais de um nó possa modificar um mesmo registro ao mesmo tempo. Considere o seguinte exemplo:

1. Dispositivo A e dispositivo B sincronizam com o servidor.
2. Um registro é atualizado no dispositivo A e o dispositivo A sincroniza. Não existe conflito e a atualização é aplicada no servidor.
3. O mesmo registro é atualizado no dispositivo B, que também sincroniza. O registro no dispositivo B está em conflito com o registro no servidor devido a atualização realizada pelo dispositivo A.
4. Se o conflito for resolvido em favor do servidor, a alteração realizada pelo dispositivo B será descartada, e a versão do registro gerada pelo dispositivo A será aplicada no dispositivo B. Se o conflito for resolvido em favor do dispositivo B, a atualização realizada pelo dispositivo B será aplicada no servidor e, quando o dispositivo A sincronizar novamente com o servidor, a modificação realizada pelo dispositivo B será aplicada no dispositivo A.

### 4.2.1 Detecção de Erros e Conflitos

O mSync identifica e trata os conflitos na granularidade de registros. Os tipos de conflitos detectados pelo *framework* e as condições necessárias para caracterizá-los são descritas a seguir:

- Um conflito do tipo ***ClientUpdateServerUpdate*** ocorre quando o servidor e um dispositivo atualizam um mesmo registro. É o tipo mais comum de conflito, sendo identificado quando o *idSyncChange* atual (ver Seção 4.3.2) do registro no servidor é superior ao *idSync* da última sincronização realizada pelo dispositivo.
- Um conflito do tipo ***ClientUpdateServerDelete*** ocorre quando um dispositivo tenta modificar um registro que foi excluído pelo servidor. Ele é detectado quando, durante a sincronização, uma operação de atualização de dados não afeta nenhum registro.
- Um conflito do tipo ***ErrorsOccured*** ocorre quando algum erro impede que uma atualização realizada pelo dispositivo seja aplicada no servidor. Este tipo de conflito normalmente está associado a restrições de chaves estrangeiras ou valores válidos para os campos no banco de dados central.

#### 4.2.2 Resolução de Erros e Conflitos

Os conflitos do tipo *ClientUpdateServerUpdate* disparam um evento no servidor que permite que os desenvolvedores, baseados em critérios como tipo de usuário que realizou a modificação, data da atualização ou qualquer outro que faça sentido do ponto de vista semântico da aplicação, definam qual medida deve ser adotada, como, por exemplo, registrar a ocorrência em um arquivo de log, mesclar os registros ou considerar uma das duas versões.

Os conflitos dos tipos *ClientUpdateServerDelete* e *ErrorsOccured* disparam um evento no dispositivo, para que a aplicação decida, em cada caso, qual providência adotar, como, por exemplo, ignorar o erro, modificar o registro e tentar novamente ou, simplesmente, apresentar o problema para que o usuário defina o que deve ser feito.

### 4.3 Rastreamento de modificações

Para possibilitar um processo de sincronização eficiente, onde apenas as informações modificadas são transmitidas entre os dispositivos e o servidor, é necessário que os dois ambientes, móvel e fixo, implementem mecanismos para registrar as modificações realizadas em suas bases de dados durante os períodos de desconexão, conforme visto na Seção 3.6. As modificações podem ser de três tipos: inserção, atualização ou exclusão de registros. Esta Seção detalha como esta questão é tratada no *framework* mSync.

#### 4.3.1 Rastreamento de modificações no dispositivo

O log de modificações no dispositivo é mantido por meio de um campo de controle, chamado *statusReg*, com a função especial de armazenar o estado de cada registro, que pode ser um dos descritos a seguir:

1. **Novo (*New*):** Registro inserido no dispositivo e ainda não sincronizado.
2. **Atualizado (*Updated*):** Registro modificado no dispositivo e ainda não sincronizado.

3. **Sincronizado (*Synchronized*):** Registro não modificado desde a última sincronização.
4. **Excluído (*Removed*):** Registro excluído no dispositivo e ainda não sincronizado.

Durante a sincronização incremental (ver Seção 4.7) são transmitidos do dispositivo para o servidor apenas os registros nos estados novo, atualizado e excluído. Após a sincronização, os estados são modificados, de forma que os registros atualizados e novos fiquem com o estado sincronizado e os excluídos sejam removidos fisicamente da base de dados local.

#### 4.3.2 Rastreamento de modificações no banco de dados central

O processo de rastreamento de modificações no banco de dados central, armazenado em um servidor da rede fixa, não pode ser realizado por meio de *flags* de estado, como é feito nos bancos de dados móveis, pois diferente dos dispositivos que sincronizam informações com um único servidor, o servidor da rede fixa deve ser capaz de sincronizar seus dados com vários dispositivos.

As âncoras de sincronização, apresentadas nas Seções 3.4 e 4.1, são utilizadas no rastreamento das modificações. As tabelas do banco de dados central que suportam a sincronização incremental possuem um campo especial, chamado na arquitetura de *idSyncChange*, que armazena o valor da âncora vigente no momento em que um registro é inserido ou atualizado. O esquema proposto não faz distinção entre inserções e atualizações, de forma que o campo *idSyncChange* deve ser atualizado no momento em que um registro é criado ou modificado.

O log de exclusão de registros é implementado por meio de uma tabela conhecida na literatura como *tombStone* (MicrosoftSyncServices (2008); SyncML Protocol (2001)). Sempre que um registro é excluído no banco de dados central, deve ser criada uma nova linha na *tombStone* para registrar o evento. A figura 4.4 apresenta a estrutura da tabela *tombStone* utilizada pelo mSync, cujos campos possuem as funções especificadas a seguir:

- **codTable:** Código da tabela de onde o registro foi excluído.
- **GUID:** Identificador único global do registro.
- **idSync:** Valor da âncora de sincronização, referente à tabela de onde o registro foi excluído, no momento da exclusão.
- **idDeviceRemove:** Identificador do dispositivo que excluiu o registro.
- **dtRemove:** Data em que o registro foi excluído.

Durante a sincronização incremental, o servidor envia para o dispositivo apenas os registros cujo *idSyncChange* seja igual ou maior do que a âncora da última sincronização realizada pelo dispositivo e menor do que a âncora da sincronização atual. Isso garante que apenas os registros inseridos ou atualizados no servidor, após a última sincronização, serão enviados para o dispositivo. A figura 4.5 ilustra este processo, onde apenas os registros cujo

| tombStone      |          |      |
|----------------|----------|------|
| codTable       | int      | <pk> |
| GUId           | int      | <pk> |
| idSync         | int      |      |
| idDeviceRemove | int      |      |
| dtRemove       | datetime |      |

Figura 4.4: Estrutura da tabela tombStone

*idSyncChange* seja maior ou igual 54 e menor do que 70 são transmitidos do servidor para o dispositivo durante a sincronização de dados.

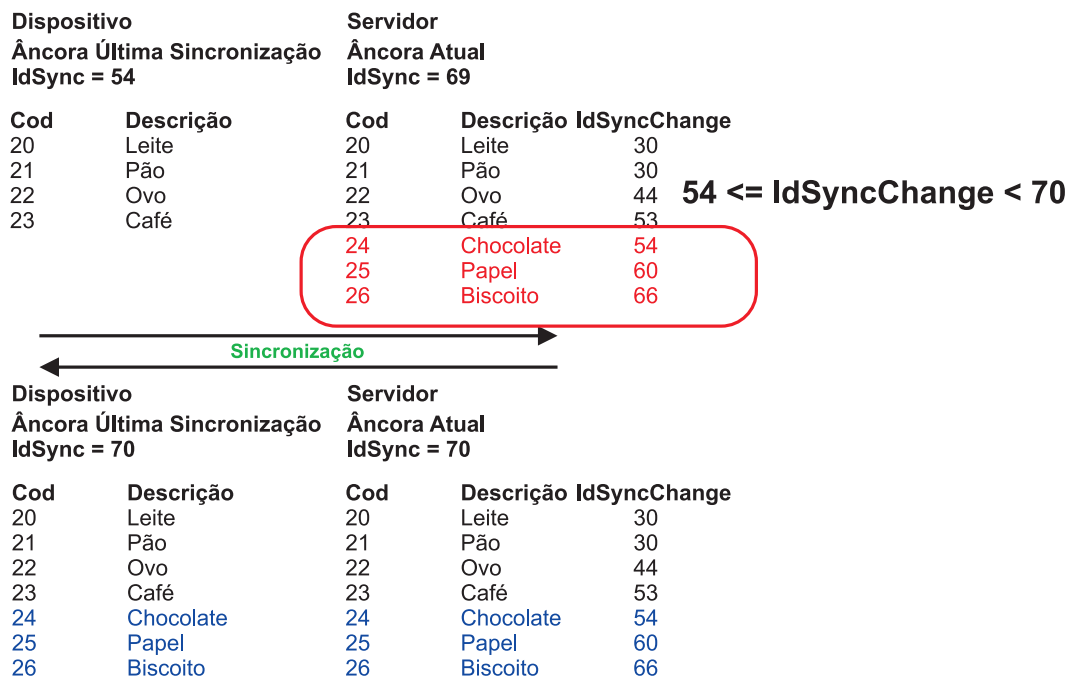


Figura 4.5: Exemplo do uso de âncoras no rastreamento de modificações

Os registros excluídos no servidor são tratados de forma análoga, sendo enviados para o dispositivo apenas os registros da *tombStone* cujo *idSync* seja igual ou maior do que a âncora da última sincronização realizada pelo dispositivo e menor do que a âncora da sincronização atual.

#### 4.3.2.1 O efeito eco na sincronização de dados

A arquitetura mSync realiza um tratamento para evitar que as modificações enviadas pelo dispositivo na primeira fase da sincronização (ver Seção 4.7), sentido DM (Dispositivo Móvel) para SF (Servidor Fixo), sejam transmitidas de volta para o DM na segunda fase da sincronização, sentido SF para DM, o que, neste trabalho, é denominado efeito eco.

Essa situação ocorre porque quando o dispositivo envia as modificações para o servidor fixo, no momento em que os registros são inseridos ou atualizados na base de dados central, o seu *idSyncChange* é atribuído com o valor da âncora de sincronização atual referente à tabela

sincronizada. Assim, na segunda fase da sincronização, quando o servidor envia as modificações para o dispositivo, esses registros terão um *idSyncChange* igual ou maior do que a âncora da última sincronização realizada pelo dispositivo e menor do que a âncora da sincronização atual e, portanto, conforme explicado anteriormente, serão enviados desnecessariamente para o dispositivo.

Para evitar esta ocorrência, o *framework* registra na base de dados central o código do dispositivo que inseriu, atualizou ou excluiu cada registro. Desta forma, durante a segunda fase da sincronização, são enviados para o dispositivo apenas os registros modificados (inseridos, atualizados ou excluídos) cujo código do dispositivo que realizou a modificação seja diferente do código do dispositivo que está sincronizando.

Deve-se garantir que quando as modificações na base de dados central não forem realizadas durante a sincronização com os dispositivos, o código do dispositivo que realizou a modificação seja atribuído com o valor zero. Esse procedimento é importante pois, caso não seja realizado, um dispositivo que inseriu ou atualizou um determinado registro não receberá as novas modificações realizadas na base de dados central por outras fontes, que não as sincronizações.

#### 4.3.2.2 Método não intrusivo para rastreamento de modificações no banco de dados central

Em algumas situações, principalmente quando a solução móvel está sendo integrada a sistemas legados, não é possível alterar a estrutura das tabelas existentes no banco de dados central para inserir os campos utilizados no processo de manutenção do log de alterações.

Nestas situações, as informações referentes ao valor da âncora de sincronização atual e o identificador do dispositivo que realizou cada modificação no banco de dados central podem ser armazenadas em uma tabela de dados separada, criada especialmente para este fim, com estrutura semelhante à exibida na figura 4.6.

| cntrChanges    |     |      |
|----------------|-----|------|
| codTable       | int | <pk> |
| GUId           | int | <pk> |
| idSyncChange   | int |      |
| idDeviceChange | int |      |

Figura 4.6: Estrutura da tabela de controle de alterações

Essa abordagem evita a necessidade de alterações na estrutura de tabelas já existentes, mas em contrapartida, cria um *overhead* para o sistema nas operações de inserção e atualização de registros. Durante a sincronização, os mesmos critérios já apresentados para selecionar os registros que serão enviados para o dispositivo continuam válidos, porém agora é necessário realizar um *join* com a tabela de controle de alterações para que eles possam ser aplicados.

## 4.4 Geração de identificadores únicos no *framework* mSync

A necessidade de identificar unicamente os registros é uma característica presente na maior parte dos sistemas de informação. Conforme visto na Seção 3.7, o caráter distribuído das soluções de mobilidade impõe um grande desafio na tarefa de gerar identificadores que sejam únicos, evitando o problema de colisão de chaves primárias, e, ao mesmo tempo, compactos, evitando que as operações de acesso a dados sejam afetadas negativamente. O *framework* mSync propõe um esquema para geração de identificadores que atende a estes dois objetivos e, por isto, é mais eficiente do que as alternativas propostas por outras arquiteturas, como o *SyncML* (SyncML Protocol (2001)) e *Microsoft Sync Service* (MicrosoftSyncServices (2008)), apresentadas na Seção 3.7.

As tabelas sincronizadas por meio do *framework* mSync devem possuir um campo utilizado para identificar unicamente cada registro. Este campo é conhecido como chave primária ou *primary key* (Vieira (2007)). Quando um registro é inserido na base de dados local de um dispositivo, o *framework* se encarrega de atribuir ao identificador do registro um valor que seja único localmente (LUID). Durante a sincronização, quando o registro é inserido na base de dados central, ele recebe um identificador global (GUID), que é gerado pelo banco de dados central. No final da sincronização, o identificador local no dispositivo é substituído pelo identificador global definitivo, que foi gerado no servidor.

O mSync trata as questões relacionadas à geração dos LUIDs e GUIDs, escondendo dos programadores a complexidade envolvida no processo. O LUID deve ser único localmente, o que significa que não deve existir, na base de dados local do dispositivo, nenhum outro registro com o mesmo identificador. O procedimento 1 descreve a regra utilizada pelo *framework* para gerar os identificadores locais.

---

### Procedimento 1-Regra utilizada para gerar os identificadores únicos locais

---

**Se**  $\max(\text{Identificador}) > 900.000.000$  **então**

$\text{NovoIdentificador} = \max(\text{Identificador}) + 1$

**Senão**

$\text{NovoIdentificador} = 900.000.001$

**Fim Se**

---

Este processo garante que o valor do LUID será sempre único localmente. A constante 900.000.000, utilizada no procedimento 1, pode ser ajustada, para mais ou para menos, dependendo do valor máximo suportado pelo tipo de campo utilizado como chave primária. Em uma análise simples, alguém poderia sugerir que os identificadores locais fossem gerados por meio do procedimento 2, descrito a seguir:

---

### Procedimento 2-Regra que não deve ser utilizada para gerar os identificadores únicos locais

---

$$\text{NovoIdentificador} = \max(\text{Identificador}) + 1$$

O problema com o procedimento 2, que simula um campo do tipo auto-incremental (Vieira (2007)), é que, no final do processo de sincronização, quando os LUIDs são substituídos pelos GUIDs na base de dados local do dispositivo, podem ocorrer colisões entre um GUID definitivo, gerado no servidor, e um LUID provisório, que ainda não foi substituído pelo GUID definitivo dentro da mesma sessão de sincronização. A classe do *framework* responsável pela geração do LUID é a *ClientSyncDAO*, que será detalhada na Seção 4.5.5.

No banco de dados central são utilizados campos do tipo auto-incremental para as chaves primárias. Com isto, quando um registro é inserido no banco de dados central, o próprio banco de dados se encarrega de criar um identificador único global para a linha.

Durante a sincronização de dados, os registros inseridos no dispositivo são aplicados no servidor e os GUIDs gerados são enviados de volta para o dispositivo para que este substitua cada LUID pelo seu respectivo GUID, de forma que os identificadores únicos sejam os mesmos nas plataformas fixa e móvel. O *framework* se encarrega de, automaticamente, durante a sincronização, substituir os LUIDs pelos GUIDs nas tabelas sincronizadas. Porém, é comum que as aplicações utilizem relacionamentos onde o identificador de uma tabela é referenciado em outras tabelas. Caso esteja sendo utilizado um banco de dados móvel com suporte para relacionamentos do tipo *update cascade* (Vieira (2007)), o próprio banco de dados pode se encarregar de atualizar o identificador do registro nas outras tabelas onde ele é utilizado. Porém, se o banco de dados não suportar este tipo de relacionamento ou, por algum motivo, ele não puder ser utilizado, o mSync, após substituir o LUID pelo GUID, dispara um evento (ver Apêndice A.2.8) no cliente que permite que a aplicação atualize o identificador da tabela sincronizada nas outras tabelas onde ele é utilizado.

## Tratamento de falhas de conexão

Quando um registro inserido no dispositivo é aplicado na base de dados central, se, por algum motivo, a conexão entre o dispositivo e o servidor for interrompida antes que o dispositivo receba o GUID correspondente ao registro aplicado, o cliente não terá como detectar se o registro foi ou não inserido na base de dados central e, na próxima sincronização, irá enviar o mesmo registro novamente para que ele seja aplicado no servidor. Como não existe relação entre o LUID e o GUID, o servidor não possui meios para detectar que este registro já foi inserido no banco de dados central, o que, neste caso, faria com que o registro fosse criado em duplicidade.

Para evitar a situação descrita acima, o mSync utiliza uma tabela de controle para realizar o mapeamento de chaves locais em chaves globais, chamada *idMapping*, cuja estrutura é exibida na figura 4.7.

Os campos da tabela *idMapping* possuem a seguinte função:

- **idSync:** Valor da âncora de sincronização referente à tabela e sincronização executada pelo dispositivo.

| idMapping |          |      |
|-----------|----------|------|
| idSync    | int      | <pk> |
| codTable  | int      | <pk> |
| LUId      | int      | <pk> |
| GUId      | int      |      |
| dtInsert  | datetime |      |

Figura 4.7: Estrutura da tabela idMapping

- **codTable:** Código da tabela onde o registro foi inserido.
- **LUId:** Identificador único local atribuído pelo mSync no momento em que o registro foi criado no banco de dados móvel.
- **GUId:** Identificador único global gerado no momento em que o registro foi inserido no banco de dados central.
- **dtInsert:** Data em que o registro foi inserido no banco de dados central.

Sempre que um registro é inserido no banco de dados central por meio do processo de sincronização de dados, o *ServerSyncDAO* (ver Seção 4.5.11) cria uma linha de controle na tabela idMapping, que é utilizada para relacionar o LUID temporário, criado no dispositivo, com o GUID definitivo, criado no servidor.

Antes que qualquer novo registro enviado pelos dispositivos seja inserido no banco de dados central, o *ServerSyncDAO* verifica, por meio de uma consulta utilizando chave exata (idSync, codTable, LUId), se a linha referente a este registro já existe na tabela idMapping. Caso a linha exista, significa que o registro já foi inserido no banco de dados central e, por algum motivo, o último processo de sincronização não foi totalmente concluído, de modo que o dispositivo ainda não foi notificado da inclusão do registro e o LUID substituído pelo GUID no banco de dados local. Nessa situação, ao invés de inserir o registro no banco de dados central, o mesmo é atualizado utilizando-se o GUID, referente a esse registro, armazenado na idMapping. Caso a linha não exista, significa que o registro é realmente novo e ele é inserido na tabela correspondente do banco de dados central, sendo que, após a inserção, é criado um registro de controle na tabela idMapping. A figura 4.8 apresenta um exemplo do processo de conversão de identificadores locais para identificadores globais.

O que garante o funcionamento da técnica proposta é o fato da chave da tabela idMapping, formada pelos campos idSync, codTable e LUId ser única. Isto ocorre porque a forma como o LUID é atribuído no dispositivo garante que, para uma mesma tabela (codTable) e um mesmo período de desconexão (idSync), nunca existirão dois LUIDs coincidentes.

## 4.5 Componentes da arquitetura

A figura 4.9 apresenta uma visão dos elementos que compõem o *framework* mSync. Os blocos circundados pelos retângulos pontilhados representam classes padronizadas para persistência

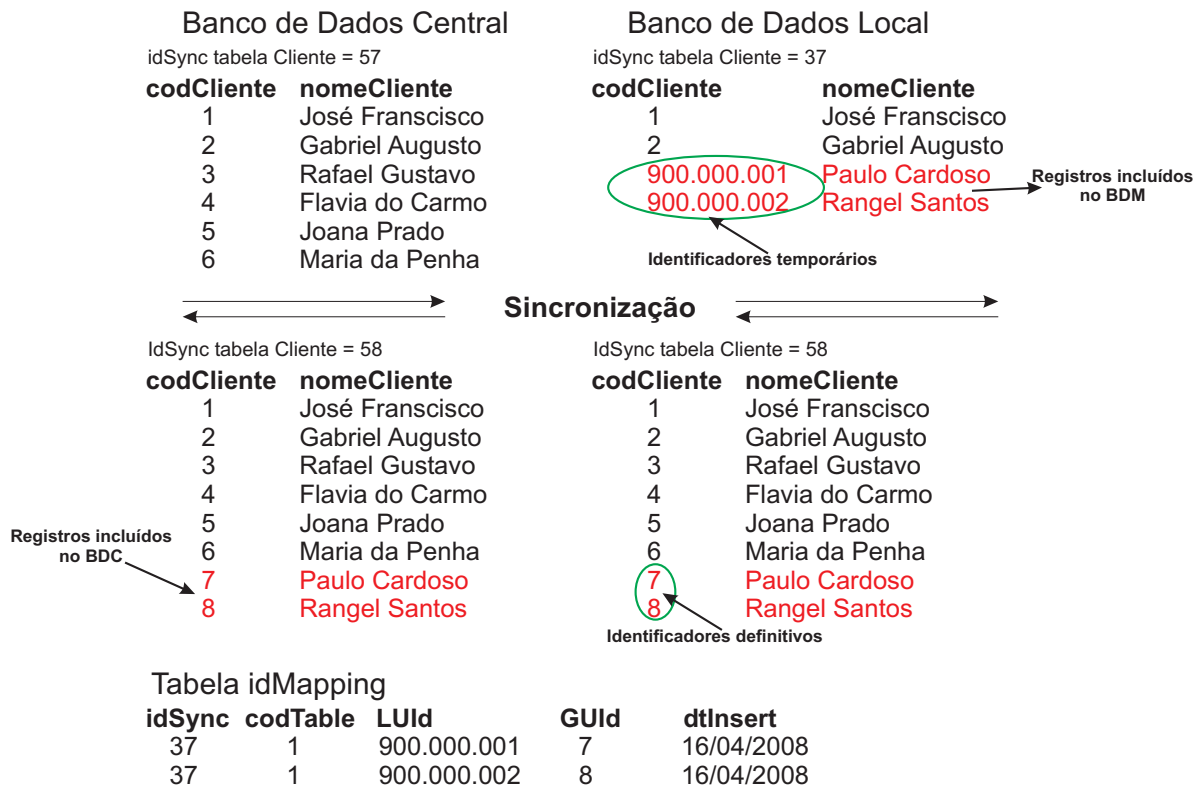
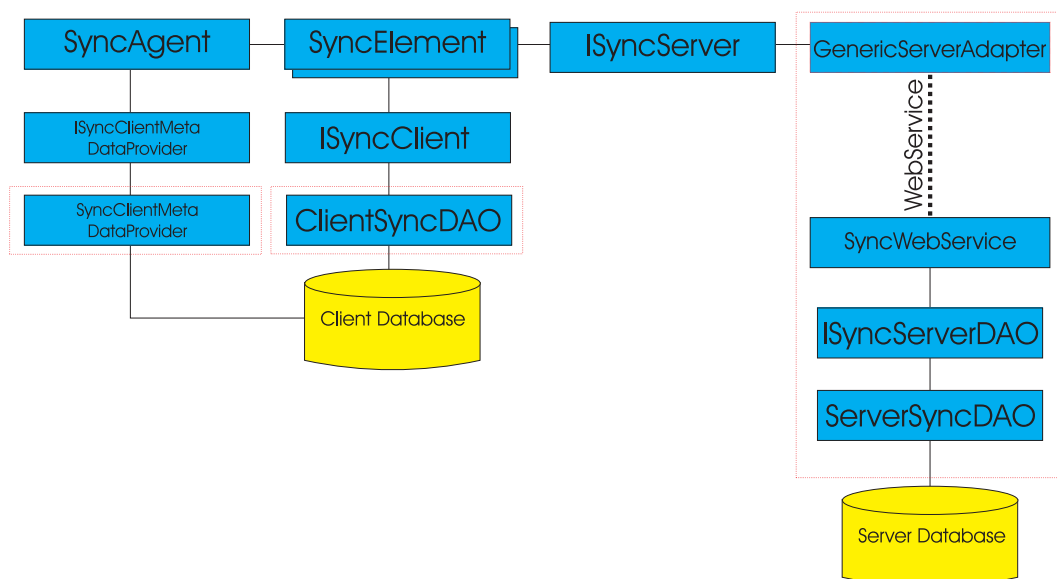


Figura 4.8: Exemplo de conversão de LUID para GUID

e transporte de dados disponíveis na arquitetura, porém, outras soluções podem ser adotadas por meio da criação de novos componentes que implementem as interfaces exigidas pelo *framework*. A seguir serão descritos cada um dos componentes apresentados na figura 4.9.

Figura 4.9: Componentes do *framework* mSync

### 4.5.1 SyncAgent

O *SyncAgent*, ou agente de sincronização, orquestra o processo de sincronização, sendo responsável pelas seguintes tarefas:

- Interagir com cada elemento de sincronização (*SyncElement*) sincronizado;
- Acionar a interface *ISyncClient* para obter e aplicar as modificações na base de dados do dispositivo;
- Acionar a interface *ISyncServer* para obter e aplicar as modificações na base de dados do servidor;
- Gerenciar os metadados utilizados para armazenar as âncoras de sincronização de cada tabela por meio da interface *ISyncClientMetaDataProvider*.

O agente de sincronização também mantém o estado da sessão durante a sincronização e provê informações referentes ao andamento do processo, erros e conflitos para a aplicação no dispositivo.

### 4.5.2 ISyncClientMetaDataProvider e SyncClientMetaDataProvider

O *ISyncClientMetaDataProvider* é a interface por meio da qual o *SyncAgent* obtém e atualiza os metadados referentes às âncoras de sincronização associadas a cada tabela. O *framework* possui uma solução padrão para armazenamento dos metadados que pode ser utilizada em grande parte das aplicações, o *SyncClientMetaDataProvider*. Porém, caso necessário, o programador pode desenvolver sua própria classe para gerenciamento dos metadados, desde que a mesma implemente a interface *ISyncClientMetaDataProvider*.

### 4.5.3 SyncElement

O *SyncElement*, ou elemento de sincronização, encapsula as informações referentes aos parâmetros de sincronização aplicáveis a cada tabela, além de disponibilizar as interfaces *ISyncClient* e *ISyncServer* que permitem que o agente de sincronização tenha acesso aos dados do dispositivo e do servidor durante o processo de sincronização de dados. Deve ser instanciado um *SyncElement* para cada tabela sincronizada.

### 4.5.4 ISyncClient

O *ISyncClient* é a interface que permite que a base de dados local do dispositivo seja acessada durante o processo de sincronização. Suas principais funções são:

- Permitir o acesso aos registros removidos, inseridos e modificados na base de dados local durante o período de desconexão;
- Permitir que as atualizações recebidas do servidor sejam aplicadas na base de dados local;

- Permitir a atualização dos identificadores únicos locais (LUID) e *flags* de estado dos registros durante o processo de sincronização de dados, conforme visto nas Seções 4.3 e 4.4.

#### 4.5.5 ClientSyncDAO

O *ClientSyncDAO* encapsula os métodos para acesso ao banco de dados local. O agente de sincronização, e a própria aplicação no dispositivo, utilizam o *ClientSyncDAO* para interagir com o banco de dados.

O mSync disponibiliza uma solução para o *ClientSyncDAO* que é aplicável a uma grande variedade de situações, mas, caso seja necessário para tratar situações específicas, o programador pode implementar sua própria *ClientSyncDAO* fazendo uma sobrecarga de métodos existentes ou criando uma classe completamente nova, desde que a mesma implemente a interface *ISyncClient* exigida pela arquitetura. As principais funções do *ClientSyncDAO* são:

- Permitir o acesso ao banco de dados do dispositivo para consulta, inserção, atualização e remoção de registros;
- Rastrear as modificações realizadas na base de dados local do dispositivo durante o período de desconexão, utilizando o *flag statusReg*, conforme visto na Seção 4.3.1.

#### 4.5.6 Client Database

O *Client Database* é o banco de dados local armazenado no dispositivo. Este banco de dados deve ser preparado, conforme apresentado nas Seções 4.3.1 e 5.1.2.1, para permitir o rastreamento das modificações realizadas durante o período de desconexão.

#### 4.5.7 ISyncServer

O *ISyncServer* é a interface utilizada pelo agente de sincronização, executado no cliente, para interagir com o servidor da rede fixa. Suas principais funções são:

- Permitir o acesso aos registros removidos, inseridos e modificados na base de dados central durante o período de desconexão;
- Enviar os registros removidos, inseridos e modificados na base de dados local do dispositivo para serem aplicados na base de dados central do servidor;
- Permitir que o dispositivo obtenha informações sobre as âncoras de sincronização, referentes a cada tabela, armazenadas no banco de dados central;
- Disparar eventos que permitam que o agente de sincronização obtenha informações referentes à quantidade de *bytes* enviados e recebidos do servidor, de modo que, caso necessário, a aplicação exiba informações sobre a progressão do processo de sincronização de dados.

#### 4.5.8 GenericServerAdapter

O *GenericServerAdapter* é uma classe, disponível no *framework*, para integração entre os dispositivos e o servidor utilizando *Web services*. O *GenericServerAdapter*, que implementa a interface *ISyncServer*, é executado no dispositivo, se comunicando com o *SyncWebService* existente no servidor. Caso o desenvolvedor deseje utilizar outra tecnologia para o transporte de dados, poderá criar seu próprio adaptador do servidor, desde que o mesmo disponibilize uma interface do tipo *ISyncServer*.

#### 4.5.9 SyncWebService

O *SyncWebService* é um *Web service*, executado no servidor de sincronização, que pode ser utilizado para integração com os dispositivos. Ele é acionado pelo *GenericServerAdapter*, executado no dispositivo, e disponibiliza as funções exigidas pela interface *ISyncServer*.

#### 4.5.10 IServerSyncDAO

O *IServerSyncDAO* é a interface que permite que o *SyncWebService* tenha acesso à base de dados central. Suas principais funções são:

- Permitir o acesso aos registros removidos, inseridos e modificados na base de dados central durante o período de desconexão do dispositivo;
- Permitir que as atualizações recebidas do dispositivo sejam aplicadas na base de dados central;
- Permitir o acesso e atualização dos metadados referentes às âncoras de sincronização.

#### 4.5.11 ServerSyncDAO

O *ServerSyncDAO* encapsula os métodos para acesso ao banco de dados central. Ele implementa a interface *IServerSyncDAO*.

#### 4.5.12 ServerDatabase

O banco de dados do servidor, ou banco de dados central, é o banco de dados corporativo replicado pelos dispositivos para permitir o acesso aos dados em modo desconectado. Este banco de dados deve possibilitar o rastreamento das modificações realizadas durante o período de desconexão dos dispositivos e o mapeamento dos identificadores únicos locais (LUID) em identificadores únicos globais (GUID), conforme visto nas Seções 4.3.2 e 4.4.

### 4.6 Modelo de reuso

Grott et al. (2004) apud Coda e Mayfield (1998), define *framework* como um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas, provendo

uma infra-estrutura genérica de soluções para um conjunto de problemas. *Frameworks* possibilitam reutilizar não só componentes isolados, como também toda a arquitetura de um domínio específico.

Uma característica importante dos *frameworks* é que os métodos, funções e procedimentos definidos pelo desenvolvedor para especializá-los são chamados de dentro do próprio *framework*, ao invés de serem chamados do código da aplicação do desenvolvedor. A reutilização em um *framework* leva a uma inversão de controle. Quando se utiliza uma biblioteca convencional de sub-rotinas, escreve-se o corpo principal da aplicação e chama-se o código que quer reutilizar. Quando se utiliza um *framework*, reutiliza-se o corpo principal e escreve-se o código que este chama (Grott et al. (2004) apud Coda e Mayfield (2000)).

Seguindo esta metodologia, o desenvolvimento de aplicações inicia-se com o entendimento do sistema contido no projeto do *framework* e segue com o detalhamento das particularidades da aplicação específica. Assim, a implementação de uma aplicação a partir do *framework* é feita pela adaptação de sua estrutura de classes, fazendo com que esta inclua as particularidades da aplicação.

Um *framework* consiste em *frozen spots* e *hot spots*. *Frozen spots* definem a arquitetura global de um sistema de *software*, seus componentes básicos e os relacionamentos entre eles. Estes permanecem imutáveis em qualquer instanciação do *framework*. *Hot spots* são projetados para serem genéricos e podem ser adaptados para atender as necessidades específicas da aplicação em desenvolvimento. Quando se cria um sistema de *software* concreto utilizando um *framework*, seus *hot spots* são preenchidos de acordo com as necessidades e requisitos específicos do sistema (Uchôa (1999)). Os *hot spots* do *framework* mSync são destacados na figura 4.9.

Dependendo da forma como os *hot spots* interagem com os *frozen spots*, os *frameworks* podem ser classificados em três grandes grupos: *frameworks* de caixa branca (*whitebox*), de caixa preta (*blackbox*) e de caixa cinza (*graybox*) (Grott et al. (2004)).

Os *frameworks* caixa-branca baseiam-se nos mecanismos de herança e ligação dinâmica (*dynamic binding*) presentes em orientação a objetos. Os recursos existentes em um *framework* deste tipo são reutilizados e estendidos a partir de herança de classes do *framework* e sobrecarga (*overriding*) de métodos “hook” pré-definidos. Métodos “hook” são definidos em interfaces ou classes abstratas e devem necessariamente ser implementados por uma aplicação. Um exemplo de método “hook” no mSync é o *DAOFactory*, pertencente à classe *BaseSyncWebService* (ver A.3.1). O mSync, por ter um modelo de reuso baseado em herança de classes, é classificado como um *framework* do tipo caixa branca.

*Frameworks* caixa-preta são baseados em componentes de software. A extensão da arquitetura e reutilização dos recursos existentes é realizada por meio da definição de componentes adequados a interfaces específicas pré-definidas (Coda e Mayfield (2000)).

Os *frameworks* de caixa-cinza possuem características pertencentes aos dois grupos anteriores, não podendo ser classificados como caixa-branca e nem como caixa-preta.

Segundo Fayad e Schmidt (1997), os *frameworks* também podem ser classificados, de acordo com o seu escopo, nas seguintes categorias:

- **System infrastructure frameworks:** são os *frameworks* utilizados, por exemplo, na construção de sistemas operacionais e interface com o usuário. Sua função é simplificar a construção de infraestrutura destes tipos de sistemas, buscando a portabilidade e eficiência dos mesmos.
- **Middleware Integration frameworks:** são os *frameworks* responsáveis pela comunicação em ambientes e aplicações distribuídas. É a categoria onde melhor se enquadra o *framework* mSync.
- **Enterprise application frameworks:** são os *frameworks* voltados para aplicações comerciais e para a área dos negócios. Este tipo de *framework* tem o objetivo de construir aplicações para o usuário final.

## 4.7 Sincronização de dados

O processo de sincronização de dados proposto no mSync é dividido em duas etapas:

- Sincronização sentido dispositivo móvel para servidor fixo (DM -> SF).
- Sincronização sentido servidor fixo para dispositivo móvel (SF -> DM).

O mSync trata estas duas etapas de forma independente, permitindo que sejam definidos tipos de sincronização diferentes para cada uma delas. Os tipos de sincronização suportados pela arquitetura são:

- **Refresh:** Quando utilizado no sentido DM -> SF transmite todos os registros, independente de terem sido ou não modificados no dispositivo, para o servidor. Quando utilizado no sentido SF -> DM, exclui todos os registros existentes na base de dados local do dispositivo e realiza uma nova cópia com as informações recebidas do servidor.
- **Incremental:** Independente do sentido da sincronização, envia somente as informações modificadas durante o período de desconexão do dispositivo.
- **None:** Não realiza a sincronização. Utilizado para desativar a sincronização em um dos sentidos quando o objetivo é realizar a sincronização apenas no sentido DM -> SF ou SF -> DM.

A figura 4.10 exibe o diagrama de seqüência do início do processo de sincronização. Os passos exibidos na figura são detalhados a seguir:

- No passo 1 a aplicação cliente atribui o *SyncClientMetadataProvider* que o *SyncAgent* deve utilizar para obter as informações referentes às âncoras de sincronização armazenadas no dispositivo.

- No passo 2 a aplicação cliente cria um *SyncElement* para cada tabela que será sincronizada. O *SyncElement* é associado às interfaces *ISyncClient* e *ISyncServer*, que serão utilizadas pelo agente de sincronização para acessar o banco de dados do dispositivo e do servidor, respectivamente.
- No passo 3 os elementos de sincronização criados no passo 2 são adicionados ao agente de sincronização.
- No passo 4 a aplicação cliente inicia o processo de sincronização acionando o método *BeginSynchronize*.
- No passo 5 o agente de sincronização comanda, para cada elemento de sincronização, o processo de sincronização sentido dispositivo móvel para servidor fixo.
- No passo 6 o agente de sincronização comanda, para cada elemento de sincronização, o processo de sincronização sentido servidor fixo para dispositivo móvel.

A Seção 4.7.1 detalha o processo de sincronização sentido dispositivo para servidor e, a Seção 4.7.2 o processo de sincronização sentido servidor para dispositivo.

#### 4.7.1 Sincronização sentido dispositivo móvel para servidor fixo

A sincronização no sentido dispositivo para servidor fixo é a primeira a ser realizada durante o processo de sincronização de dados. Durante esta etapa, as informações do dispositivo são enviadas para o servidor central. A figura 4.11 exibe o diagrama de seqüência do processo executado para cada elemento de sincronização. Para simplificar o diagrama, não estão sendo exibidas as interações entre as interfaces *ISyncClient*, *ISyncServer*, *ISyncClientMetaDataProvider* e os objetos que de fato as implementam, que são, respectivamente, o *ClientSyncDAO*, *GenericServerAdapter* e *SyncClientMetaDataProvider*. Os passos apresentados na figura 4.11 são detalhados a seguir:

- No passo 1, o *SyncAgent* obtém o valor da âncora referente a última sincronização (*idSyncInsert*), relacionada à tabela sincronizada, com envio de novos registros para o servidor.
- No passo 2, o *SyncAgent*, acionando a interface *ISyncClient*, obtém a lista de registros inseridos no banco de dados local durante o período de desconexão.
- No passo 3, os registros obtidos no passo 2 são enviados para serem aplicados no servidor por meio da interface *ISyncServer*. O valor do *idSyncInsert*, obtido no passo 1, é passado como parâmetro, permitindo que, conforme detalhado na Seção 4.4, o servidor verifique se o registro enviado é realmente novo, e deve ser inserido no servidor, ou se o registro já foi inserido no servidor em uma sincronização anterior interrompida inesperadamente.
- No passo 4, o *ISyncClient* é acionado para que os LUIDs, gerados no dispositivo, sejam substituídos pelos GUIDs, gerados no servidor, conforme detalhado na Seção 4.4. O

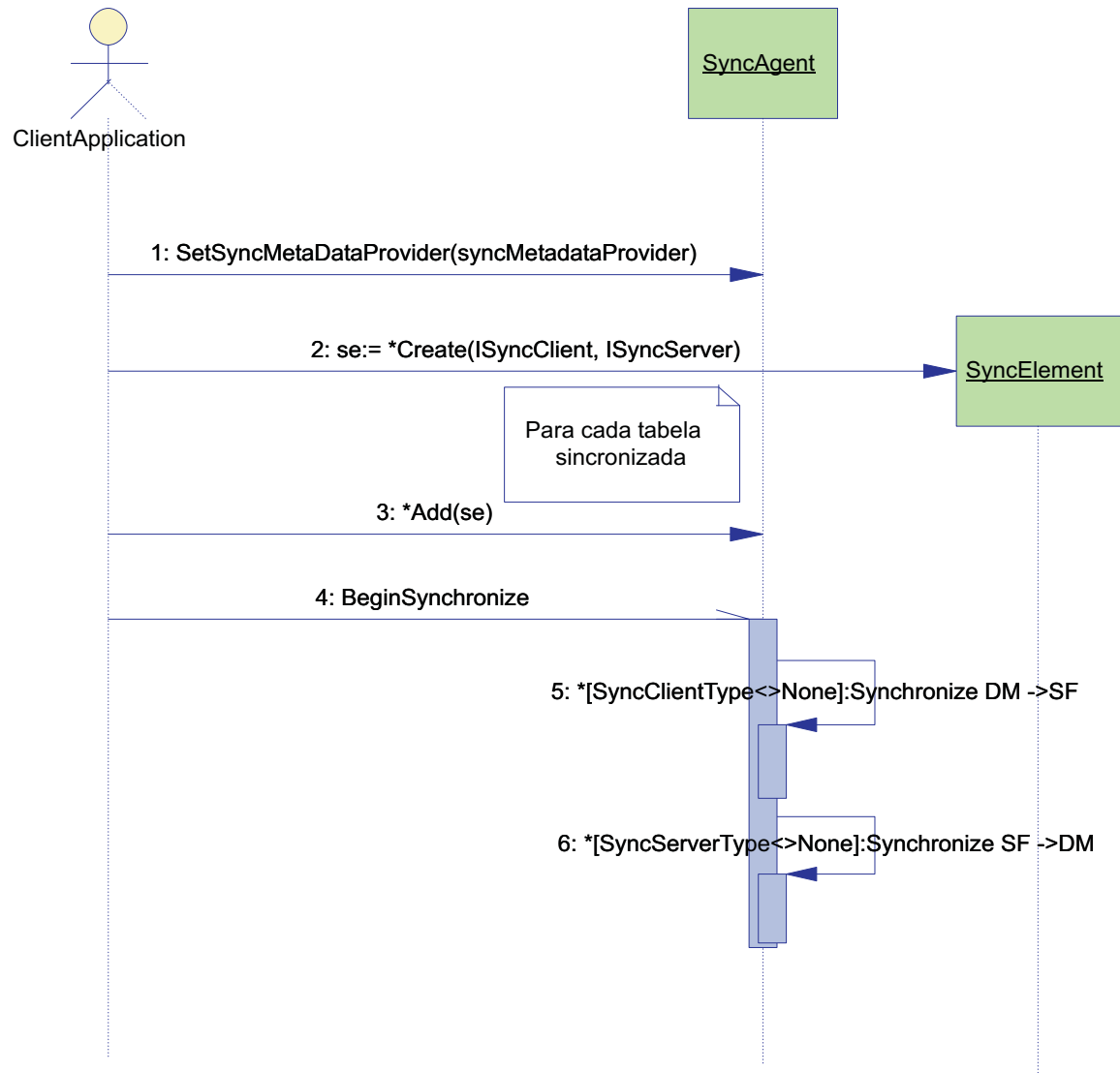


Figura 4.10: Diagrama de seqüência do início do processo de sincronização

mapeamento entre os identificadores locais dos registros (LUIDs), gerados no dispositivo, e os identificadores globais, gerados no servidor (GUID), são obtidos por meio do resultado da função *Insert*, invocada no passo 3.

- No passo 5, o *SyncAgent* aciona a interface *ISyncServer* para obter um novo valor para a âncora de sincronização (idSync).
- No passo 6, a âncora de sincronização obtida no passo 5 é armazenada no dispositivo.
- No passo 7, o *SyncAgent*, por meio da interface *ISyncClient*, obtém a lista de registros modificados no banco de dados local durante o período de desconexão.
- Nos passos 8 e 9, caso o tipo de sincronização seja *refresh*, os registros já sincronizados são obtidos e adicionados à lista de registros modificados, pois, neste tipo de sincronização, todos os registros do dispositivo devem ser transmitidos para o servidor.

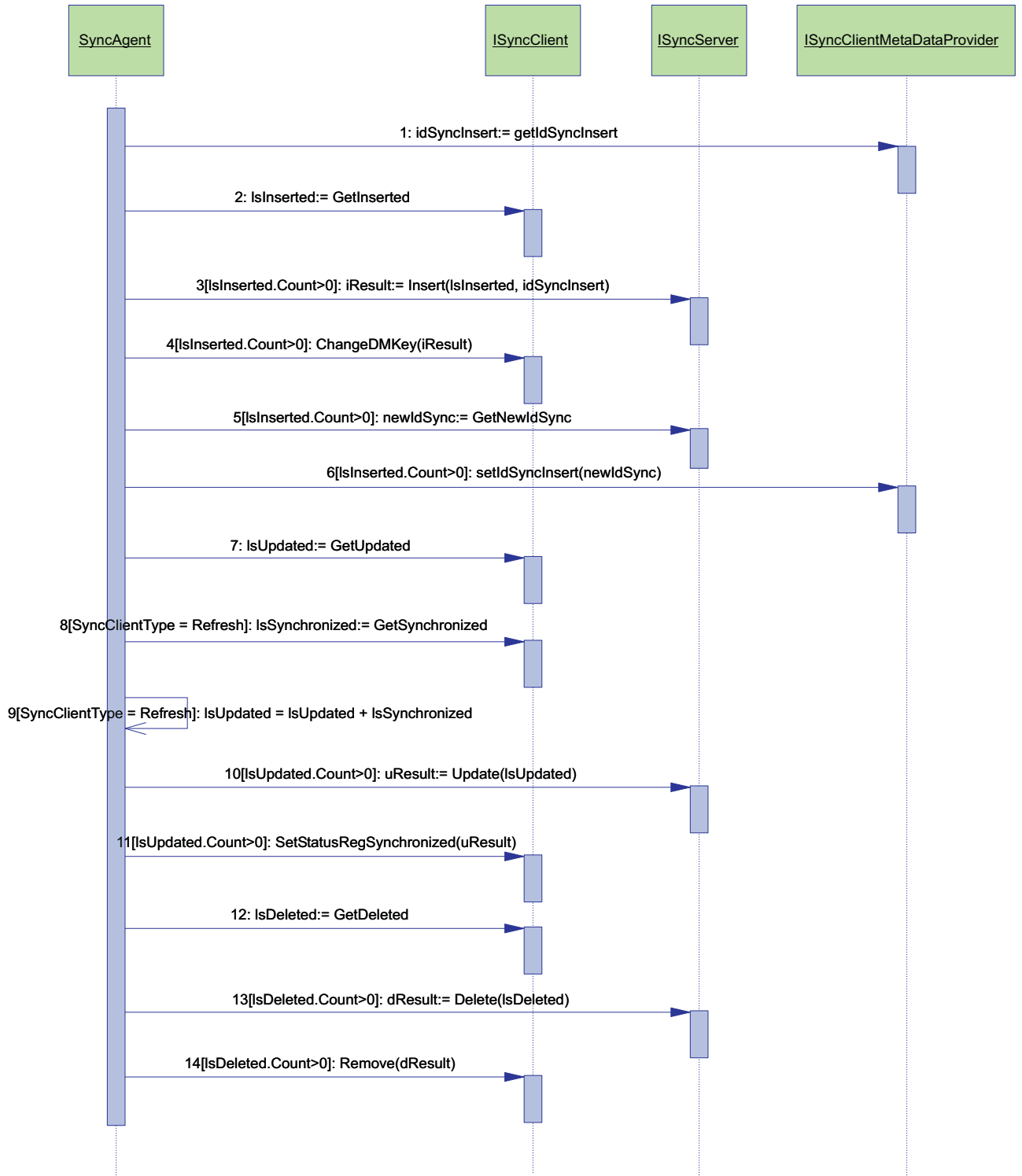


Figura 4.11: Diagrama de seqüência do processo de sincronização sentido DM -> SF

- No passo 10, os registros modificados são transmitidos para serem aplicados no servidor por meio da interface *ISyncServer*.
- No passo 11, o *flag statusReg* dos registros modificados enviados para o servidor são

atualizados para o estado *Synchronized*, conforme detalhado na Seção 4.3.1.

- No passo 12, é obtida a lista de registros excluídos na base local do dispositivo durante o período de desconexão.
- No passo 13, os registros excluídos são enviados para serem aplicados no servidor.
- No passo 14, os registros excluídos no servidor são fisicamente excluídos da base de dados local do dispositivo por meio do método *Remove* da interface *ISyncClient*.

### Tratamento de Falhas

O ponto mais crítico em caso de falhas, no fluxo de sincronização apresentado na figura 4.11, é o passo 3. Se a comunicação for interrompida inesperadamente durante a operação em que os novos registros criados no dispositivo são transmitidos para o servidor, o dispositivo não poderá determinar, com certeza, se os mesmos foram ou não inseridos com sucesso na base de dados central. Neste caso, na próxima tentativa de sincronização, estes registros serão transmitidos para o servidor e, se eles já tiverem sido inseridos no banco de dados central na última tentativa, serão criados em duplicidade.

O *framework* mSync, conforme detalhado na Seção 4.4, propõe um esquema para evitar a situação descrita. Antes que qualquer registro seja inserido no banco de dados central, o *ServerSyncDAO* (ver Seção 4.5.11) verifica se, na tabela de controle idMapping (figura 4.7), existe uma entrada associada à chave idSync, codTable e LUId referente ao registro sincronizado. Caso a entrada exista, significa que, na sincronização anterior, o registro foi transmitido e inserido com sucesso no banco de dados central, e, neste caso, o registro não é inserido novamente, mas apenas atualizado utilizando como chave o identificador único global (GUID) correspondente armazenado na tabela idMapping. A situação contrária, quando a entrada na tabela idMapping não é localizada, significa que o registro é realmente novo, devendo ser inserido no banco de dados central e a entrada correspondente registrada na tabela idMapping.

Caso a comunicação entre o dispositivo e o servidor seja interrompida em outros pontos, a consistência das informações também não é comprometida. Conforme pode ser observado nos passos 11 e 14 da figura 4.11, as ações de atualizar os *flags* de estado dos registros e remover fisicamente as linhas marcadas para serem excluídas, respectivamente, só são efetuadas depois que o dispositivo recebe do servidor o resultado da operação indicando se a mesma foi ou não concluída com sucesso. A interrupção da comunicação durante os passos 10 e 13, quando os registros modificados e excluídos no dispositivo são transmitidos para o servidor, pode fazer com que as modificações e exclusões já aplicadas no servidor sejam retransmitidas durante a próxima tentativa, porém essa situação não traz problemas relacionados à consistência dos dados.

Para simplificar o diagrama de seqüência, os eventos utilizados pelo *framework* para notificar a aplicação sobre a ocorrência de erros e conflitos não foram apresentados na figura 4.11. A Seção 4.2 detalha como estas situações são detectadas e resolvidas pelo *framework*.

#### 4.7.2 Sincronização sentido servidor fixo para dispositivo móvel

Esta Seção detalha o processo de sincronização, sentido servidor fixo para dispositivo móvel, proposto no *framework*. Durante esta etapa, as informações do servidor são aplicadas no banco de dados local do dispositivo. A figura 4.12 exibe o diagrama de seqüência do processo executado para cada elemento de sincronização. Novamente, para simplificar o diagrama, não são exibidas as interações entre as interfaces *ISyncClient*, *ISyncServer*, *ISyncClientMetaDataProvider* e os objetos que de fato as implementam, que são, respectivamente, o *ClientSyncDAO*, *GenericServerAdapter* e *SyncClientMetaDataProvider*. Os passos apresentados na figura 4.12 são detalhados a seguir.

- No passo 1, o *SyncAgent* obtém, por meio da interface *ISyncClientMetaDataProvider*, o valor da âncora de sincronização (*idSyncStart*) armazenada no dispositivo referente à última sincronização sentido SF -> DM concluída com sucesso.
- No passo 2, o *SyncAgent* utiliza a interface *ISyncServer* para obter do servidor um novo valor para a âncora de sincronização.
- No passo 3, caso o tipo de sincronização seja *refresh*, o *idSyncStart* é atribuído com o valor 0. Esta operação é efetuada para garantir que, neste tipo de sincronização, todos os registros do servidor sejam enviados para o dispositivo e não somente aqueles modificados durante o período de desconexão, conforme regra detalhada na Seção 4.3.2.
- No passo 4, o *SyncAgent* aciona a interface *ISyncServer* para obter do servidor os registros referentes à tabela sincronizada. Caso a sincronização seja do tipo *refresh*, serão obtidos os registros do banco de dados central, independente deles terem sido ou não modificados durante o período de desconexão. Se a sincronização for do tipo incremental, serão obtidos apenas os registros modificados ou inseridos entre a última sincronização e a sincronização atual, ou seja, aqueles cujo o *idSyncChange* seja maior ou igual ao *idSyncStart* e menor do que o *idSyncEnd*, conforme detalhado na Seção 4.3.2.
- No passo 5, caso o tipo de sincronização seja *refresh*, os dados armazenados localmente no dispositivo são excluídos. Esta operação é realizada porque, neste tipo de sincronização, as informações disponíveis no dispositivo devem ser removidas antes que uma nova cópia seja obtida do servidor.
- No passo 6, os registros obtidos do servidor são aplicados no banco de dados local do dispositivo. É realizada uma operação de *InsertOrUpdateServerSync* onde, caso o registro não exista na base dados local, ele é criado e, caso exista, é atualizado.
- No passo 7, para sincronizações do tipo incremental, o *SyncAgent* obtém do servidor os registros excluídos durante o período de desconexão.
- No passo 8, os registros obtidos no passo 7 são removidos da base de dados local do dispositivo.

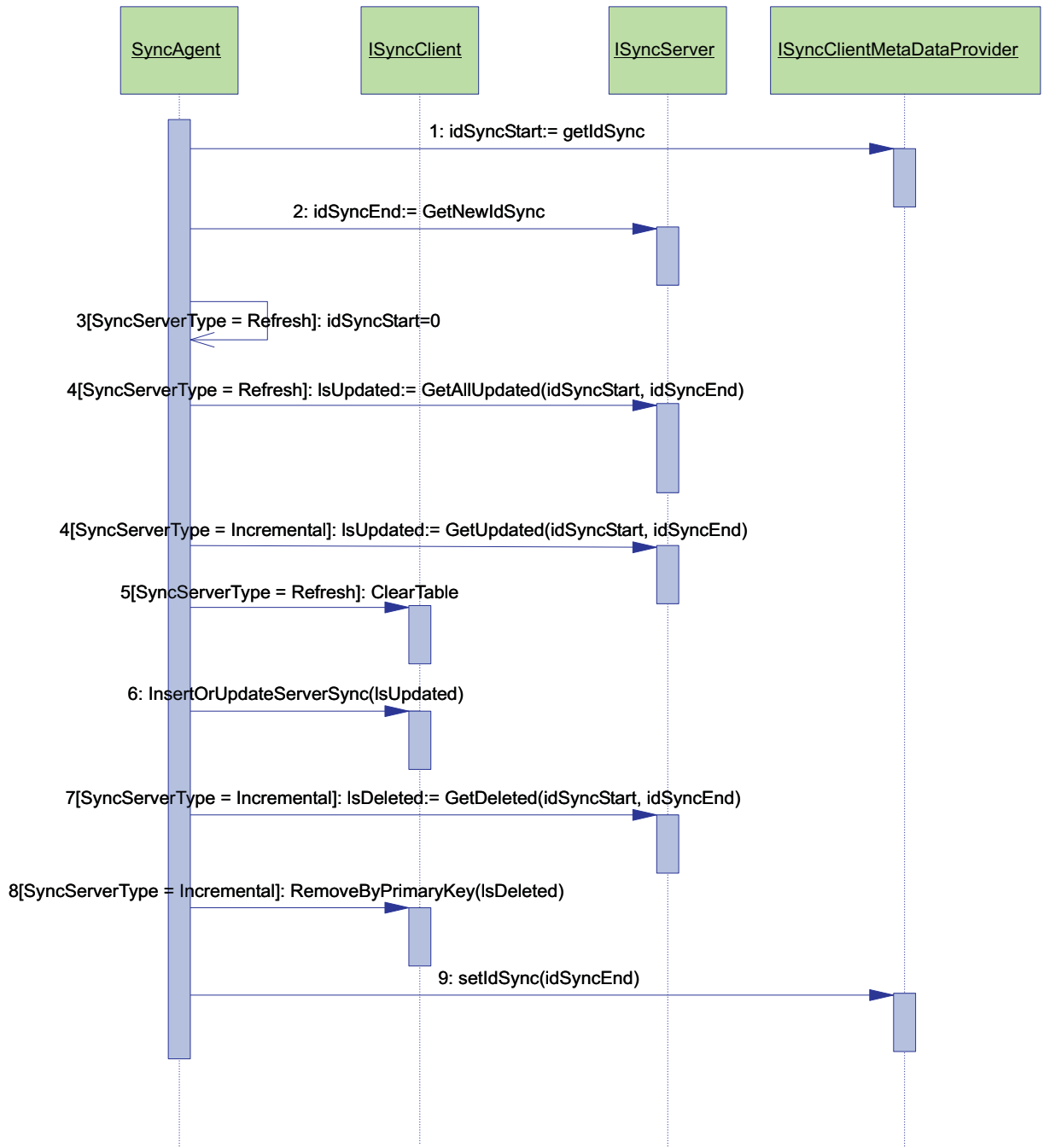


Figura 4.12: Diagrama de seqüência do processo de sincronização sentido SF -> DM

- No passo 9, a nova âncora de sincronização, obtida no passo 1, é armazenada no dispositivo, de forma que, na próxima sincronização, este seja o valor do *idSyncStart* utilizado para determinar quais informações devem ou não ser transmitidas para o dispositivo nas sincronizações do tipo incremental.

### Tratamento de Falhas

A proposta de fluxo de sincronização apresentada na figura 4.12 é robusta, não comprometendo a consistência dos dados em caso de falhas de comunicação e interrupção inesperada da sincronização. O valor da âncora de sincronização (*idSync*) é persistida apenas no final do processo, no passo 9, garantindo que, mesmo que a sincronização seja interrompida, as modificações que ainda não foram aplicadas no dispositivo sejam recebidas na próxima tentativa. Além disto, os registros modificados ou inseridos no servidor durante o período de desconexão são aplicados no dispositivo da mesma forma, por meio do método *InsertOrUpdateServerSync*, acionado no passo 6. Este procedimento contribui para segurança do processo, pois, caso uma sincronização seja abortada e, na tentativa seguinte, um registro inserido no banco de dados central seja transmitido para o dispositivo, o processo funcionará corretamente, independente se, na última sincronização, interrompida inesperadamente, o registro tenha ou não sido inserido na base de dados local do dispositivo.

Novamente, para simplificar o diagrama, os eventos utilizados pelo *framework* para notificar a aplicação sobre a ocorrência de erros e conflitos não foram apresentados no diagrama da figura 4.12. A Seção 4.2 detalhou como estas situações são detectadas e resolvidas pelo *framework*.

## 4.8 Conclusões

Este Capítulo introduziu a arquitetura mSync. Primeiro foi apresentado o cenário de aplicação do *framework*. Depois foram detalhadas as soluções propostas para tratar problemas comuns ao processo de sincronização de dados, cujos conceitos foram introduzidos no Capítulo 3. O Capítulo detalhou a forma com que as âncoras de sincronização são utilizadas no *framework*, assim como o esquema adotado para detecção / resolução de conflitos e rastreamento das modificações realizadas durante o período de desconexão, tanto nos dispositivos quando no banco de dados central.

O Capítulo apresentou um novo esquema para geração de identificadores únicos em ambientes distribuídos, que possui vantagens em relação a outras abordagens encontradas na literatura e em projetos comerciais. Foram apresentados os principais componentes do *framework* mSync. No Capítulo também foi explicitado o modelo de reuso e classificação do *framework* mSync. Por fim, o Capítulo foi concluído apresentando o fluxo proposto para sincronização de dados nos sentidos dispositivo para servidor e servidor para dispositivo, com destaque para a forma como o *framework* se comporta em situações de falha de comunicação, comuns no ambiente de mobilidade.

No próximo Capítulo serão apresentados dois estudos de uso. O primeiro é uma aplicação simples de cadastrado de clientes e pedidos, desenvolvida com o objetivo didático de se exemplificar a utilização do *framework*. O segundo estudo de caso é uma aplicação real de fiscalização de obras, empreendimentos e serviços utilizada pelo Conselho Regional de Arquitetura, Engenharia e Agronomia do estado de Minas Gerais (Crea-MG), que é apresentado com o objetivo de se comprovar a abrangência e utilidade do trabalho proposto.

## Capítulo 5

# Estudos de Caso

Este Capítulo apresenta dois estudos de caso utilizados para demonstrar o funcionamento e utilidade do *framework* proposto. O primeiro estudo de uso, apresentado na Seção 5.1, trata de uma aplicação simples de cadastro de clientes e pedidos, desenvolvida com o objetivo didático de demonstrar o funcionamento do *framework*.

O segundo estudo de caso, detalhado na Seção 5.2, apresenta uma aplicação real de fiscalização de obras, empreendimentos e serviços que é utilizada pelo Conselho Regional de Engenharia, Arquitetura e Agronomia de Minas Gerais (Crea-MG). A Seção 5.3, que finaliza o Capítulo, apresenta os resultados obtidos.

### 5.1 Estudo de caso 1: Cadastro de Clientes e Pedidos

O objetivo deste estudo de caso é demonstrar o funcionamento e utilização do *framework* mSync por meio de uma aplicação simples de cadastro de clientes.

No exemplo são sincronizadas informações referentes a produtos, clientes, pedidos e itens de pedidos. A Seção 5.1.1 apresenta a preparação do banco de dados central e do servidor de sincronização e, a 5.1.2, trata da preparação da aplicação executada nos dispositivos.

#### 5.1.1 Preparação do Banco de Dados Central e Servidor de Sincronização

Esta Seção apresenta os procedimentos necessários para preparação do banco de dados central e do servidor de sincronização de dados.

##### 5.1.1.1 Preparação do Banco de Dados Central

O exemplo utiliza o SGBD *Microsoft SQL Server* (SQL Server 2005 Home (2008)) como banco de dados central. O primeiro passo na preparação do banco de dados é a criação das tabelas de controle AppTables, IdMapping e TombStone, apresentadas nas Seções 4.1, 4.3.2 e 4.4, respectivamente. A figura 5.1 apresenta a estrutura das tabelas do banco de dados central que serão replicadas nos dispositivos móveis.

A tabela produtos é modificada apenas no banco de dados central, sendo disponibilizada nos dispositivos para consulta. Os clientes, pedidos e itens de pedidos podem ser inseridos,

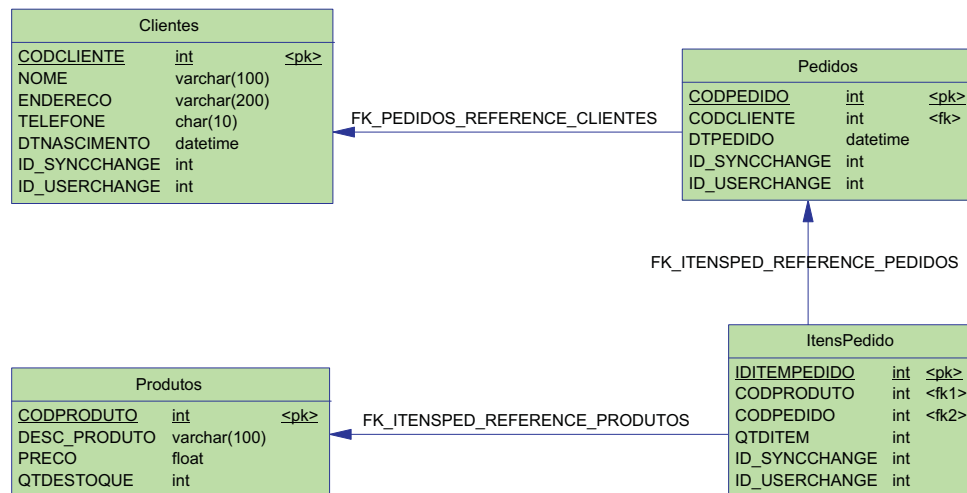


Figura 5.1: Caso 1: Tabelas do servidor fixo replicadas nos dispositivos

excluídos e atualizados no banco de dados central e nos dispositivos, sendo que, durante a sincronização, as informações modificadas nas duas plataformas devem conciliadas.

A tabela AppTables, conforme detalhado na Seção 4.1, é quem armazena o valor atualizado das âncoras de sincronização, devendo ser povoada com registros referentes a cada tabela que irá suportar a sincronização do tipo incremental, como, para este exemplo, é apresentado na figura 5.2.

| Tabela appTables |             |        |
|------------------|-------------|--------|
| CodTable         | TableName   | IdSync |
| 1                | Clientes    | 0      |
| 2                | Pedidos     | 0      |
| 3                | ItensPedido | 0      |

Figura 5.2: Caso 1: Povoamento inicial da tabela AppTables

O campo ID\_SYNCCHANGE, presente nas tabelas Clientes, Pedidos e ItensPedido, apresentadas na figura 5.1, é utilizado como âncora de sincronização. Sempre que um registro é inserido ou modificado no banco de dados central, o valor deste campo deve ser atribuído com o valor do IdSync atual, armazenado na tabela AppTables, referente à tabela em questão.

Esta tarefa pode ser simplificada pelo uso de um valor padrão para o campo e gatilhos de atualização de dados. A figura 5.3 exibe o *script* de criação da tabela CLIENTES, evidenciando o valor padrão do campo ID\_SYNCCHANGE, definido como “getIdSync(‘CLIENTES’)”. A função *getIdSync*, apresentada na figura 5.4, recebe como parâmetro o nome da tabela e retorna o valor da âncora de sincronização correspondente. A utilização deste valor padrão garante que todo registro inserido no banco de dados central seja atribuído com um valor de âncora de sincronização atualizado.

A atualização do valor da âncora no momento em que o registro é modificado pode ser implementado por meio de gatilho semelhante ao apresentado na figura 5.5, que é utilizado,

```

/*=====*/
/* Table: CLIENTES */
/*=====*/
create table CLIENTES (
    CODCLIENTE          int          identity,
    NOME                 varchar(100) null,
    ENDERECO             varchar(200) null,
    TELEFONE             char(10)    null,
    DTNASCIMENTO         datetime   null,
    ID_SYNCCHANGE        int          null default dbo.getIdSync('CLIENTES'),
    ID_USERCHANGE        int          null,
    constraint PK_CLIENTES primary key (CODCLIENTE)
)
go

```

Figura 5.3: Caso 1: *Script* de criação da tabela Clientes

```

CREATE Function getIdSync(@iTableName varchar(30))
RETURNS int
AS
BEGIN
    RETURN (select idsync from apptables where tablename=@iTableName)
END
go

```

Figura 5.4: Caso 1: *Script* de criação da função *getIdSync*

neste exemplo, para atualizar o valor do ID\_SYNCCHANGE da tabela clientes quando a mesma é modificada.

```

/* Update trigger "CLIENTES_UPDATE" for table "CLIENTES" */
create trigger CLIENTES_UPDATE on CLIENTES for update as
begin
    UPDATE CLIENTES SET ID_SYNCCHANGE=DBO.GETIDSYNC('CLIENTES') WHERE CODCLIENTE
    IN (SELECT CODCLIENTE FROM INSERTED)
end
go

```

Figura 5.5: Caso 1: Gatilho de atualização da tabela Clientes

O rastreamento das operações de exclusão de registros no banco de dados central, de forma que as mesmas possam ser replicadas nos dispositivos, é realizada por meio da tabela de controle *tombStone*. Conforme detalhado na Seção 4.3.2, sempre que um registro é excluído de alguma tabela que suporte sincronização incremental, a operação é registrada por meio da inserção de uma nova linha na tabela *tombStone*. Este mecanismo também pode ser implementado por meio de um gatilho de exclusão, semelhante ao exibido na figura 5.6, que é utilizado para registrar as exclusões realizadas na tabela Clientes.

Além das tabelas e funções já apresentadas, devem ser criadas duas outras funções no banco de dados central: *getCodTable* e *newIdSync*. Estas funções são acionadas durante o processo de sincronização para permitir que o *syncAgent* obtenha o código das tabelas sincronizadas e obtenha / incremente o valor do contador / âncoras de sincronizações (*idSync*),

```

/* Delete trigger "CLIENTES_DELETE" for table "CLIENTES" */
create trigger CLIENTES_DELETE on CLIENTES for delete as
begin
    SET NOCOUNT ON
    DELETE FROM TOMBSTONE
        WHERE CODTABLE=(DBO.getCodTable('CLIENTES')) AND
        GUID IN (SELECT CODCLIENTE FROM deleted)
    INSERT INTO TOMBSTONE(CODTABLE, GUID, IDSYNC)
    SELECT DBO.getCodTable('CLIENTES'), CODCLIENTE, DBO.getIdSync('CLIENTES') FROM deleted
    SET NOCOUNT OFF
end
go

```

Figura 5.6: Caso 1: Gatilho de exclusão da tabela Clientes

respectivamente. O *script* de criação destas funções é apresentado nas figuras 5.7 e 5.8.

```

CREATE Function getCodTable(@iTableName varchar(30))
RETURNS int
AS
BEGIN
    RETURN (select codTable from apptables where tablename=@iTableName)
END
go

```

Figura 5.7: *Script* de criação da função *getCodTable*

```

CREATE procedure [dbo].[newIdSync] ( @iTableName char(30) ) as

DECLARE @Numero int
begin transaction

    select @Numero = (select idsync from apptables where tablename=@iTableName)

    update apptables set idsync = @Numero+1 where tablename=@iTableName

    select @Numero+1

commit
go

```

Figura 5.8: *Script* de criação da função *newIdSync*

#### 5.1.1.2 Definição dos Modelos e Objetos de Acesso a Dados no Servidor

O mSync utiliza o *framework* GDA (Sales (2008)), apresentado na Seção 3.9, como suporte para acesso e persistência de dados. O GDA implementa a técnica de programação conhecida como mapeamento objeto - relacional ou, do inglês *object relational mapping* (ORM) (Keller (1998)). O mapeamento entre os objetos que representam os registros do banco de dados e o banco de dados físico é realizado na própria classe do modelo, conforme apresentado na

figura 5.9. O atributo *PersistenceClass* indica o nome da tabela do banco de dados ao qual o modelo está associado e, o atributo *PersistenceProperty*, utilizado nas propriedades da classe, indica o nome e tamanho do campo no banco de dados ao qual a propriedade está associada.

```
[PersistenceClass("Clientes")]
public class Clientes : BaseServerSyncModel
{
    private int m_codCliente;
    [PersistenceProperty("codCliente", PersistenceParameterType.IdentityKey)]
    public int CodCliente
    {
        get { return m_codCliente; }
        set { m_codCliente = value; }
    }

    private string m_nome;
    [PersistenceProperty("nome", 100)]
    public string Nome...

    private string m_endereco;
    [PersistenceProperty("endereco", 200)]
    public string Endereco...

    private string m_telefone;
    [PersistenceProperty("telefone", 10)]
    public string Telefone...

    private DateTime m_dtnascimento;
    [PersistenceProperty("dtnascimento")]
    public DateTime Dtnascimento...
}
```

Figura 5.9: Caso 1: Classe de modelo da tabela Clientes utilizada no servidor

Além do modelo de dados, deve ser definido um objeto de acesso a dados, do inglês *Data Access Object* (DAO) (Coda e Mayfield (2000)), associado a cada tabela sincronizada. A figura 5.10 apresenta a definição da DAO referente à tabela Clientes.

```
public class ClientesDAO : BaseSyncDAO<Clientes>
{
    public ClientesDAO() : base(DBUtils.DbProvider)
    {
    }
}
```

Figura 5.10: Caso 1: DAO da tabela clientes utilizada no servidor

Os modelos de dados e objetos de acesso a dados, referentes às tabelas que suportam sincronização incremental, que, neste exemplo, são as tabelas Clientes, Pedidos e ItensPedido, devem ser especializações das classes *BaseServerSyncModel* e *BaseSyncDAO*, respectivamente (ver Apêndice A.3.3). O modelo associado à tabela Produtos, que suporta apenas a sincronização do tipo *refresh*, não precisa ser uma especialização de nenhuma classe do *framework*. A

DAO referente à tabela produtos deve ser uma especialização da classe *BaseSimpleSyncDAO* (ver Apêndice A.3.3).

#### 5.1.1.3 Definição do Web service de Sincronização

O *Web service* de sincronização deve ser uma especialização da classe *BaseSyncWebService*, apresentada no Apêndice A.3.1. A figura 5.11 exibe a classe *Service*, utilizada, neste exemplo, como *Web service* de sincronização. Como pode ser observado na figura, apenas dois métodos precisam ser sobrescritos: *DAOFactory* e *GetUpdated*. O método *DAOFactory*, durante o processo de sincronização, é acionado pelo *Web service* para obter o DAO (*Data Access Object*) utilizado para interagir com o banco de dados central. A implementação da classe *DAOServerFactory*, utilizada no exemplo, é exibida na figura 5.12.

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : BaseSyncWebService
{
    public Service ()
    {
    }

    public override ISyncServerDAO DAOFactory(string ClassName)
    {
        return DAOServerFactory.FactoryMethod(ClassName);
    }

    [WebMethod]
    [XmlInclude(typeof(Clientes))]
    [XmlInclude(typeof(Produtos))]
    [XmlInclude(typeof(Pedidos))]
    [XmlInclude(typeof(ItensPedido))]
    public override object[] GetUpdated(SyncSession syncSession, string className)
    {
        return base.GetUpdated(syncSession, className);
    }
}
```

Figura 5.11: Web service utilizado no estudo de caso 1

O método *GetUpdated* precisa ser sobrescrito porque o *XmlSerializer*, utilizado pelo *framework* .NET para serializar e deserializar objetos, exige que as classes que serão serializadas sejam conhecidas previamente, o que é realizado por meio do atributo *XmlInclude*, utilizado no método *GetUpdated*, apresentado na figura 5.11. O *Web service* é hospedado em uma pasta virtual do IIS 6.0 (*Microsoft Internet Information Services*).

#### 5.1.2 Preparação do Dispositivo Móvel

Esta Seção detalha os passos necessários para instanciação do *framework* mSync e preparação da aplicação de sincronização de dados que é executada nos dispositivos ubíquos.

```

public class DAOConnectionFactory
{
    /// <summary>
    /// Retorna uma instancia de ISyncServer
    /// </summary>
    /// <param name="ClassName">Nome da classe a ser instanciada</param>
    /// <returns>Instancia da classe</returns>
    public static ISyncServerDAO FactoryMethod(string ClassName)
    {
        switch (ClassName)
        {
            case "Clientes" :
                return new ClientesDAO();
            case "Produtos":
                return new ProdutosDAO();
            case "Pedidos":
                return new PedidosDAO();
            case "ItensPedido":|
                return new ItensPedidoDAO();
            default:
                throw new InvalidOperationException("Classe " + ClassName +
                                                    " não conhecida.");
        }
    }
}

```

Figura 5.12: *DAO Factory* utilizado no estudo de caso 1

#### 5.1.2.1 Preparação do Banco de Dados do Dispositivo

Este exemplo utiliza o SGBD *Microsoft Sql Server Mobile* como banco de dados local dos dispositivos. A figura 5.13 exibe as tabelas utilizadas no banco de dados móvel.

As tabelas Clientes, Pedidos e ItensPedido possuem um campo chamado StatusReg que, conforme detalhado na Seção 4.3.1, é utilizado para registrar as modificações realizadas no banco de dados local. A tabela Produtos está disponível apenas para consulta nos dispositivos e, por este motivo, não precisa do campo de controle.

A tabela AppTable do dispositivo possui dois campos para armazenar as âncoras de sincronização: o *idSync* e o *idSyncInsert*. O *idSync* é a âncora utilizada para determinar quais modificações do servidor devem ser aplicadas no dispositivo. O *idSyncInsert*, como detalhado na Seção 4.7.1, é utilizado para evitar que, em situações de falha de comunicação, um mesmo registro criado no dispositivo seja inserido em duplicidade no banco de dados central.

O banco de dados *Microsoft Sql Server Mobile* (SQL Server Mobile (2008)) oferece suporte para relacionamentos do tipo *update cascade* (Vieira (2007)). Os relacionamentos entre as tabelas Clientes e Pedidos e Pedidos e ItensPedido devem possuir esta característica para garantir que, quando o identificador local (LUID) de um registro em uma destas tabelas for substituído pelo identificador global (GUID), conforme detalhado na Seção 4.4, a operação seja propagada para as tabelas que utilizem este identificador como chave estrangeira. Caso

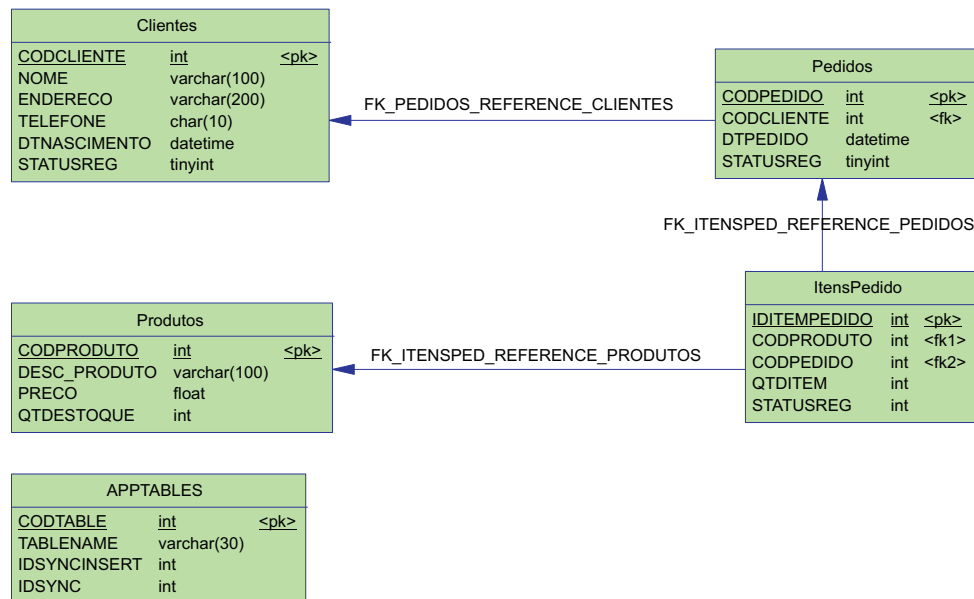


Figura 5.13: Caso 1: Tabelas utilizadas no banco de dados móvel

seja utilizado um banco de dados que não ofereça suporte para relacionamentos do tipo *update cascade*, o evento *AfterInsertServer*, disparado pelo *SyncAgent* (ver Apêndice A.2.8), pode ser utilizado para propagar a atualização dos identificadores únicos dos registros nas outras tabelas que o utilizam como chave estrangeira.

### 5.1.2.2 Definição dos Modelos e Objetos de Acesso a Dados no Dispositivo

Os modelos e objetos de acesso a dados utilizados no dispositivo são definidos de forma semelhante à realizada no servidor e apresentada na Seção 5.1.1.2. Os modelos e DAOs referentes às tabelas Clientes, Pedidos e ItensPedido, que suportam sincronização incremental, devem ser especializações das classes *BaseClientSyncModel* e *BaseClientSyncDAO*, apresentadas no Apêndice A.2.2. O DAO da classe Produtos, que realiza apenas sincronizações do tipo *refresh*, deve ser uma especialização da classe *BaseClientSimpleSyncDAO*, também apresentada no Apêndice A.2.2.

### 5.1.2.3 Configuração do SyncAgent e Início do Processo de Sincronização

Esta Seção detalha a utilização do SyncAgent no dispositivo. O primeiro passo é a instanciamento do agente de sincronização, conforme apresentado na figura 5.14. Este exemplo, por razões de simplicidade, define o código do usuário que irá realizar a sincronização com o valor fixo 1. A figura 5.14 também apresenta a atribuição do *SyncMetaDataProvider*, que é utilizado pelo agente de sincronização para obter informações referentes às âncoras de sincronização armazenadas no dispositivo (ver Apêndice A.2.6).

A figura 5.15 apresenta a instanciamento do *GenericWebServiceAdapter*. Conforme detalhado no Apêndice A.2.4, ele é utilizado pelo agente de sincronização para interagir com o banco de dados central. O construtor do *GenericWebServiceAdapter* recebe como parâme-

```
//Instância o agente de sincronização
SyncAgent sa = new SyncAgent();
//Atribui o código do usuário
sa.IdUser = 1;
//Atribui o SyncMetaDataProvider utilizado pelo SyncAgent
sa.SyncMetaDataProvider =
    new BaseSyncClientMetaDataProvider<AppTables>(DBUtils.DbProvider);
```

Figura 5.14: Caso 1: Instanciação do *SyncAgent*

tro um objeto do tipo *WebServiceC1*, apresentado na figura 5.16, que é uma especialização da classe *mWebService* (ver Apêndice A.2.4). As funções *Insert* e *GetUpdated* da classe *mWebService* foram sobrescritas porque, como discutido na Seção 5.1.1.3, o *XmlSerializer*, do *framework* .NET, precisa conhecer previamente as classes que serão serializadas / deserializadas, o que é realizado por meio do atributo *XmlInclude* destacado na figura 5.16.

```
//Instância o WebService
WebServiceC1 WebService =
    new WebServiceC1("http://192.168.0.3:1075/SyncWebService/Service.asmx");

//Instância WebServiceAdapter genérico
GenericWebServiceAdapter webServiceAdapter =
    new GenericWebServiceAdapter(WebService);
```

Figura 5.15: Caso 1: Instanciação do *GenericWebServiceAdapter*

O próximo passo na utilização do *framework* mSync é a instanciação de um elemento de sincronização associado a cada tabela, conforme apresentado na figura 5.17. Os elementos de sincronização referentes às tabelas clientes, pedidos e itens de pedido foram configurados com o tipo de sincronização incremental em ambos os sentidos: cliente para servidor e servidor para cliente. O elemento de sincronização referente à tabela produtos, como pode ser observado nas linhas 80 e 81 da figura 5.17, foi configurado para não realizar a sincronização no sentido cliente para servidor e, no sentido servidor para cliente, realizar uma sincronização do tipo *refresh*, onde a cópia local do dispositivo é substituída por uma nova cópia obtida do servidor durante o processo de sincronização. O construtor do elemento de sincronização possui como parâmetros uma interface para acesso ao banco de dados local e uma interface para acesso ao servidor de sincronização. Cada elemento de sincronização utiliza como interface para o banco de dados local um *data access object* (DAO) referente à tabela pertinente e, como interface para acesso ao servidor, todos eles utilizam o mesmo *GenericWebServiceAdapter*, definido na figura 5.15.

A tabela *ItensPedido* pode possuir muitos registros e, por isto, na linha 75 da figura 5.17, é definido que os registros criados ou modificados no banco de dados local do dispositivo serão sincronizados com o servidor em blocos de tamanho máximo de 50 registros. Neste caso, caso a sincronização seja abortada por alguma falha de comunicação, os registros já enviados para

```

public class WebServiceC1 : mWebService
{
    /// <summary>
    /// Cria WebService
    /// </summary>
    /// <param name="Url"></param>
    public WebServiceC1(string Url)
        : base(Url)
    {
    }

    [XmlInclude(typeof(Clientes))]
    [XmlInclude(typeof(Produtos))]
    [XmlInclude(typeof(Pedidos))]
    [XmlInclude(typeof(ItensPedido))]
    public override InsertionResult[] Insert(SyncSession syncSession, object[] ls,
        bool createElementsKey, String methodName)
    {
        return base.Insert(syncSession, ls, createElementsKey, methodName);
    }

    [XmlInclude(typeof(Clientes))]
    [XmlInclude(typeof(Produtos))]
    [XmlInclude(typeof(Pedidos))]
    [XmlInclude(typeof(ItensPedido))]
    public override object[] GetUpdated(SyncSession syncSession,
        string className, string methodName)
    {
        return base.GetUpdated(syncSession, className, methodName);
    }
}

```

Figura 5.16: Caso 1: Classe *WebServiceC1*

o servidor não serão retransmitidos na próxima tentativa.

O *SyncAgent*, conforme detalhado no Apêndice A.2.8, disponibiliza alguns eventos que permitem que a aplicação realize operações específicas durante a sincronização, como excluir registros já atualizado no servidor ou atualizar chaves estrangeiras relacionadas as tabelas sincronizadas, além de monitorar o andamento do processo de sincronização e a ocorrência de erros / conflitos. A figura 5.18 apresenta a forma como estes eventos são configurados e, na linha 93 da mesma figura, o processo de sincronização é iniciado por meio da função *BeginSynchronize*, que dispara o processo de forma assíncrona.

## 5.2 Estudo de caso 2: Sistema de fiscalização de obras, empreendimentos e serviços do Crea-MG

O *framework* *mSync*, antes mesmo da conclusão desta dissertação, está sendo utilizado na solução de fiscalização de obras, empreendimentos e serviços do Conselho Regional de Engenharia, Arquitetura e Agronomia do Estado de Minas Gerais (Crea-MG), o que comprova o funcionamento e utilidade do trabalho proposto.

O Crea é responsável por fiscalizar o exercício profissional das profissões regulamentadas

```

59 //Cria o Elemento de Sincronização referente a tabela de clientes
60 SyncElement se = new SyncElement(new ClientesDAO(), webServiceAdapter);
61 se.SyncClientType = SyncType.Incremental;
62 se.SyncServerType = SyncType.Incremental;
63 sa.SyncElements.Add(se);
64
65 //Cria o Elemento de Sincronização referente a tabela de pedidos
66 se = new SyncElement(new PedidosDAO(), webServiceAdapter);
67 se.SyncClientType = SyncType.Incremental;
68 se.SyncServerType = SyncType.Incremental;
69 sa.SyncElements.Add(se);
70
71 //Cria o Elemento de Sincronização referente a tabela de itens de pedidos
72 se = new SyncElement(new ItensPedidoDAO(), webServiceAdapter);
73 se.SyncClientType = SyncType.Incremental;
74 se.SyncServerType = SyncType.Incremental;
75 se.SyncClient.SetClientMsgPacketSize(50);
76 sa.SyncElements.Add(se);
77
78 //Cria o Elemento de Sincronização referente a tabela de produtos
79 se = new SyncElement(new ProdutosDAO(), webServiceAdapter);
80 se.SyncClientType = SyncType.None;
81 se.SyncServerType = SyncType.Refresh;
82 sa.SyncElements.Add(se);

```

Figura 5.17: Caso 1: Instanciação dos elementos de sincronização

```

84 //Configura os eventos disponibilizados pelo SyncAgent
85 sa.StateChanged += new SyncAgentStateChangedDelegate(sa_StateChanged);
86 sa.AfterDeleteServer += new DeleteResultDelegate(sa_AfterDeleteServer);
87 sa.AfterInsertServer += new InsertionResultDelegate(sa_AfterInsertServer);
88 sa.AfterUpdateServer += new UpdateResultDelegate(sa_AfterUpdateServer);
89 sa.BeforeChangeIdInsertServer += new InsertionResultDelegate
90     (sa_BeforeChangeIdInsertServer);
91
92 //Inicia a sincronização de forma assíncrona
93 sa.BeginSynchronize();

```

Figura 5.18: Caso 1: Configuração dos eventos do *SyncAgent*

pelo sistema CONFEA/Crea. Esta fiscalização, antes da adoção do sistema informatizado, era realizada de forma manual, com a utilização de planilhas, material impresso e banco de dados não centralizado. Em virtude do grande número de empreendimentos, dos diferentes níveis de complexidade dos vistoriados e da necessidade de se consultar informações referentes às empresas, profissionais e anotações de responsabilidade técnica (ARTs) durante a fiscalização, a produtividade e qualidade do trabalho realizado ficava comprometida.

A partir da implantação do sistema de fiscalização, os fiscais passaram a contar com dispositivos ubíquos de acesso a dados, do tipo telefones inteligentes (*SmartPhones*), integrados com GPS e câmara digital. A figura 5.19 exibe alguns dos modelos de equipamentos utilizados pelos fiscais do Crea-MG.

Um dos requisitos do projeto era que, mesmo que algumas informações, como situação dos



Figura 5.19: Dispositivos utilizados no sistema de fiscalização do Crea-MG

profissionais e empresas, fossem acessadas de forma *on-line*, por meio de conexão GPRS, o sistema deveria permitir sua utilização em modo desconectado, pois, em muitas situações, as conexões GPRS oferecem qualidade de serviço insuficiente ou mesmo se tornam indisponíveis. O *framework* mSync foi escolhido como ferramenta para atender este importante requisito, permitindo que parte das informações disponíveis no servidor central fossem copiadas para o banco de dados local do dispositivo, modificadas e, posteriormente, sincronizadas de volta para o servidor.

A solução de fiscalização é dividida em três módulos: GeFisc Web, GeFisc Móvel e Sincronização de dados, conforme detalhado a seguir.

### 5.2.1 GeFisc Web

O módulo GeFisc Web é acessado por meio de navegadores Web comuns no mercado, como o *Internet Explorer* e *Mozilla FireFox*. Este módulo permite o planejamento, acompanhamento e controle das atividades de fiscalização, incluindo o acompanhamento da evolução dos processos, prazos de recurso e notificações. Outra funcionalidade deste módulo é a possibilidade de visualização em um mapa do trajeto percorrido pelos fiscais e a localização das obras, serviços e empreendimentos fiscalizados. As figuras 5.20 e 5.21 apresentam algumas telas do módulo GeFisc Web.

### 5.2.2 GeFisc Móvel

O módulo GeFisc móvel é utilizado pelos fiscais para consultar dados e registrar as informações referentes ao trabalho de fiscalização em campo. Ele é compatível com telefones inteligentes equipados com o sistema operacional *Windows Mobile 5* e suas principais funcionalidades incluem:

- Cadastro das obras, empreendimentos e serviços fiscalizados;
- Cadastro de pessoas fiscalizadas;
- Cadastro de atividades;

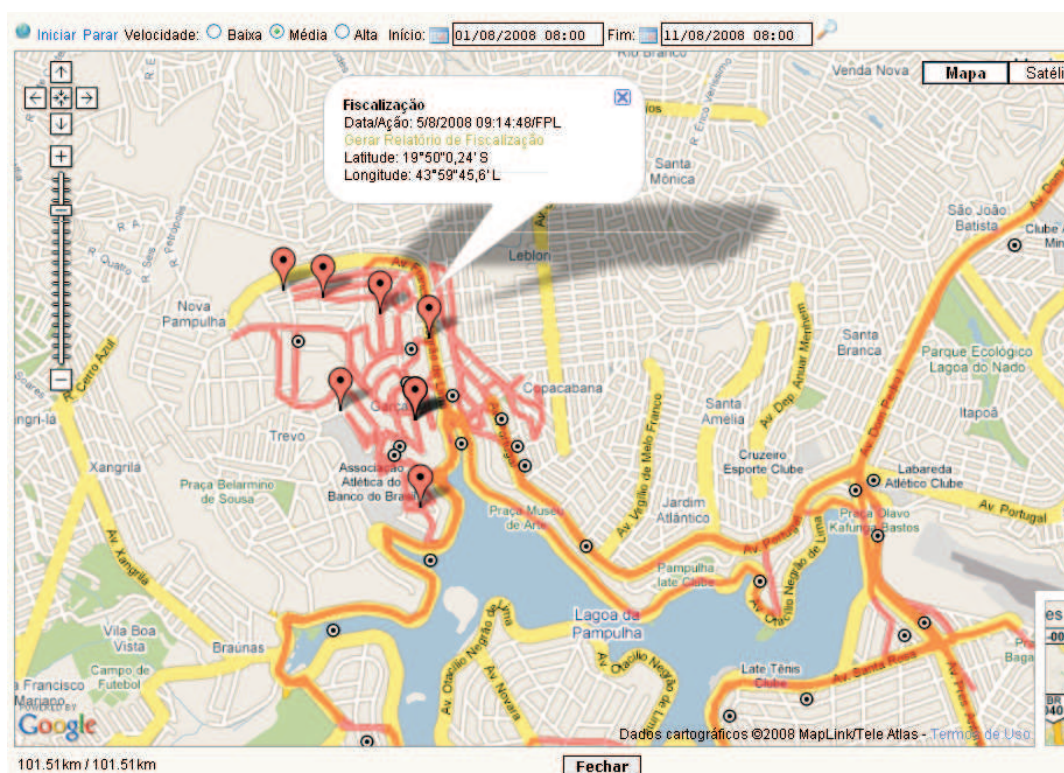


Figura 5.20: Tela de visualização da rota percorrida pelo fiscal no GeFisc Web

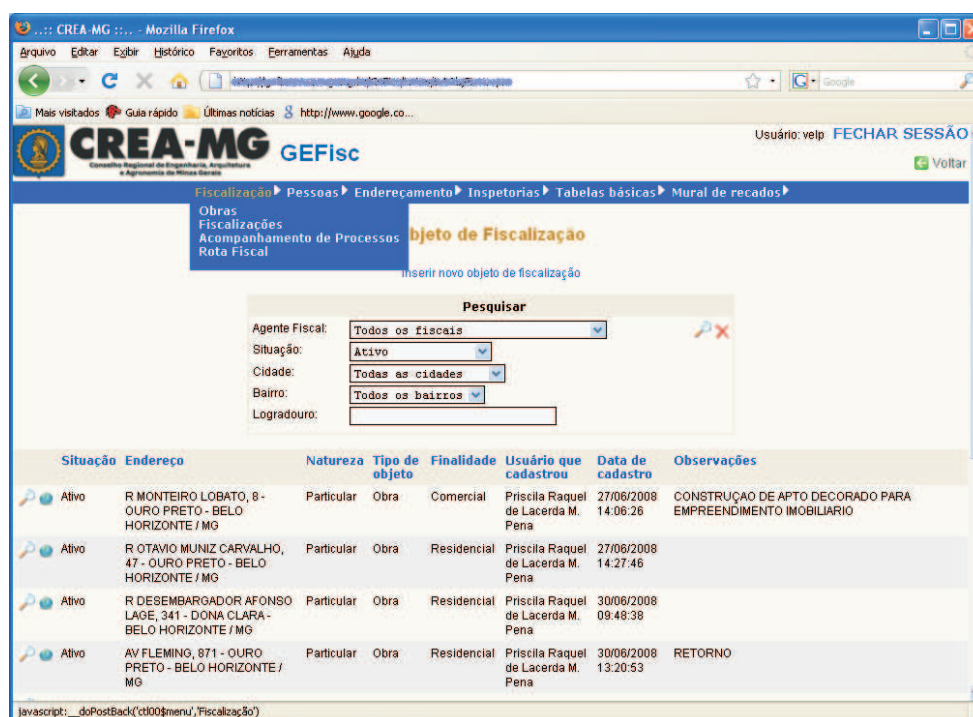


Figura 5.21: Tela de objetos de fiscalização do GeFisc Web

- Registro do relatório de fiscalização;

- Emissão de notificações;
- Georeferenciamento das fiscalizações por meio do GPS integrado ao dispositivo utilizado pelo fiscal;
- Rastreamento do trajeto percorrido pelo fiscal durante o trabalho de fiscalização;
- Registro de imagens digitais, obtidas com a câmera integrada ao dispositivo, associadas às fiscalizações realizadas;
- Acesso *on-line* ao banco de dados corporativo do Crea-MG, permitindo que os fiscais consultem informações referentes às empresas, profissionais e anotações de responsabilidade técnica;
- Acompanhamento da agenda de fiscalização.

A figura 5.22 apresenta algumas telas do módulo GeFisc Móvel.

### 5.2.3 Sincronização de Dados

O módulo de sincronização de dados foi implementado utilizando como base o *framework* mSync, sendo responsável pela integração entre os módulos GeFisc Web e GeFisc Móvel. O sistema permite que a sincronização de dados entre os dispositivos e o servidor seja realizada diretamente do local fiscalizado, por meio de uma conexão GPRS, ou do escritório, conectando o dispositivo a Internet por meio de um computador de mesa ou rede sem fio (802.11). Durante a sincronização, os dados coletados pelos fiscais, incluindo as fotografias dos locais fiscalizados e registro das coordenadas do trajeto percorrido, são enviados para o servidor central e, as informações modificadas no servidor, como, por exemplo, obras cadastradas por outros fiscais, modificações nos processos existentes e mensagens enviadas pelos gerentes, são aplicadas no dispositivo.

O sistema de fiscalização possui mais de quarenta tabelas sincronizadas, onde algumas pouco modificadas, como, por exemplo, as tabelas de municípios, bairros e logradouros, suportam apenas a sincronização do tipo *refresh*, enquanto outras, como as tabelas de fiscalização, proprietários e notificações, realizam a sincronização do tipo incremental.

A preparação do servidor de sincronização e utilização do *framework* mSync no dispositivo foram realizadas de forma análoga a apresentada no estudo de caso 1, detalhado na Seção 5.1. Contudo, necessidades específicas exigiram que, em algumas situações, o comportamento do *framework* fosse modificado. Nas Seções 5.2.3.1, 5.2.3.2 e 5.2.3.3 são apresentados alguns exemplos onde os recursos disponibilizados pelo *framework* são adaptados para solucionar problemas específicos da aplicação, o que demonstra a flexibilidade com que o mSync permite a construção de soluções de sincronização personalizadas.

#### 5.2.3.1 Utilização do Evento *AfterInsertServer*

O evento *AfterInsertServer* (ver Apêndice A.2.8) é disparado pelo agente de sincronização (*SyncAgent*) depois que os registros criados no dispositivo são aplicados no servidor central.

| Consulta Online                   |                        | Objeto de Fiscalização |                                                                |
|-----------------------------------|------------------------|------------------------|----------------------------------------------------------------|
| Tipo Pessoa:                      | Pessoa Física          | Endereço:              | AV AFONSO PENA, 765 -<br>FUNCIONARIOS - BELO HORIZONTE /<br>MG |
| Reg:                              | MG 95715 SD-Sup        |                        |                                                                |
| Nome:                             | PAULO HENRIQUE LOSSIO  |                        |                                                                |
| CPF:                              | 01352212692            |                        |                                                                |
| Tp. Reg.:                         | RD-REGISTRO DEFINITIVO |                        |                                                                |
| Sit. Reg.:                        | AAA-ATIVO              |                        |                                                                |
| Déb. anuidade:                    | NAO                    |                        |                                                                |
| Deve exercício:                   | NAO                    |                        |                                                                |
| Deve exe. ant.:                   | NAO                    |                        |                                                                |
| Sit. anuidade:                    | Q-QUITADA              |                        |                                                                |
| Geral Observações Endereço Respor |                        | Online Voltar          |                                                                |

| Cadastro de Fiscalização                        |            | Cadastro de Infração                                                                                        |                                                                       |
|-------------------------------------------------|------------|-------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| Situação Obra:                                  | Paralisada | Cod:                                                                                                        | 12                                                                    |
| Fase:                                           | ACABAMENTO | Descrição:                                                                                                  | Falta de registro da ART Obra/Serviço.                                |
| Tipo Exec.:                                     | Direta     | Observação:                                                                                                 | Entende-se por registro da ART, a<br>quitaação, o cadastramento e a l |
| Tipo Contrato:                                  | Empreitada | Data da Ocorrência.:                                                                                        | 13/ 8 /08                                                             |
| Tipo Obra:                                      | Ampliação  | <input checked="" type="checkbox"/> Ocorrência entregue com assinatura<br>e identificação do real infrator. |                                                                       |
| Área Construída:                                | 1000 m²    | Buscar Capitulação                                                                                          |                                                                       |
| Nr Pavimentos.:                                 | 2          | Capitulação Atividades Notificado Ob                                                                        |                                                                       |
| <b>Dados do Contrato</b><br>R\$ Prazo (dias): 0 |            | Salvar Cancelar                                                                                             |                                                                       |
| Fonte Info.: PLACA                              |            | Salvar Cancelar                                                                                             |                                                                       |
| Geral Obra Entrevistado Obs Outr                |            | Salvar Cancelar                                                                                             |                                                                       |

Figura 5.22: Telas do GeFisc Móvel

A solução de fiscalização do Crea-MG utiliza este evento para atualizar o valor dos identificadores únicos das tabelas “fiscalizacao” e “pessoa” com seus valores definitivos, recebidos do servidor durante a sincronização, nas tabelas que utilizam estes identificadores como chaves estrangeiras. Esta operação é necessária porque, por questões relacionadas à modelagem de dados que estão fora do escopo deste trabalho, alguns relacionamentos não puderam ser definidos como sendo do tipo *update cascade*, onde a atualização da chave estrangeira seria realizada pelo próprio SGBD, de forma transparente para a aplicação.

Outro uso comum para o evento *AfterInsertServer* é exclusão de registros do banco de

dados local do dispositivo depois que os mesmos são enviados para o servidor. A aplicação apresentada no estudo de caso utiliza este procedimento para excluir os registros da tabela “mensagem”. A tabela mensagem é utilizada para a troca de mensagens entre os fiscais, semelhante a um correio interno e, depois que os registros da caixa de saída são enviados para o servidor, os mesmos podem ser excluídos do banco de dados do dispositivo.

A figura 5.23 exibe a função *sa\_AfterInsertionResult*, que, nesta aplicação, é acionada pelo evento *AfterInsertServer*. Na figura pode ser observado o tratamento especial dispensado as tabelas “fiscalização”, “pessoa” e “mensagem”. Nas linhas 478 e 481 da figura, o método “UpdateFiscKey”, dos DAOs pertinentes, é acionado para executar a atualização do identificador único dos registros nas tabelas que o utilizam como chave estrangeira. Na linha 484 os registros da tabela “mensagem” que foram criados no dispositivo e aplicados no servidor são excluídos do banco de dados local.

```
464 void sa_EndInsertionResult(object sender, InsertionResultArg insertionResultArg)
465 {
466     string tableName = insertionResultArg.ModelType.Name.ToLower();
467
468     try
469     {
470         foreach (InsertionResult item in insertionResultArg.Result)
471             if (item.ErrorCod != 0)
472                 Utils.MsgError(item.ErrorDescription, "Erro");
473             else
474             {
475                 switch (tableName)
476                 {
477                     case "fiscalizacao":
478                         Singleton.infracaoDAO.UpdateFiscKey(item.IdDM, item.IdSF);
479                         break;
480                     case "pessoa":
481                         Singleton.pessoaDAO.UpdateFiscKey(item.IdDM, item.IdSF);
482                         break;
483                     case "mensagem":
484                         Singleton.mensagemDAO.DelMsgSync();
485                         break;
486                 }
487             }
488     }
489     catch (Exception ex)
490     {
491         Utils.MsgError("Erro: " + ex.Message, "Erro");
492     }
493 }
```

Figura 5.23: Caso 2: Exemplo de utilização do evento *AfterInsertServer*

Caso o valor da propriedade *item.ErrorCod*, verificado na linha 472 da figura 5.23, seja diferente de zero, significa que, conforme detalhado no Apêndice A.1, ocorreu algum erro ou conflito durante a inserção do registro correspondente no banco de dados central. Neste exemplo, a ocorrência do conflito é tratada pela função “*Utils.MsgError*”, que armazena a descrição do problema em um arquivo de log.

### 5.2.3.2 Configuração da quantidade máxima de registros enviados para o servidor em uma única operação

O mSync permite a configuração da quantidade máxima de registros novos ou modificados enviados do dispositivo para o servidor durante uma única operação. Esta funcionalidade é útil nos casos onde a quantidade de registros novos ou modificados no dispositivo é grande. Nestas situações, caso ocorresse alguma falha de conexão durante o processo de sincronização, todos os registros da tabela sincronizada teriam que ser retransmitidos na próxima tentativa. Para contornar este problema, o mSync permite a configuração da quantidade máxima de registros que podem ser transmitidos para o servidor em uma única operação. Com isto, caso a conexão seja interrompida, os registros presentes nos pacotes já transmitidos não são retransmitidos na próxima tentativa.

A aplicação apresentada no estudo de caso utiliza esta propriedade na sincronização de duas tabelas: “PontosRota”, que armazena as coordenadas do trajeto percorrido pelo fiscal, e “Fotos”, que armazena as fotografias obtidas pelo fiscal com a câmera integrada ao dispositivo.

A tabela “PontosRota”, dependendo da distância percorrida pelo fiscal, pode possuir uma quantidade grande de registros, da ordem de milhares de linhas. Na linha 148 da figura 5.24, o *SyncElement* referente a esta tabela é configurado para transmitir os registros em blocos de tamanho máximo de 100 registros.

As fotografias das fiscalizações obtidas pelos fiscais por meio da câmera integrada aos dispositivos possuem tamanho médio de 60KB. Na linha 159 da figura 5.24, a aplicação configura o *SyncElement* de forma que as imagens digitais sejam transmitidas uma por vez para o servidor de sincronização.

```
144 // PontosRota
145 PontosRotaDAO pontosDAO = new PontosRotaDAO();
146 //Defini a quantidade máxima de registros transmitidos
147 //para o servidor em um única operação
148 pontosDAO.SetClientMsgPacketSize(100);
149 se = new SyncElement(pontosDAO, webService);
150 se.ServerCreateElementKey = true;
151 se.SyncClientType = SyncType.Incremental;
152 se.SyncServerType = SyncType.Incremental;
153 sa.SyncElements.Add(se);
154
155 // Fotos
156 FotosDAO fotosDAO = new FotosDAO();
157 //Define que as fotografias serão transmitidas
158 //uma por vez para o servidor
159 fotosDAO.SetClientMsgPacketSize(1);
160 se = new SyncElement(fotosDAO, webService);
161 se.ServerCreateElementKey = true;
162 se.SyncClientType = SyncType.Incremental;
163 se.SyncServerType = SyncType.Incremental;
164 sa.SyncElements.Add(se);
```

Figura 5.24: Caso 2: Exemplo de utilização da propriedade *SetClientMsgPacketSize*

### 5.2.3.3 Sobrescrevendo funções da *BaseSyncDAO* no servidor

As áreas de atuação dos fiscais do Crea-MG são divididas em regiões chamadas inspetorias. Para diminuir o volume de dados armazenados na memória local dos dispositivos e obter os outros benefícios relacionados ao particionamento de dados para trabalho em modo desconectado, apresentados na Seção 3.5, podem ser armazenadas na memória local dos dispositivos apenas as informações referentes as obras, empreendimentos e serviços pertencentes a inspetoria do fiscal.

Para atender este requisito as funções *GetAllUpdated* e *GetUpdated* da classe *BaseSyncDAO* (ver Apêndice A.3.3) foram sobrescritas. Estas funções são responsáveis por gerar a lista de registros, obtidos do banco de dados central, que serão armazenados nos dispositivos. A figura 5.25 apresenta como estas funções foram sobrescritas no DAO referente a tabela “ObjFisc”, que armazena os objetos fiscalizados.

```
public class ObjFiscDAO : BaseSyncDAO<ObjFisc>
{
    public ObjFiscDAO()
        : base(DBUtils.DbProvider)
    {
    }

    public override List<object> GetAllUpdated(int idSyncStart, int idSyncEnd, int idUser)
    {
        // Código da inspetoria do usuário
        int idInsp = new BaseSimpleDAO<Usuario>().GetElementByPrimarykey(idUser).IdInsp;

        string sql = "select * from tbObjFisc where ID_SYNCCHANGE>=" + idSyncStart +
            " and ID_SYNCCHANGE<" + idSyncEnd + " and ST_OBJ='A' and ID_INSP=" + idInsp;

        return base.LoadDataObject(sql);
    }

    public override List<object> GetUpdated(int idSyncStart, int idSyncEnd, int idUser)
    {
        // Código da inspetoria do usuário
        int idInsp = new BaseSimpleDAO<Usuario>().GetElementByPrimarykey(idUser).IdInsp;

        string sql = "select * from tbObjFisc where ID_SYNCCHANGE>=" + idSyncStart +
            " and ID_SYNCCHANGE<" + idSyncEnd + " and ID_USERCHANGE<>" + idUser +
            " and ST_OBJ='A' and ID_INSP=" + idInsp;

        return base.LoadDataObject(sql);
    }
}
```

Figura 5.25: Caso 2: Classe ObjFiscDAO

Procedimento análogo ao exibido na figura 5.25, para a tabela “ObjFisc”, foi realizado nos outros DAOs referentes as tabelas cujos registros também podem ser divididos por meio da inspetoria do fiscal.

## 5.3 Resultados

Os resultados de um *framework* de sincronização de dados devem ser avaliados levando-se em consideração vários fatores, como facilidade de uso, funcionamento correto, robustez em caso

de falhas e desempenho. Uma avaliação completa da proposta desenvolvida incluiria questões relacionadas a flexibilidade, reutilização, manutenção, entre outras, que não são claramente perceptíveis (Eden e Mens (2006); Poulin (1994); Paulish e Carleton (1994)) e estão fora do escopo deste trabalho.

Os fatores facilidade de uso e robustez em caso de falhas também são difíceis de serem avaliados, porém, a solução de fiscalização de obras, empreendimentos e serviços, apresentada no estudo de caso 2, está sendo utilizada desde abril de 2008 por fiscais do Crea-MG que atuam na região metropolitana de Belo Horizonte. O cronograma de implantação do projeto prevê que todos os 170 fiscais do Crea-MG estarão utilizando a solução até o final de 2009. Os resultados obtidos até o momento indicam o funcionamento correto do arcabouço, não tendo sido identificadas ocorrências de perda de dados ou duplicação de registros.

Além do funcionamento correto do *framework*, que é comprovado por meio dos mais de cinco meses de uso intenso da solução, outra característica importante que pode ser medida é o tempo gasto nas operações de sincronização de dados, considerando-se diferentes tipos de redes de dados e operações. Esta Seção apresenta os resultados obtidos nos testes de desempenho do *framework*. Foram realizadas medições dos tempos de sincronização referentes as operações de inserção, atualização e exclusão de registros nos sentidos dispositivo para servidor e servidor para dispositivo. As sincronizações foram testadas por meio de rede celular GPRS EDGE e rede sem fio local 802.11g, conforme cenário apresentado na figura 5.26.

As tabelas 5.1, 5.2 e 5.3 detalham, respectivamente, a configuração do dispositivo ubíquo, do servidor de sincronização e do servidor de banco de dados utilizados nos testes. Os tempos de sincronização foram obtidos sincronizando-se registros da tabela “TBPESSOA”, apresentada na figura 5.28.

Tabela 5.1: Dispositivo Ubíquo (figura 5.27)

|                      |                                        |
|----------------------|----------------------------------------|
| Modelo               | PDA N3 Navigator da Luxicom            |
| Sistema Operacional  | Windows Mobile 6                       |
| Processador          | Marvell PXA270 416 MHz                 |
| Memória              | 128MB                                  |
| Rede                 | GSM/GPRS/EDGE                          |
| Conectividade        | Wi-Fi (IEEE 802.11 b/g), Bluetooth 2.0 |
| Funções Extras       | Câmera 2.0MP, GPS Integrado            |
| Banco de Dados Móvel | Microsoft SQL Mobile 2005              |

Tabela 5.2: Servidor de Sincronização

|                     |                                                |
|---------------------|------------------------------------------------|
| Processador         | AMD Athlon XP 2400+ 2.00 GHz                   |
| Memória             | 512 MB de RAM                                  |
| Sistema Operacional | Microsoft Windows Server 2003 Standard Edition |
| Servidor Web        | IIS (Internet Information Services) 6.0        |

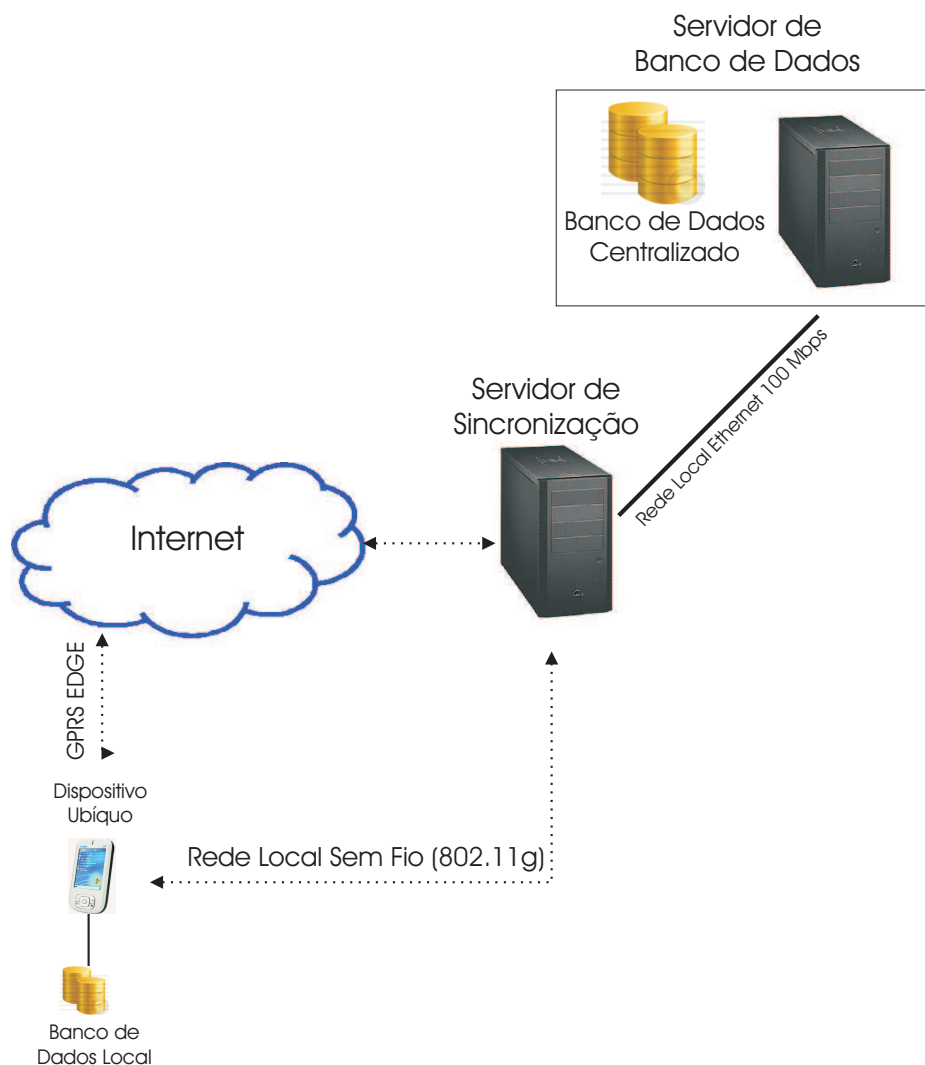


Figura 5.26: Cenário utilizado nos testes de performance

Tabela 5.3: Servidor de Banco de Dados

|                     |                                                      |
|---------------------|------------------------------------------------------|
| Processador         | Intel Core2 Duo CPU E4500 2.20GHz                    |
| Memória             | 2.0 GB de RAM                                        |
| Sistema Operacional | Microsoft Windows Server 2003 Enterprise Edition SP1 |
| Banco de Dados      | Microsoft SQL Server 2000                            |

### Sincronização sentido Dispositivo para Servidor

Os tempos médios das sincronizações no sentido dispositivo para servidor são apresentados na tabela 5.4. Todas as operações testadas apresentam comportamento assintótico linear no número de registros sincronizados. As sincronizações realizadas por meio da rede local sem fio (802.11g), conforme esperado, foram mais rápidas do que as realizadas por meio da rede celular GPRS EDGE. De modo geral, os tempos de sincronização referentes as operações de inserção, atualização e exclusão de registros, para um mesmo tipo de rede, são próximos entre si. Os tempos das sincronizações realizadas por meio da rede celular EDGE possuem



Figura 5.27: Dispositivo ubíquo utilizado nos testes de performance

Tabela 5.4: Tempos de Sincronização (s): Dispositivo para Servidor

| -              | Rede Sem Fio 802.11g |           |         | Rede Celular GPRS EDGE |           |         |
|----------------|----------------------|-----------|---------|------------------------|-----------|---------|
| Qtd. Registros | Inserir              | Atualizar | Excluir | Inserir                | Atualizar | Excluir |
| 10             | 6                    | 4         | 5       | 12                     | 9         | 9       |
| 20             | 10                   | 6         | 7       | 17                     | 12        | 14      |
| 30             | 12                   | 11        | 11      | 19                     | 19        | 17      |
| 40             | 17                   | 14        | 16      | 27                     | 22        | 26      |
| 50             | 19                   | 17        | 19      | 28                     | 26        | 27      |
| 60             | 23                   | 19        | 22      | 35                     | 29        | 32      |
| 70             | 26                   | 23        | 26      | 43                     | 36        | 43      |
| 80             | 30                   | 26        | 28      | 43                     | 38        | 47      |
| 90             | 37                   | 29        | 33      | 49                     | 46        | 51      |
| 100            | 39                   | 32        | 36      | 57                     | 53        | 54      |

um comportamento menos linear do que os realizados utilizando a rede sem fio local. Este comportamento pode ser creditado ao fato da rede celular e da Internet serem menos estáveis do que a rede local sem fio, diminuindo a precisão com que os tempos de sincronização podem ser estimados. As figuras 5.29 e 5.30 ilustram os comentários apresentados. As linhas contínuas na figura 5.29 indicam a tendência dos tempos das operações e foram obtidas por meio de regressão linear.

### Sincronização sentido Servidor para Dispositivo

A tabela 5.5 e figuras 5.31 e 5.32 apresentam os tempos médios de sincronização no sentido servidor para dispositivo. Diferente da sincronização no outro sentido, onde os tempos obtidos para as três operações são muito próximos, no sentido servidor para dispositivo é possível identificar que, quando utilizado rede local sem fio (802.11g), as operações de atualização são mais rápidas do que as de exclusão e inserção de registros.

| TBPESSOA         |              |      |
|------------------|--------------|------|
| <u>ID_PESSOA</u> | int          | <pk> |
| TP_PESSOA        | char(1)      |      |
| FL_REG_CREA      | bit          |      |
| RG_CREA          | int          |      |
| TP_CREA          | int          |      |
| NR_CREA          | int          |      |
| NM_PESSOA        | varchar(100) |      |
| NR_CPF_PESSOA    | varchar(11)  |      |
| NM_REP_LEGAL     | varchar(100) |      |
| NR_CPF_REP_LEGAL | varchar(11)  |      |
| NM_FANTASIA      | varchar(100) |      |
| NR_CNPJ          | varchar(14)  |      |
| SG_UF            | char(2)      |      |
| CD_CIDADE        | int          |      |
| DS_BAIRRO        | varchar(50)  |      |
| TP_LOGRAD        | varchar(10)  |      |
| DS_LOGRAD        | varchar(100) |      |
| NR_LOGRAD        | varchar(10)  |      |
| DS_COMPL         | varchar(50)  |      |
| NR_CEP           | varchar(15)  |      |
| NR_FONE          | varchar(15)  |      |
| NR_FAX           | varchar(15)  |      |
| DS_EMAIL         | varchar(50)  |      |
| ID_SYNCCHANGE    | int          |      |
| ID_USERCHANGE    | int          |      |
| CD_USU_INS       | int          |      |
| DT_INS           | datetime     |      |
| CD_USU_ALT       | int          |      |
| DT_ALT           | datetime     |      |

Figura 5.28: Tabela TBPESSOA

Conforme detalhado nas Seções 4.3.2 e 4.7.2, o *framework* não faz distinção entre registros inseridos e modificados, de forma que, durante a sincronização, ambos são enviados para o dispositivo da mesma forma, onde são aplicados no banco de dados local por meio da operação *InsertOrUpdateServerSync*. Esta operação garante que, caso o registro exista na base de dados local, ele será atualizado e, caso não exista, será inserido. A implementação do método *InsertOrUpdateServerSync* primeiro tenta atualizar o registro e, se o número de linhas afetadas for igual a zero, executa a operação de inserção. Com isto, durante as sincronizações do tipo incremental, toda operação de inserção de registro no banco de dados é precedida de uma tentativa de atualização, o que explica porque as operações de atualização de registros são um pouco mais rápidas do que as de inserção.

Quanto às operações de exclusão, não foram localizados no *framework* fatores que justifiquem o fato delas serem mais lentas do que as de inserção, o que sugere que a remoção de registros no banco de dados *SQL Server Mobile* possa levar mais tempo do que a inserção.

Outra observação interessante é que o tempo de exclusão de registros, quando considerado sincronizações por meio da rede GPRS EDGE, é menor do que o tempo das outras operações. Uma justificativa possível é que, nas operações de exclusão, são transmitidos apenas os

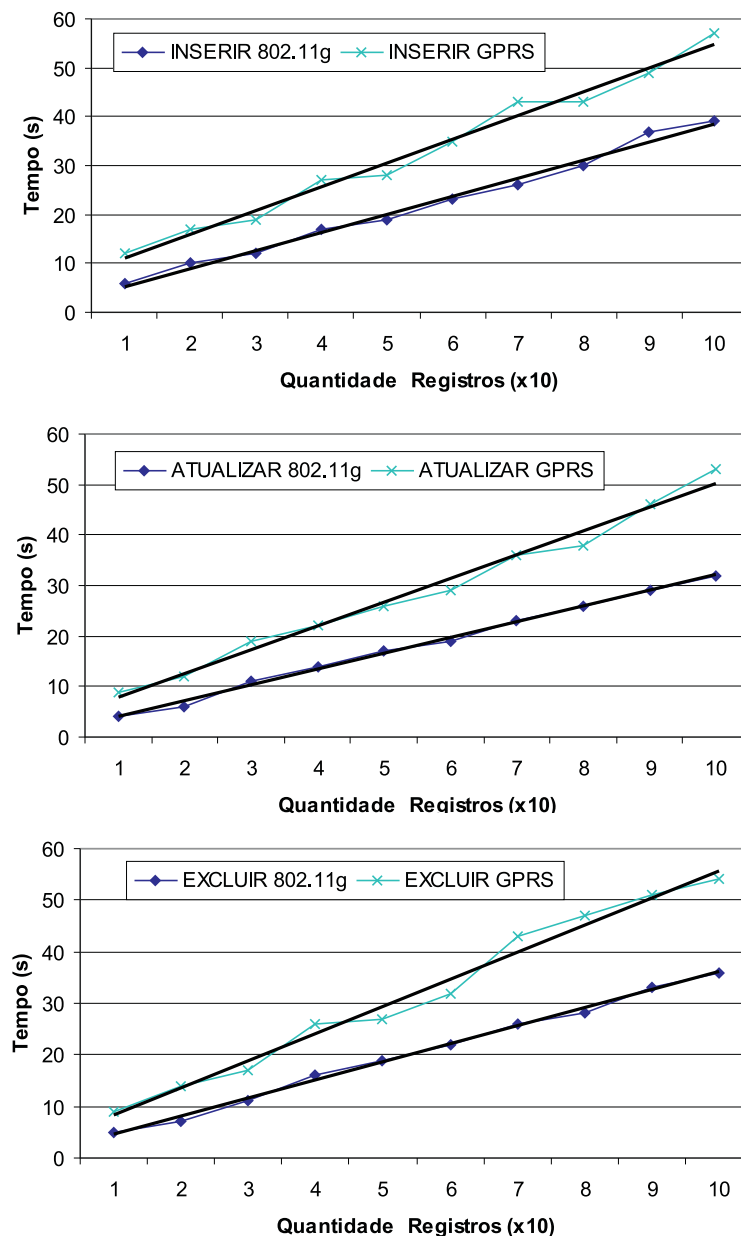


Figura 5.29: Tempos de sincronização sentido dispositivo para servidor

identificadores dos registros que devem ser excluídos, enquanto nas operações de inserção e atualização é necessário a transmissão de todos os campos. No momento em que é utilizada uma rede de dados de menor velocidade, como a GPRS EDGE, o fato das operações de exclusão exigirem a transmissão de um volume menor de dados se torna significativo, fazendo com que esta operação se torne, proporcionalmente, mais rápida do que as de inserção e atualização de registros.

Conforme esperado, para uma mesma operação, o tempo de sincronização utilizando rede sem fio local foi menor que utilizando a rede celular GPRS EDGE. Novamente o comporta-

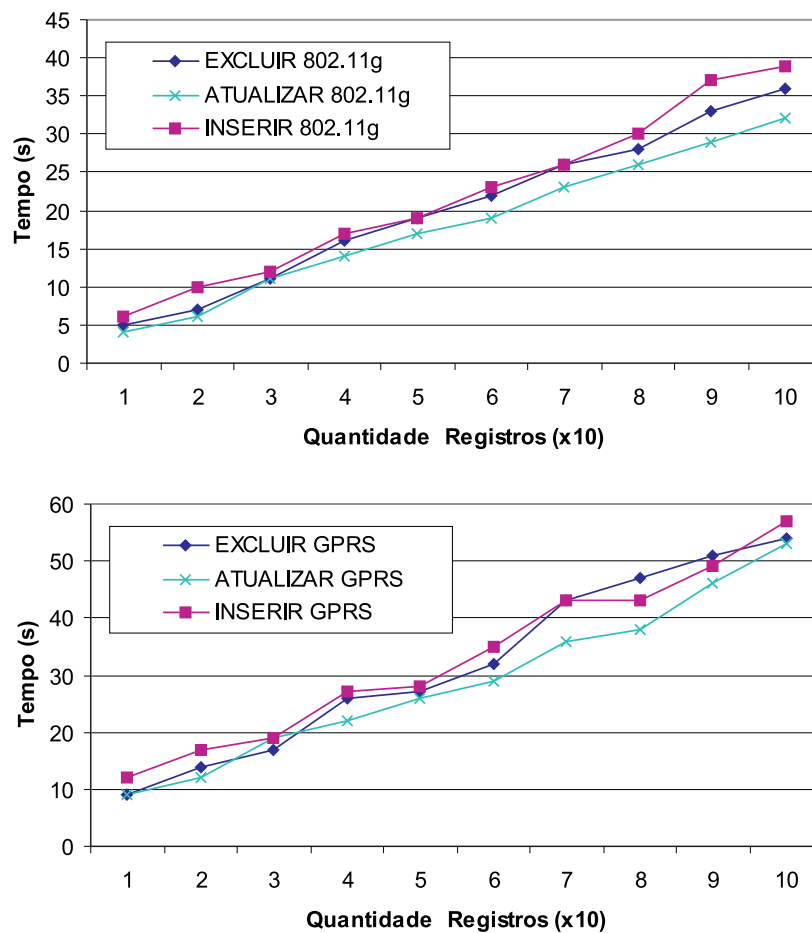


Figura 5.30: Comparação entre os tempos das operações no sentido dispositivo para servidor

mento dos tempos de sincronização utilizando a rede celular foi menos linear do que utilizando a rede local sem fio.

### Comparação entre as Sincronizações sentido Dispositivo para Servidor e Servidor para Dispositivo

A figura 5.33 apresenta três gráficos que comparam o desempenho das operações de inserção, alteração e exclusão de registros nos sentidos dispositivo para servidor e servidor para dispositivo. Pode-se observar que os tempos de sincronização para as operações de inserção e exclusão de registros, nos dois sentidos, são parecidos. Quanto as operações de atualização de dados, elas são mais rápidas quando executadas no sentido servidor para dispositivo do que dispositivo para servidor.

A diferença observada entre os tempos de sincronização nos dois sentidos pode ser motivada por vários fatores, como, por exemplo, assimetria na capacidade computacional dos dispositivos e dos servidores, possíveis diferenças nas velocidades de *upload* e *download* de dados, diferença na performance de execução de operações nos bancos de dados, entre ou-

Tabela 5.5: Tempos de Sincronização (s): Servidor para Dispositivo

| -              | Rede Sem Fio 802.11g |           |         | Rede Celular GPRS EDGE |           |         |
|----------------|----------------------|-----------|---------|------------------------|-----------|---------|
| Qtd. Registros | Inserir              | Atualizar | Excluir | Inserir                | Atualizar | Excluir |
| 10             | 7                    | 6         | 6       | 17                     | 15        | 14      |
| 20             | 12                   | 8         | 9       | 22                     | 20        | 16      |
| 30             | 15                   | 11        | 13      | 23                     | 21        | 21      |
| 40             | 18                   | 13        | 16      | 28                     | 25        | 24      |
| 50             | 22                   | 14        | 20      | 29                     | 31        | 25      |
| 60             | 24                   | 16        | 23      | 38                     | 35        | 27      |
| 70             | 28                   | 18        | 26      | 36                     | 37        | 35      |
| 80             | 31                   | 19        | 29      | 44                     | 48        | 35      |
| 90             | 35                   | 23        | 33      | 49                     | 47        | 40      |
| 100            | 38                   | 25        | 36      | 50                     | 51        | 47      |

tras. O mapeamento, identificação e detalhamento destes fatores está fora do escopo deste trabalho.

## 5.4 Conclusão

Este Capítulo apresentou dois estudos de caso. O primeiro, uma aplicação simples de cadastro de clientes e pedidos, foi apresentada com o objetivo didático de demonstrar a utilização do *framework*. Foram detalhadas as configurações necessárias no banco de dados, no servidor de sincronização e nos dispositivos.

O segundo estudo de caso apresentou uma aplicação real de fiscalização de obras, empreendimentos e serviços que utiliza o *framework* mSync como ferramenta para permitir a consulta e modificação de dados em modo desconectado. A aplicação é utilizada pelos fiscais do Crea-MG na região metropolitana de Belo Horizonte e, segundo o cronograma de implantação, será utilizada por todos os 170 fiscais até o final de 2009. A apresentação do estudo de caso 2 se concentrou na exemplificação de como o *framework* pode ser modificado para atender necessidades específicas da aplicação como, por exemplo, armazenar localmente em um dispositivo móvel apenas as informações sobre as obras, empreendimentos e serviços pertencentes a mesma inspetoria que o fiscal.

O Capítulo foi concluído com a apresentação dos resultados de desempenho da sincronização considerando-se as operações de inserção, atualização e exclusão de registros, utilizando-se uma rede de dados local sem fio 802.11g e rede celular GPRS EDGE.

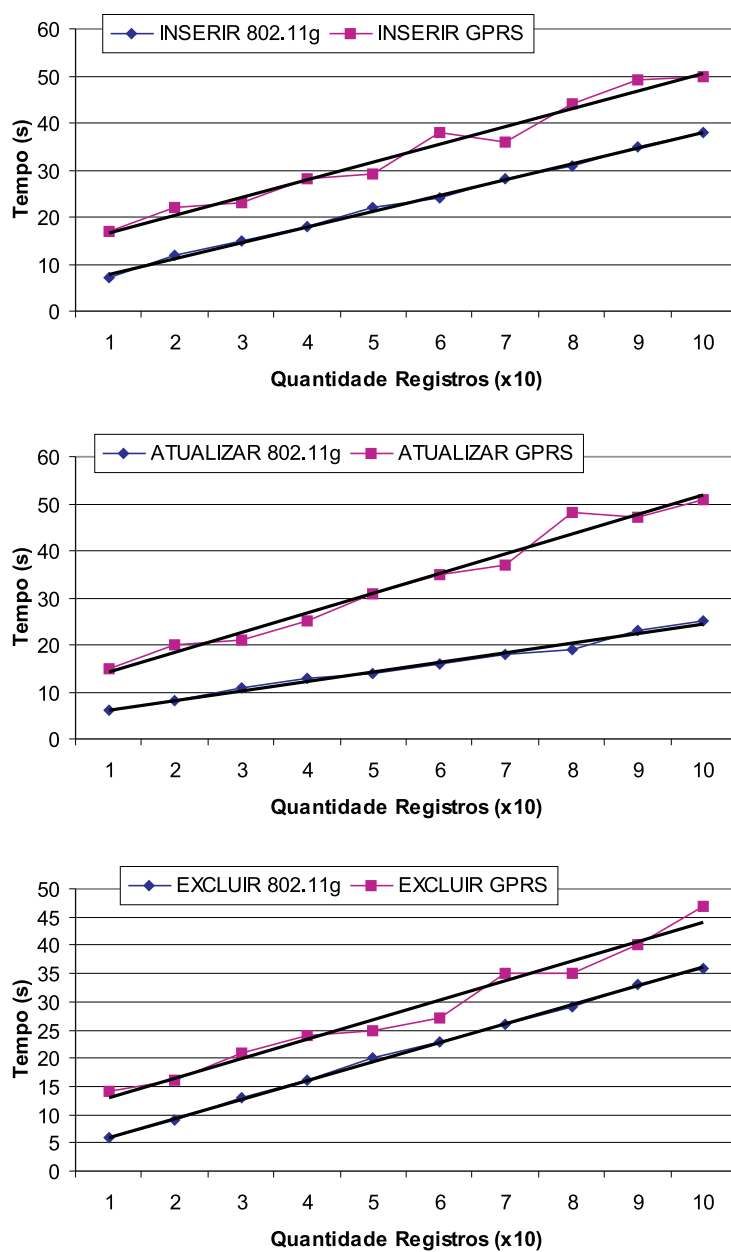


Figura 5.31: Tempos de sincronização sentido servidor para dispositivo

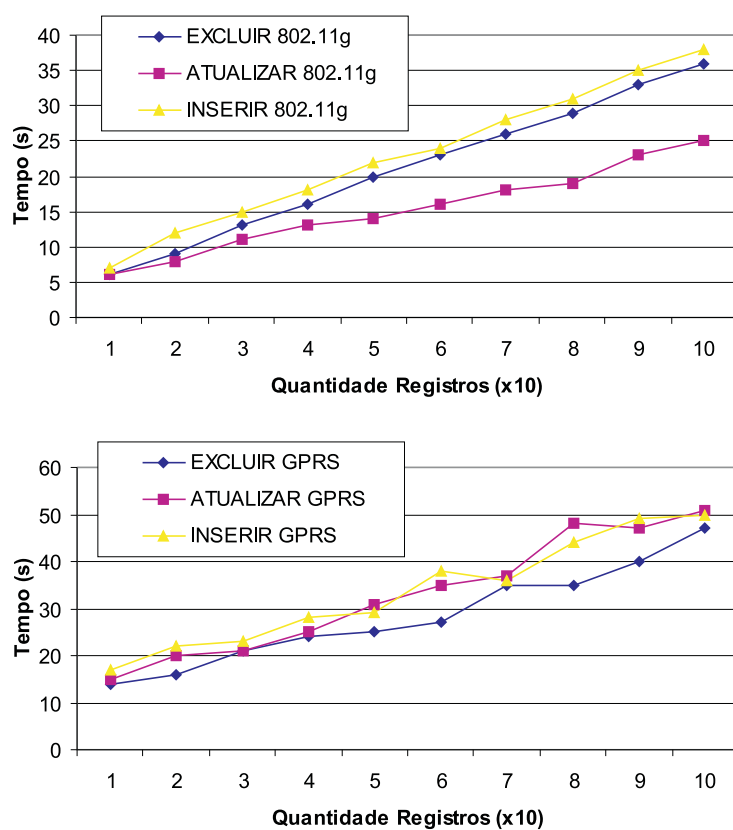


Figura 5.32: Comparação entre os tempos das operações no sentido servidor para dispositivo

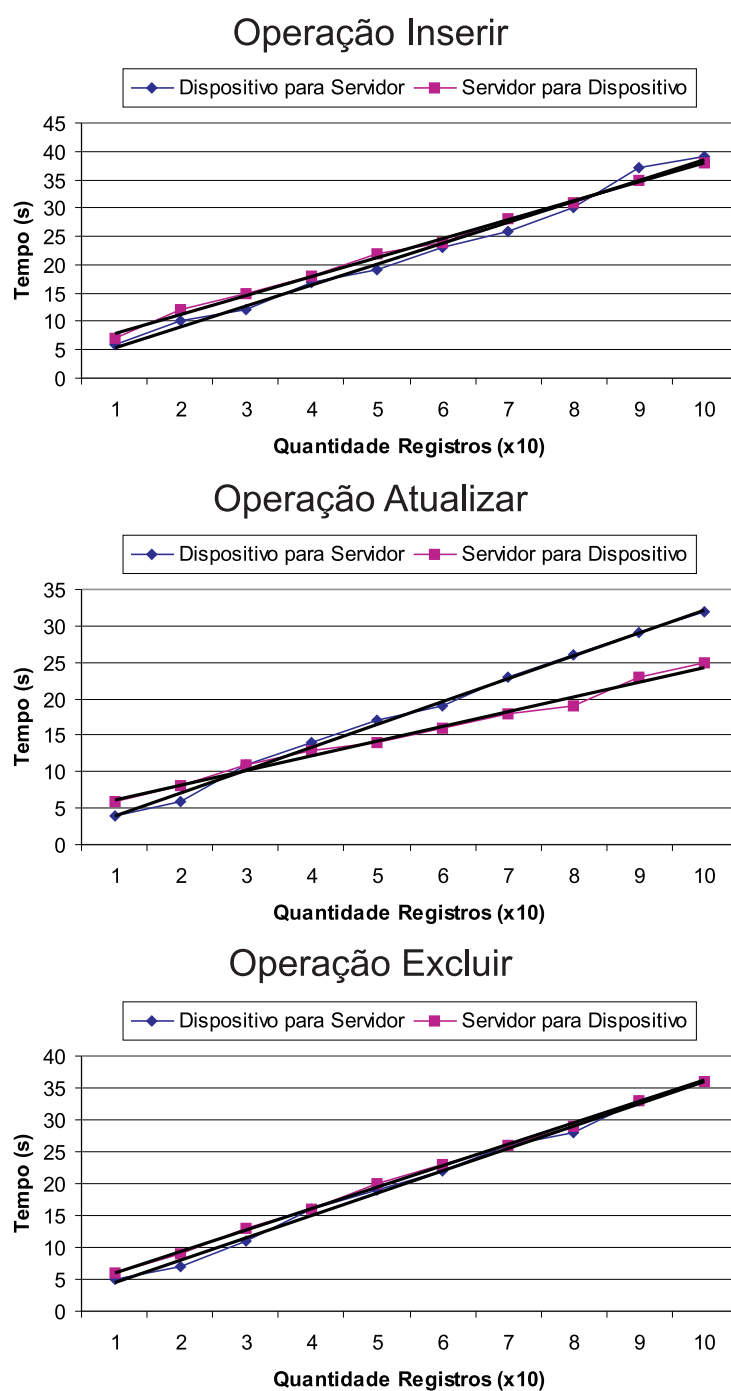


Figura 5.33: Comparação entre os tempos de sincronização nos sentidos dispositivo para servidor e servidor para dispositivo

## Capítulo 6

# Conclusão

A visão de mundo da computação ubíqua implica que as pessoas tenham acesso a informações a todo o momento e em todo o lugar. Mesmo com os avanços na tecnologia de dispositivos e de comunicação, ainda não é possível estar conectado o tempo inteiro por razões como falta de cobertura de redes sem fio ou limitações energéticas. Uma forma de emular um ambiente de conectividade onipresente é copiar parte das informações disponíveis em servidores da rede fixa para os dispositivos ubíquos, onde poderão ser consultadas e modificadas durante os períodos sem conexão. Depois, com a conexão restabelecida, as modificações realizadas no banco de dados local dos dispositivos precisam ser aplicadas no banco de dados central, assim como as modificações efetuadas no banco de dados central precisam ser aplicadas no banco de dados local dos dispositivos, em um processo conhecido como sincronização.

A sincronização de dados introduz complexidade aos sistemas computacionais, exigindo que os programadores se ocupem com questões como rastreamento de modificações, detecção / resolução de conflitos e geração de identificadores únicos em um ambiente distribuído. A contribuição principal deste trabalho é a proposta e implementação de um *framework*, chamado mSync, para sincronização de dados em dispositivos ubíquos. O *framework* trata vários problemas relacionados à consulta e modificações de dados em um ambiente distribuído, permitindo que desenvolvedores utilizem a sincronização de dados em suas aplicações como se fosse um serviço, podendo concentrar esforços em questões específicas da aplicação. O uso do *framework* agiliza o desenvolvimento de soluções que utilizem replicação e sincronização de dados, provendo um nível elevado de abstração que esconde dos programadores a complexidade introduzida pela distribuição.

O trabalho apresentou uma visão geral sobre computação ubíqua, introduzindo os principais conceitos e terminologia. Foi visto que o *framework* proposto vai ao encontro dos princípios da simplicidade, diversidade, descentralização e conectividade da computação ubíqua (Hansmann et al. (2001)). Foi apresentada uma revisão sobre as principais categorias de dispositivos ubíquos e sistemas operacionais utilizados, com ênfase especial aos dispositivos de acesso à informação, que são os mais relevantes para o trabalho proposto.

O texto introduziu conceitos relacionados à replicação e sincronização de dados, como consistência em bancos de dados móveis, âncoras de sincronização, carga inicial de dados e

rastreamento de modificações. Foi visto que as estratégias relacionadas à consistência em bancos de dados móveis podem ser divididas em duas categorias: pessimistas e otimistas (Davidson et al. (1984)). A pessimista garante a consistência do banco de dados limitando a disponibilidade de informações. A otimista, em contrapartida, aumenta a disponibilidade dos dados flexibilizando algumas propriedades de consistência. O *framework* proposto vai ao encontro dos principais trabalhos encontrados na literatura e adota a estratégia otimista, que é considerada a mais apropriada para um ambiente de computação ubíqua.

Quando a estratégia otimista é adotada, é comum a existência de conflitos, que ocorrem quando dois ou mais dispositivos modificam a mesma informação. O trabalho detalhou o conceito de conflito e mostrou como o *framework* oferece a infra-estrutura necessária para sua detecção e resolução.

O mSync utiliza *Web services* para intercâmbio de dados entre os dispositivos e o servidor. *Web services*, por ser um padrão aberto, flexível e independente da arquitetura, tem se apresentado como uma tendência para troca de dados entre aplicações (Kangasharju et al. (2007)). Porém, como visto no texto, o XML / SOAP não são apropriados para o ambiente de redes sem fio e dispositivos ubíquos. Uma sugestão de trabalho futuro, que já está sendo desenvolvida, é adaptar o mSync para suportar outros protocolos mais eficientes para troca de dados entre os dispositivos e o servidor, como, por exemplo, o WBXML (*Wap Binary XML*).

Um dos desafios em soluções móveis é a geração de identificadores para os registros de forma que eles sejam únicos, evitando o problema conhecido como colisão de chaves primárias e, ao mesmo tempo, compactos. O trabalho propôs um esquema para geração de identificadores diferentes dos encontrados em projetos comerciais e na literatura científica e, conforme visto no texto, possui vantagens em relação às alternativas estudadas.

O *framework* se aplica a cenários onde vários dispositivos sincronizam informações com um único servidor central, em uma topologia do tipo estrela. Em algumas aplicações, pode ser necessário que diferentes dispositivos possam sincronizar dados entre si, se comportando hora como cliente e hora como servidor. Para suportar esta abordagem, algumas técnicas propostas no *framework*, como a geração de identificadores únicos e rastreamento de modificações nos dispositivos, precisam revistas e adaptadas.

Foram apresentados dois estudos de caso para ilustrar a eficiência e utilidade do *framework*. O primeiro, uma aplicação simples de cadastro de clientes e pedidos, foi desenvolvido com o objetivo didático de demonstrar a utilização do *framework*. Foram detalhadas as configurações necessárias no banco de dados, no servidor de sincronização e nos dispositivos. O segundo estudo de caso apresentou uma aplicação real de fiscalização de obras, empreendimentos e serviços que utiliza o mSync como ferramenta para permitir a consulta e modificação de dados em modo desconectado. A aplicação é utilizada pelos fiscais do Crea-MG na região metropolitana de Belo Horizonte desde de maio de 2008 e, segundo o cronograma de implantação, será utilizada por todos os 170 fiscais até o final de 2009. Os resultados obtidos indicam o funcionamento correto do *framework*, não tendo sido identificadas ocorrências de perda de dados ou problemas de sincronização.

Uma das dificuldades encontradas foi a definição da metodologia a ser utilizada para avaliação do *framework*, pois fatores importantes como facilidade de uso, flexibilidade, reutilização, manutenção, entre outros, não são claramente perceptíveis. Pelos mesmos motivos, a comparação com outros *frameworks* propostos na literatura e em projetos comerciais também não é uma tarefa simples. Optou-se pela realização de testes comparativos entre o desempenho da sincronização utilizado-se rede sem fio local 802.11g e rede celular GPRS/EDGE. Conforme esperado, a sincronização por meio da rede sem fio local foi mais rápida do que a realizada por meio da rede celular. Porém, a diferença entre os tempos de sincronização quando utilizada a rede sem fio local 802.11g e a rede GPRS/EDGE não foi diretamente proporcional a diferença de velocidade entre os dois tipos de rede. Esta observação se deve ao fato de uma parcela significativa do tempo total de sincronização estar relacionado ao processamento dos dados no dispositivo, fazendo que o impacto de se utilizar uma rede de dados de maior velocidade tenha seu efeito suavizado. Esta observação é importante por indicar que, em algumas situações, o uso de redes de dados mais rápidas, como as redes celulares 3G, podem não significar tempos de sincronização significativamente menores.

Um grande benefício de um *framework* é que ele pode ser evoluído por meio da implementação de melhorias e novas funcionalidades. A medida em que as evoluções são implementadas, as aplicações já existentes também podem se beneficiar das novas implementações por meio da atualização do *framework*. Este trabalho deu o passo inicial, estruturando o *framework*, resolvendo diversos problemas e permitindo que o mesmo seja utilizado em uma solução completa para sincronização de dados. Mas vários pontos podem ser melhorados. A lista abaixo apresenta algumas sugestões de trabalhos futuros, sendo que algumas destas já estão sendo desenvolvidas.

- Pesquisa e utilização de protocolos mais eficientes e apropriados para o ambiente de computação ubíqua, do que *Web services / SOAP*, para troca de informações entre os dispositivos e o servidor. Uma alternativa promissora é a utilização do WBXML (*Wap Binary XML*).
- Melhorias no *framework* utilizado para persistência de dados, de forma que o tempo de processamento de dados durante a sincronização seja reduzido.
- Estudo e implementação de técnicas onde a sincronização seja realizada por meio da transmissão do arquivo de banco de dados do dispositivo para o servidor, e vice-versa, ao invés do envio dos registros individualmente. Esta abordagem, na maior parte dos casos, exige que seja transmitido um volume maior de dados, mas, em contrapartida, o processamento é deslocado do dispositivo, computacionalmente limitado, para o servidor, mais potente, o que, em muitas situações, pode representar tempos de sincronização menores.
- Estudo, implementação e avaliação de técnicas de segurança na sincronização de dados apropriadas para o ambiente de redes e dispositivos ubíquos.

- 
- Criação de um portal na Internet para disponibilizar a documentação e informações sobre o *framework* desenvolvido.

## Apêndice A

# Implementação do *Framework* mSync

A ferramenta escolhida para a implementação do *framework* foi o *Microsoft Visual Studio .NET* (VisualStudio (2008)), porém os conceitos e idéias propostos na arquitetura podem ser portados para outros ambientes.

A escolha do .NET se deve ao fato de, conforme constatado nas últimas edições do *Mobile Word Congress* (<http://www.mobileworldcongress.com>), que é uma das principais feiras mundiais dos fabricantes de dispositivos móveis, uma parte significativa dos novos equipamentos lançados no mercado possuem o sistema operacional *Windows Mobile* (ver Seção 2.3.3), da *Microsoft*, e o .NET é uma das ferramentas mais utilizadas no desenvolvimento de aplicações para este sistema operacional.

A implementação do *framework* é dividida em três arquivos de biblioteca de ligação dinâmica, do inglês *Dynamic link library* ou DLL, diferentes. Os arquivos que compõem o *framework* são:

- *mSyncCommon.dll*: Contém as classes que são utilizadas no servidor e nos dispositivos.
- *mSyncClient.dll*: Contém as classes da arquitetura utilizadas no dispositivo.
- *mSyncServer.dll*: Contém as classes da arquitetura utilizadas no servidor.

O Apêndice está organizado da seguinte forma. A Seção A.1 apresenta as principais classes definidas no arquivo *mSyncCommon*. A Seção A.2 detalha o *mSyncClient* e a Seção A.3 trata do *mSyncServer*.

### A.1 mSyncCommon

O *mSyncCommon* contém as classes comuns aos módulos executados nos dispositivos e no servidor. Para permitir a portabilidade para as duas plataformas, servidor com o sistema operacional da família *Windows 2000, XP, 2003 Server* e dispositivos com sistema operacional *Windows Mobile*, o módulo foi implementado utilizando-se o *.NET Compact Framework*

(.NET CF) (ver seção 2.3.2), da *Microsoft*, que é uma versão simplificada do *.NET Framework*.

A figura A.1 apresenta a estrutura da classe *SyncSession*, definida no módulo *mSyncCommon*. A *SyncSession* é utilizada como parâmetro de entrada nos métodos de integração entre os dispositivos e o servidor, conforme será apresentado na Seção A.2.3. As propriedades *IdSyncStart* e *IdSyncEnd* se referem às âncoras de sincronização iniciais e finais utilizadas para recuperar as informações do servidor. O *IdSyncInsert* é a âncora utilizada para verificar se um novo registro transmitido pelo dispositivo para ser inserido no servidor já foi ou não aplicado anteriormente durante uma sincronização que não foi finalizada com sucesso, conforme detalhado na Seção 4.4. A propriedade *IdUser* se refere à identificação do dispositivo que está realizando a sincronização.

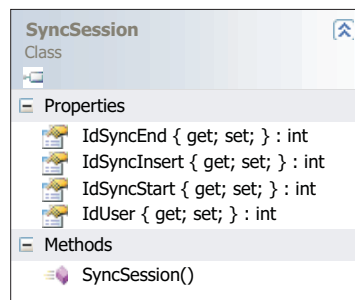


Figura A.1: Classe *SyncSession*

A figura A.2 exibe o diagrama de classes dos objetos retornados pelas funções de exclusão, atualização e inserção de registros no servidor durante a sincronização de dados.

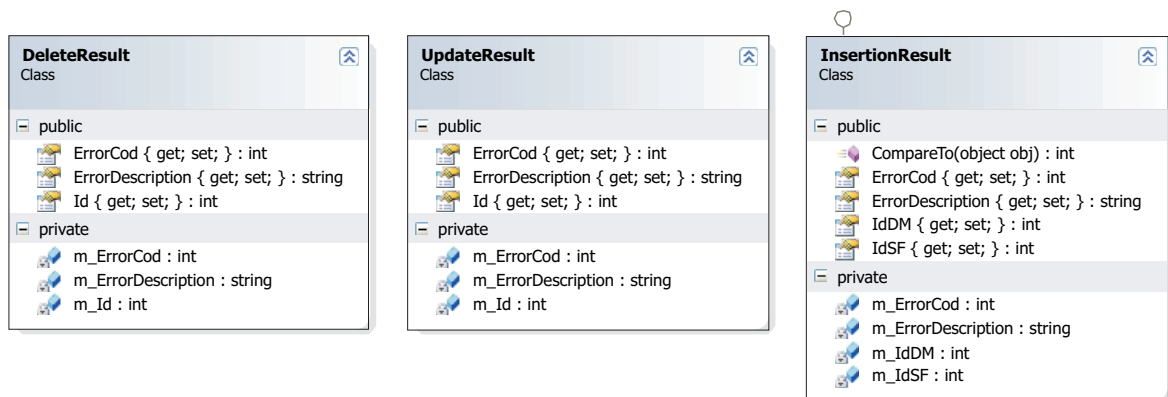


Figura A.2: Objetos de retorno das funções de exclusão, atualização e inserção de registros no servidor

Nas classes *DeleteResult* e *UpdateResult*, a propriedade *Id* representa o identificador único do registro ao qual o resultado da operação está associado, enquanto as propriedades *ErrorCode* e *ErrorDescription* se referem ao código e a descrição do possível erro ocorrido durante a aplicação da operação no servidor. Caso a operação seja executada com sucesso, os cam-

pos *ErrorCode* e *ErrorDescription* são retornados pelo servidor com os valores zero e vazio, respectivamente.

A classe *InsertionResult* é utilizada no retorno da função de inserção de dados no servidor durante a sincronização de dados. As propriedades *IdDM* e *IdSF* se referem ao identificador único temporário do registro, criado no dispositivo móvel, e ao identificador único definitivo, criado no servidor fixo, respectivamente. Estas duas informações permitem que o identificador único local (LUID) de cada registro seja substituído pelo identificador único global (GUID) durante o processo de sincronização.

A figura A.3 exibe as enumerações referentes ao tipo de sincronização e ao estado dos registros armazenados no banco de dados do dispositivo. Estas informações também são definidas no módulo *mSyncCommon*.

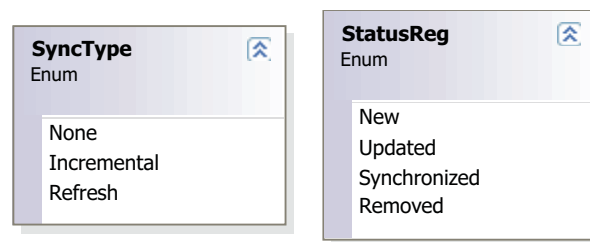


Figura A.3: Enumerações utilizadas no mSync

O módulo *mSyncCommon* também define a classe *BaseDAO*, apresentada na figura A.4. Esta é a classe base dos DAOs (*Data Access Object* (Gamma et al. (1994))) utilizados no *framework*, tanto no servidor quanto nos dispositivos, e disponibiliza diversos métodos importantes, como os utilizados para inserção, atualização e exclusão de registros.

## A.2 mSyncClient

O *mSyncClient* é o módulo do *framework* utilizado nos dispositivos móveis. Ele foi desenvolvido utilizando-se o *.NET Compact Framework* (.NET CF) e, nesta Seção, são apresentadas suas principais classes e funcionalidades.

### A.2.1 Interface ISyncClient

A figura A.5 apresenta a interface *ISyncClient*. Todos os DAOs (*Data Access Object*) utilizados para persistência de dados durante a sincronização de dados devem implementar esta interface, pois, conforme apresentado na figura 4.9, é por meio dela que o *SyncAgent* tem acesso à base de dados local do dispositivo. Os métodos da interface *ISyncClient* são detalhados a seguir:

- *ChangeDMKey(InsertionResult[] insertionResults) : void* - Recebe como parâmetro a lista de resultados da inserção de registros no banco de dados central e realiza a subs-

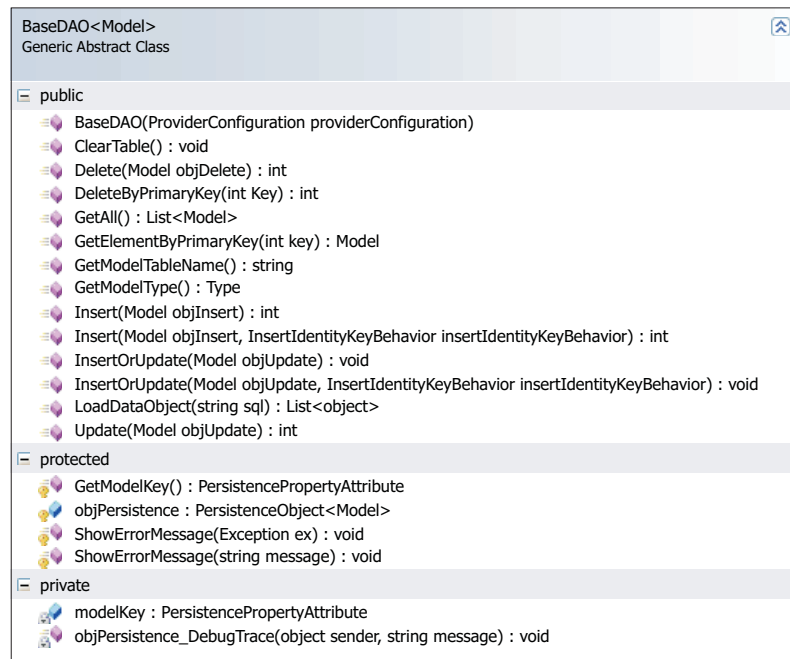


Figura A.4: Classe BaseDAO

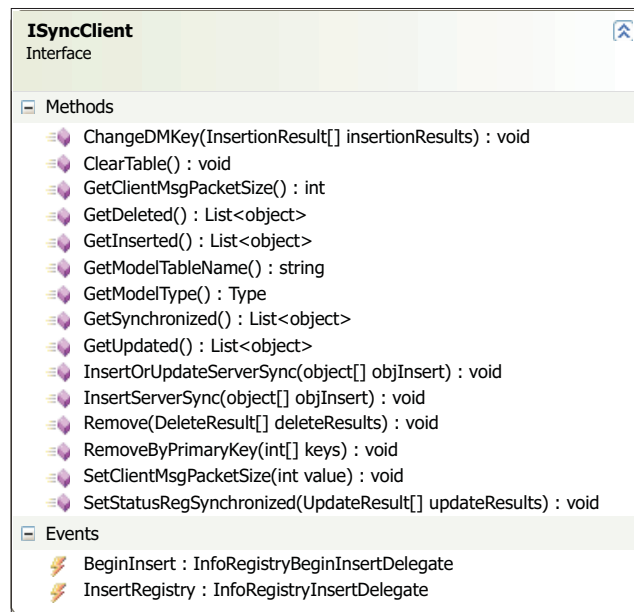


Figura A.5: Interface ISyncClient

tituição do LUID temporário, gerado no dispositivo, pelo GUID definitivo, gerado no servidor, conforme detalhado na Seção 4.4.

- *ClearTable()* : *void* - Exclui fisicamente todos os registros da tabela no banco de dados do dispositivo móvel. Utilizado nas sincronizações do tipo *refresh*.

- *GetClientMsgPacketSize()* : *int* - Retorna a quantidade máxima de registros que podem ser transmitidos e confirmados no servidor em uma mesma operação. Esta funcionalidade possibilita que, caso a conexão seja interrompida, os registros de uma mesma tabela que já foram transmitidos não precisem ser retransmitidos na próxima tentativa.
- *GetDeleted()* : *List<object>* - Retorna uma lista de objetos contendo os registros excluídos no banco de dados local do dispositivo.
- *GetInserted()* : *List<object>* - Retorna uma lista de objetos contendo os registros inseridos no banco de dados local do dispositivo.
- *GetModelTableName()* : *string* - Retorna o nome da tabela associada ao modelo (*model*) utilizado pelo DAO.
- *GetModelType()* : *Type* - Retorna o tipo do modelo utilizado pelo DAO.
- *GetSynchronized()* : *List<object>* - Retorna uma lista de objetos contendo os registros que não foram modificados no banco de dados do dispositivo durante o período de desconexão.
- *GetUpdated()* : *List<object>* - Retorna uma lista de objetos contendo os registros modificados no banco de dados local do dispositivo.
- *InsertOrUpdateServerSync(object[] objInsert)* : *void* - Insere ou atualiza uma lista de registros recebidos do servidor no banco de dados local do dispositivo. Caso o registro não exista na base de dados, ele é inserido, caso já exista, é atualizado.
- *InsertServerSync(object[] objInsert)* : *void* - Insere um registro recebido do servidor no banco de dados local do dispositivo.
- *Remove(DeleteResult[] deleteResults)* : *void* - Remove fisicamente os registros do banco de dados do dispositivo que foram excluídos com sucesso no servidor fixo.
- *RemoveByPrimaryKey(int[] keys)* : *void* - Remove fisicamente os registros do banco de dados do dispositivo utilizando como chave um vetor de inteiros contendo o identificador único dos registros que devem ser excluídos.
- *SetClientMsgPacketSize(int value)* : *void* - Atribui a quantidade máxima de registros que podem ser transmitidos e confirmados no servidor em uma mesma operação.
- *SetStatusRegSynchronized(UpdateResult[] updateResults)* : *void* - Atualiza o estado dos registros modificados, cuja modificação tenha sido aplicada com sucesso no servidor, para o estado sincronizado.

A interface define dois eventos, descritos abaixo, que podem ser utilizados pela aplicação para exibir para o usuário informações relativas ao andamento do processo de sincronização de dados.

- *BeginInsert* : *InfoRegistryBeginInsertDelegate* - Este evento é acionado antes que os registros recebidos do servidor comecem a ser inseridos no banco de dados local do dispositivo. O parâmetro *InfoRegistryBeginInsertDelegate* é utilizado para indicar a quantidade total de registros que serão inseridos.
- *InsertRegistry* : *InfoRegistryInsertDelegate* - Este evento é acionado durante a inserção dos registros recebidos do servidor no banco de dados local do dispositivo. O parâmetro *InfoRegistryInsertDelegate* é utilizado para indicar a quantidade atual de registros já inseridos.

### A.2.2 ClientSyncDAO

O *ClientSyncDAO* implementa a interface *ISyncClient*, apresentada na Seção A.2.1, permitindo que a aplicação no dispositivo e o agente de sincronização tenham acesso à base de dados local do dispositivo.

Deve ser definido um *ClientSyncDAO* para cada tabela sincronizada. O mSync disponibiliza duas opções de classes bases para o *ClientSyncDAO*, que são aplicáveis a uma grande variedade de situações, mas, caso seja necessário tratar situações específicas, o programador pode implementar sua própria *ClientSyncDAO* sobrecarregando métodos já existentes ou criando novas classes, desde que a mesma implemente a interface *ISyncClient* exigida pelo *framework*. A seguir são apresentadas as duas classes bases para o *ClientSyncDAO* disponíveis na arquitetura.

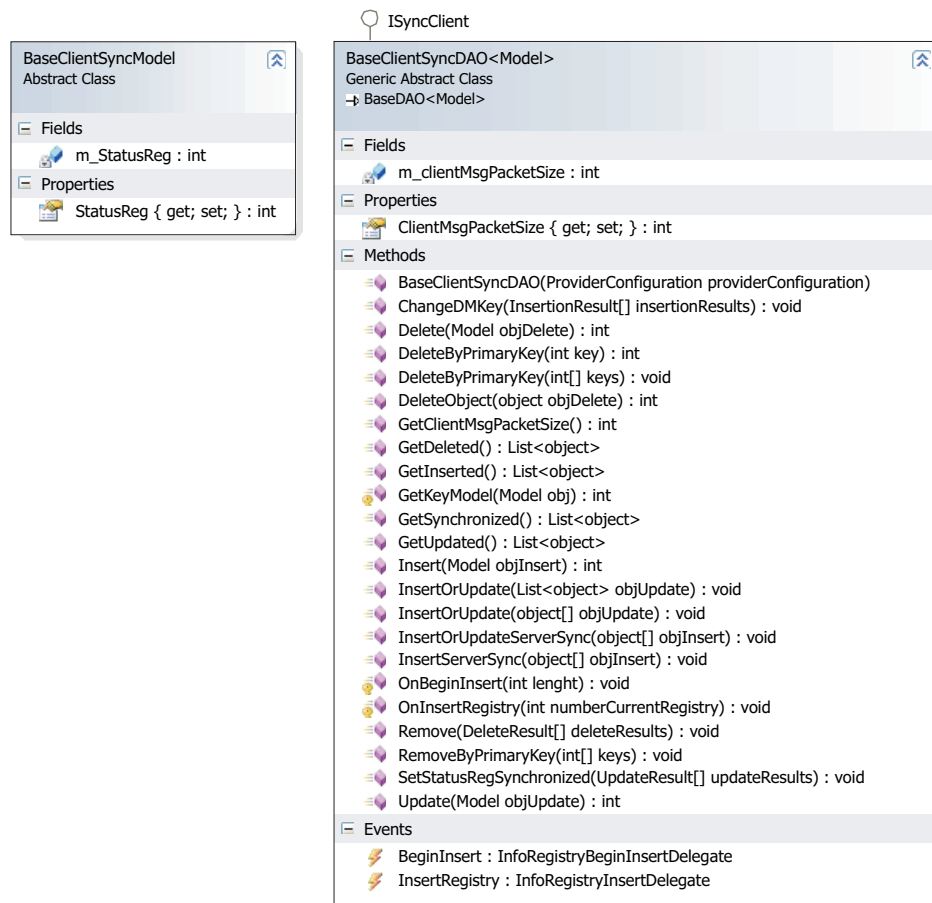
#### BaseClientSyncDAO

A figura A.6 apresenta a classe abstrata genérica *BaseClientSyncDAO*. Ela pode ser utilizada como base para o DAO das tabelas que precisam suportar sincronização do tipo *incremental* no sentido dispositivo móvel -> servidor fixo.

Os métodos disponibilizados no *BaseClientSyncDAO* implementam o rastreamento das modificações realizadas na base de dados local do dispositivo durante o período de desconexão, conforme detalhado na Seção 4.3.1. Para permitir o rastreamento das modificações, as tabelas no dispositivo devem possuir um campo do tipo inteiro chamado *StatusReg*, que é utilizado para armazenar o estado dos registros. A classe *BaseClientSyncDAO* exige que os modelos utilizados sejam especializações da classe *BaseClientSyncModel*, que também é apresentada na figura A.6.

#### BaseClientSimpleSyncDAO

A classe *BaseClientSimpleSyncDAO*, exibida na figura A.7, é utilizada como base para o DAO das tabelas que suportam apenas a sincronização do tipo *refresh* e, portanto, não precisam registrar as alterações realizadas na base de dados local do dispositivo durante o período de desconexão.

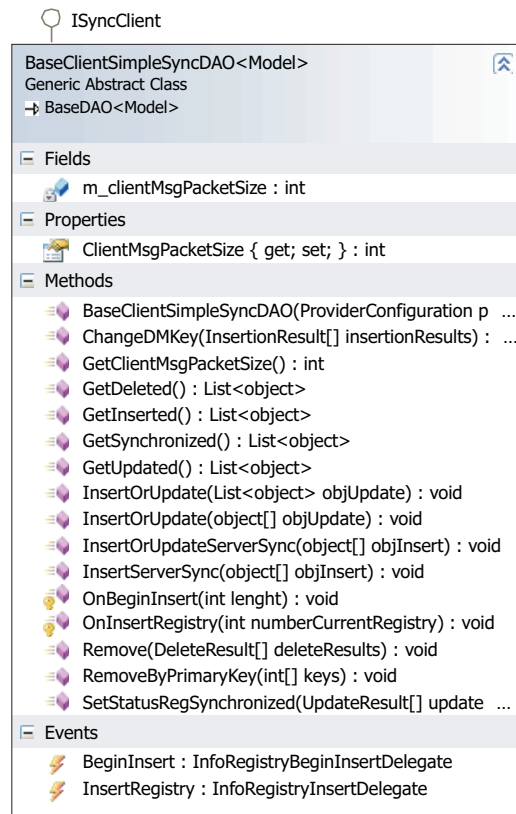
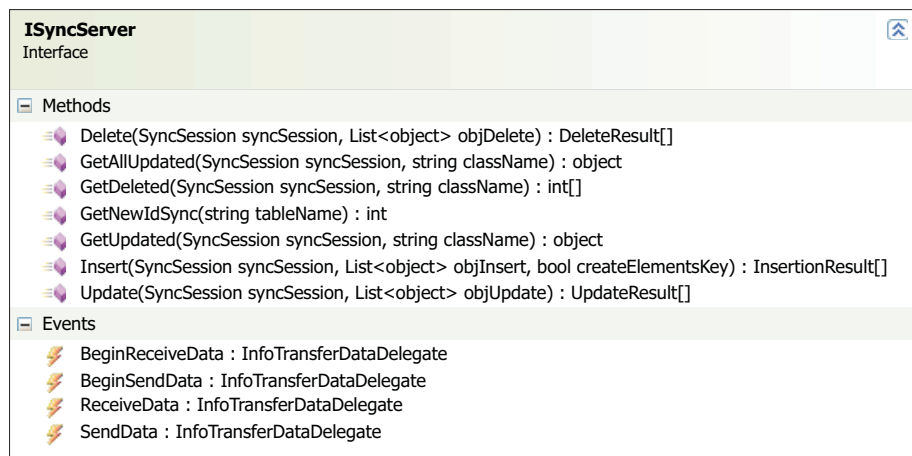
Figura A.6: Classe *ClientBaseSyncDAO*

DAOs derivadas desta classe são úteis na sincronização de tabelas que não são alteradas no dispositivo como, por exemplo, uma tabela de tipos de produto em uma aplicação de automação de força de vendas.

### A.2.3 Interface *ISyncServer*

Conforme apresentado na figura 4.9, o agente de sincronização utiliza a interface *ISyncServer* para acessar os dados do servidor central. Vários métodos disponibilizados pela interface *ISyncServer*, apresentada na figura A.8, recebem como parâmetro um objeto do tipo *SyncSession*, que armazena informações referentes às âncoras de sincronização e à identificação do dispositivo que está sincronizando. A classe *SyncSession* foi apresentada na figura A.1. Alguns métodos também recebem como parâmetro a *string* `className`, que representa o nome da classe do modelo associado à tabela sincronizada.

A interface *ISyncServer* disponibiliza métodos que permitem ao dispositivo obter e aplicar atualizações no banco de dados central, assim como eventos que permitem que o agente de sincronização obtenha e disponibilize para a aplicação no dispositivo informações referentes à quantidade de *bytes* recebidos e enviados para o servidor. Os métodos da interface

Figura A.7: Classe *ClientBaseSimpleSyncDAO*Figura A.8: Interface *ISyncServer*

*ISyncServer* são detalhados a seguir:

- *Delete(SyncSession syncSession, List<object> objDelete) : DeleteResult[]* - Exclui registros no banco de dados central e retorna o resultado da operação referente a cada tentativa de exclusão.
- *GetAllUpdated(SyncSession syncSession, string className) : object* - Retorna uma

lista com todos os registros do servidor que tenham sido criados ou modificados entre as âncoras de sincronização *IdSyncStart* e *IdSyncEnd*, presentes no parâmetro *syncSession*, independente de se quem realizou a modificação foi o próprio dispositivo que está sincronizando. Esta função é utilizada nas sincronizações do tipo *refresh*.

- *GetDeleted(SyncSession syncSession, string className) : int[]* - Retorna uma lista com os registros excluídos no servidor durante o período de desconexão.
- *GetNewIdSync(string tableName) : int* - Retorna uma nova âncora de sincronização para a tabela sincronizada.
- *GetUpdated(SyncSession syncSession, string className) : object* - Semelhante ao método *GetAllUpdated*, com a diferença de que não retorna os registros que tenham sido criados ou modificados pelo dispositivo que está realizando a sincronização. Utilizado para evitar o efeito eco detalhado na Seção 4.3.2.1.
- *Insert(SyncSession syncSession, List<object> objInsert, bool createElementsKey) : InsertionResult[]* - Insere registros no banco de dados central e retorna o resultado da operação referente a cada registro inserido. O parâmetro *createElementsKey* indica se o servidor é o responsável por gerar os identificadores únicos globais dos registros ou se os identificadores globais são definidos no próprio dispositivo e enviados para o servidor.
- *Update(SyncSession syncSession, List<object> objUpdate) : UpdateResult[]* - Atualiza registros no banco de dados central e retorna o resultado da operação referente a cada registro atualizado.

A interface também define quatro eventos, descritos a seguir, que são úteis para que o agente de sincronização obtenha informações relativas à quantidade de *bytes* enviados e recebidos do servidor.

- *BeginReceiveData : InfoTransferDataDelegate* : Evento acionado quando os dados começam a ser recebidos do servidor. Possui como parâmetro a quantidade total de *bytes* que serão recebidos.
- *BeginSendData : InfoTransferDataDelegate* : Evento acionado quando os dados começam a ser enviados para o servidor. Possui como parâmetro a quantidade total de *bytes* que serão transmitidos.
- *ReceiveData : InfoTransferDataDelegate* : Evento acionado enquanto os dados são recebidos do servidor. Possui como parâmetro a quantidade de *bytes* já recebidos.
- *SendData : InfoTransferDataDelegate* : Evento acionado enquanto os dados são enviados para o servidor. Possui como parâmetro a quantidade de *bytes* já enviados.

### A.2.4 Server Adapter

O *ServerAdapter* é o objeto que implementa a interface *ISyncServer*, permitindo a integração entre os dispositivos e o servidor. O mSync disponibiliza um *ServerAdapter* genérico que pode ser utilizado em situações onde a integração entre os dispositivos e o servidor é realizada por meio do *Web service* definido no módulo mSyncServer. Contudo, caso necessário, um programador pode implementar seu próprio *ServerAdapter* para tratar situações específicas ou utilizar outros meios de integração com o servidor que não *Web services*. A figura A.9 apresenta o *ServerAdapter* disponibilizado no *framework*.

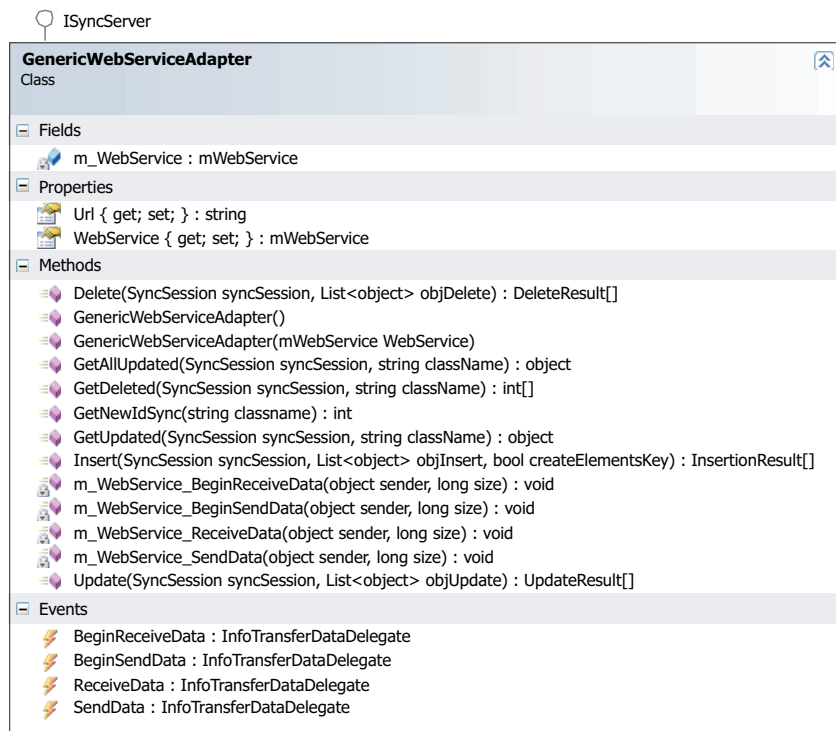


Figura A.9: *GenericWebserviceAdapter*

### mWebService

Na figura A.9 é possível observar o campo `m_WebService`, do tipo `mWebService`. O `mWebService`, definido no módulo `mSyncClient`, é uma especialização da classe `SoapHttpClientProtocol` do .NET CF, e define o *proxy* utilizado pelo dispositivo para acessar o *Web service* hospedado no servidor.

O `mWebService` disponibiliza os eventos `BeginReceiveData`, `ReceiveData`, `BeginSendData` e `SendData` exigidos pelo *ServerAdapter* para obter e disponibilizar para a aplicação informações relativas à quantidade de *bytes* recebidos e enviados para o servidor.

### A.2.5 SyncElement

O *SyncElement*, ou elemento de sincronização, encapsula as informações relativas a cada tabela necessárias à sincronização. A figura A.10 apresenta a classe *SyncElement*, cujas propriedades possuem as funções detalhadas a seguir:

- *ServerAdapter* { *get*; *set*; } : *ISyncServer* - Interface para o servidor.
- *ServerCreateElementKey* { *get*; *set*; } : *bool* - Identifica se o servidor é responsável por gerar os identificadores únicos definitivos dos registros criados no dispositivo ou se o próprio dispositivo é quem gera os registros já com os identificadores definitivos.
- *SyncClient* { *get*; *set*; } : *ISyncClient* - Interface para o banco de dados local do dispositivo.
- *SyncClientType* { *get*; *set*; } : *SyncType* - Tipo de sincronização no sentido dispositivo para servidor.
- *SyncServerType* { *get*; *set*; } : *SyncType* - Tipo de sincronização no sentido servidor para dispositivo.

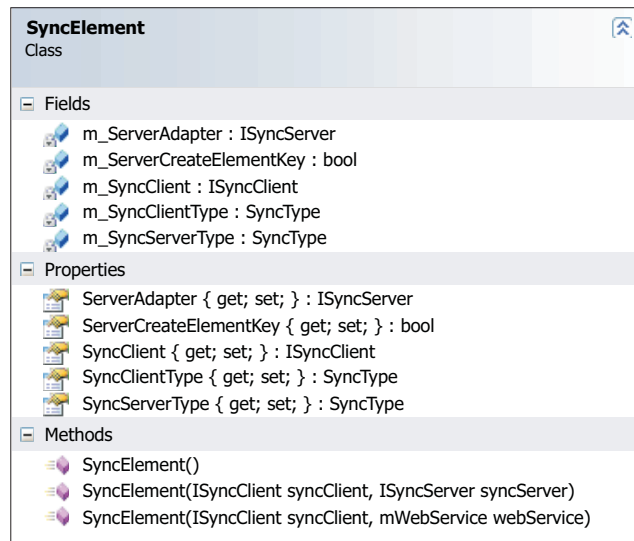
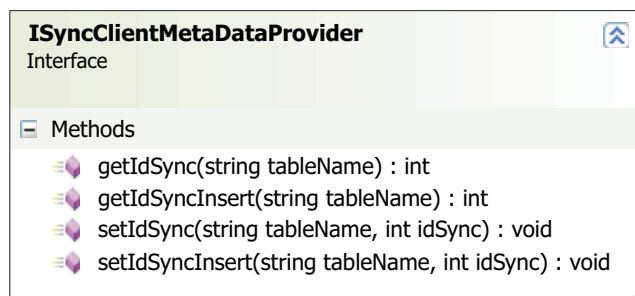


Figura A.10: *SyncElement*

### A.2.6 Interface ISyncClientMetaDataProvider

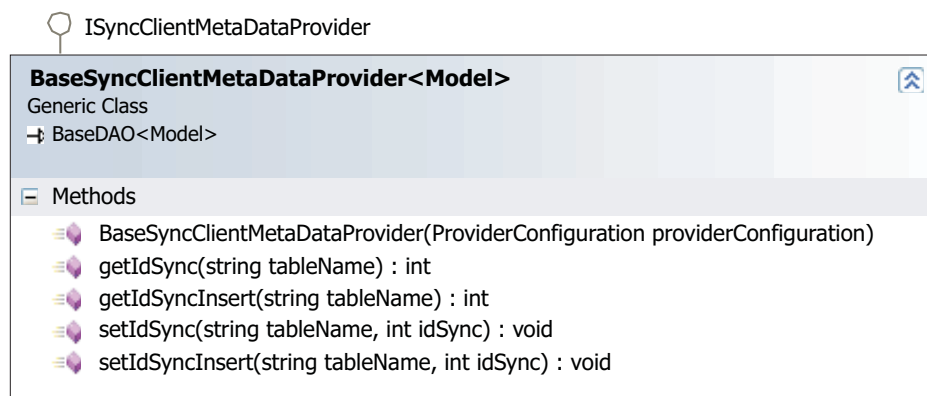
A interface *ISyncClientMetaDataProvider*, apresentada na figura A.11, é utilizada pelo agente de sincronização para obter informações relativas às âncoras de sincronização armazenadas localmente no dispositivo (ver Seção 4.1). Os métodos da interface possuem as seguintes funções:

- *getIdSync(string tableName) : int* - Retorna o valor atual do *IdSync*, que é a âncora de sincronização referente à última sincronização sentido servidor para dispositivo realizada com sucesso.
- *getIdSyncInsert(string tableName) : int* - Retorna o valor atual do *IdSyncInsert*, que é a âncora de sincronização referente à última sincronização sentido dispositivo para servidor, com envio de novos registros, realizada com sucesso.
- *setIdSync(string tableName, int idSync) : void* - Atribui um valor ao *IdSync*.
- *setIdSyncInsert(string tableName, int idSync) : void* - Atribui um valor ao *IdSyncInsert*.

Figura A.11: Interface *ISyncClientMetaDataProvider*

### A.2.7 BaseSyncClientMetaDataProvider

O mSync disponibiliza uma classe que implementa a interface *ISyncClientMetaDataProvider* e pode ser utilizada como base para o *SyncClientMetaDataProvider* em uma grande parte das aplicações.

Figura A.12: Classe *BaseSyncClientMetaDataProvider*

A classe *BaseSyncClientMetaDataProvider*, apresentada na figura A.12, armazena as informações relativas aos metadados em uma tabela do banco de dados local do dispositivo cuja estrutura é exibida na figura A.13.

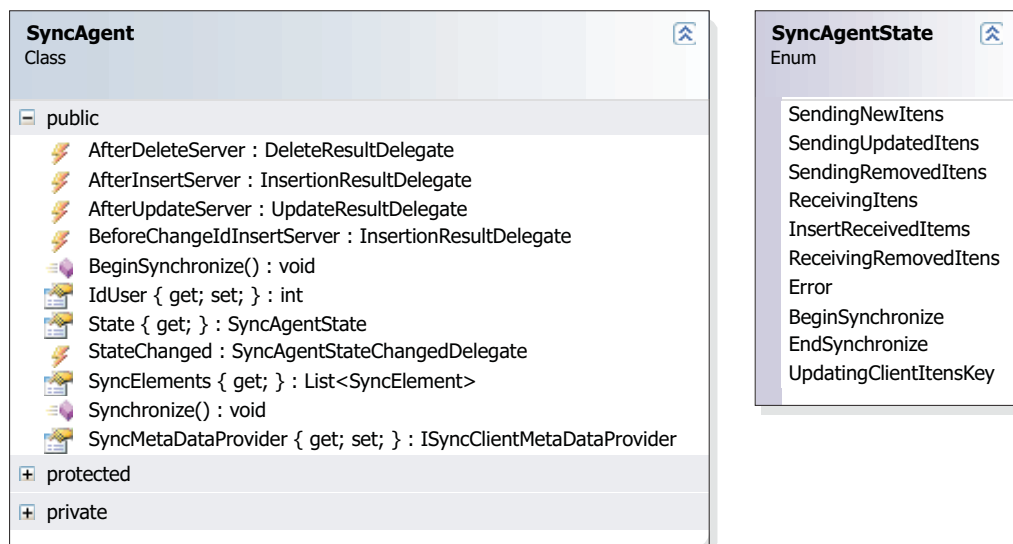
| appTables    |          |      |
|--------------|----------|------|
| tableName    | char(30) | <pk> |
| idSync       | int      |      |
| idSyncInsert | int      |      |

Figura A.13: Estrutura da tabela appTables

### A.2.8 SyncAgent

O *SyncAgent*, ou agente de sincronização, é responsável por orquestrar o processo, interagindo com cada elemento de sincronização, além de manter o estado da sessão e prover para a aplicação informações referentes ao andamento do processo e possível ocorrência de erros.

A figura A.14 apresenta a classe *SyncAgent* e os estados possíveis para o processo de sincronização. O *SyncAgent* disponibiliza para a aplicação as seguintes propriedades e métodos públicos:

Figura A.14: Classe *SyncAgent*

- `IdUser { get; set; } : int` - Identificação do dispositivo que irá realizar a sincronização.
- `State { get; } : SyncAgentState` - Estado atual do processo de sincronização. Pode ser um dos estados definidos na enumeração *SyncAgentState*, apresentada na figura A.14.
- `SyncElements { get; } : List<SyncElement>` - Lista com os elementos de sincronização que devem ser sincronizados.
- `SyncMetaDataProvider { get; set; } : ISyncClientMetaDataProvider` - *MetaDataProvider* por meio do qual o agente de sincronização tem acesso às informações relativas às âncoras de sincronização armazenadas no dispositivo.

Também são disponibilizados os métodos *Synchronize* e *BeginSynchronize*, que são responsáveis por iniciar o processo de sincronização. A diferença entre eles é que o *Synchronize* inicia o processo de sincronização de forma síncrona, onde a execução da aplicação fica parada aguardando o fim da sincronização, enquanto o *BeginSynchronize* realiza o processo de forma assíncrona, onde a aplicação no dispositivo pode continuar sua execução ou exibir mensagens para o usuário referentes ao andamento do processo enquanto a sincronização acontece.

Para permitir que a aplicação verifique a ocorrência de erros/conflitos, execute tratamentos de situações específicas e, quando pertinente, exiba informações relativas ao andamento do processo de sincronização para o usuário, o *SyncAgent* disponibiliza os seguintes eventos:

- *AfterDeleteServer : DeleteResultDelegate* - Evento acionado após o agente de sincronização transmitir para o servidor a lista de registros que foram excluídos do dispositivo durante o período de desconexão. Permite que aplicação verifique se alguma exclusão não pode ser aplicada no servidor e, caso necessário, realize os tratamentos pertinentes.
- *BeforeChangeIdInsertServer : InsertionResultDelegate* - Evento acionado após os registros criados no dispositivo serem inseridos no servidor e antes que o identificador único local (LUID) seja substituído pelo identificador único global (GUID) no banco de dados local.
- *AfterInsertServer : InsertionResultDelegate* - Evento acionado após o agente de sincronização substituir os identificadores locais (LUID) pelos identificadores globais (GUID) no banco de dados do dispositivo. Permite que a aplicação verifique a ocorrência de erros e/ou conflitos nas operações de inserção de registros no servidor, além de possibilitar que os identificadores sejam atualizados nas tabelas que utilizem a tabela sincronizada como chave estrangeira (Vieira (2007)).
- *AfterUpdateServer : UpdateResultDelegate* - Evento acionado após os registros modificados no dispositivo serem aplicados no servidor. Permite que aplicação verifique se alguma atualização não pode ser aplicada no banco de dados central e, caso necessário, realize os tratamentos pertinentes.
- *StateChanged : SyncAgentStateChangedDelegate* - Evento acionado sempre que o processo de sincronização muda de estado. Permite que aplicação monitore e exiba para o usuário informações referentes ao andamento do processo.

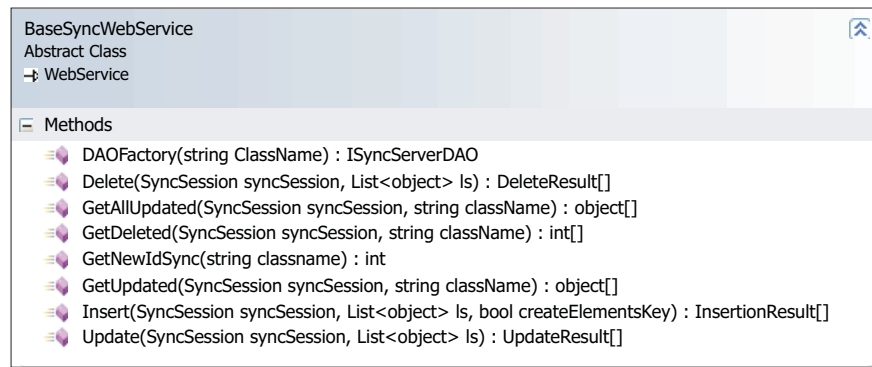
### A.3 mSyncServer

O módulo mSyncServer, que utiliza o *framework* .NET, da *Microsoft*, é executado no servidor para permitir a sincronização com os dispositivos por meio de *Web services*. Esta Seção trata das funcionalidades e principais classes definidas neste módulo.

### A.3.1 BaseSyncWebService

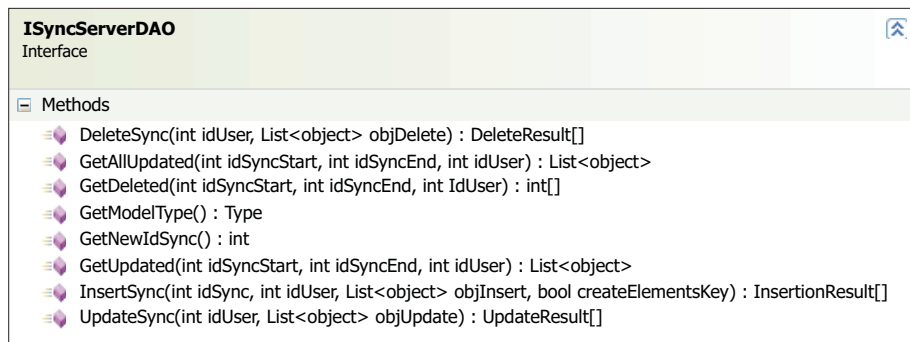
A classe abstrata *BaseSyncWebService* é a base do *Web service* utilizado no *framework*. Ele disponibiliza os métodos acionados pelo *ServerAdapter* (ver Seção A.2.4), no dispositivo, para interagir com o servidor de sincronização. Os métodos da classe *BaseSyncWebService*, apresentada na figura A.15, são detalhados a seguir:

- *DAOFactory(string ClassName) : ISyncServerDAO* : Método abstrato, que precisa ser definido nas especializações da classe, que retorna o DAO (*Data Access Object*) utilizado pelo *Web service* para interagir com banco de dados central. Os DAOs devem implementar a interface *ISyncServerDAO*, detalhada na Seção A.3.2.
- *Delete(SyncSession syncSession, List<object> ls) : DeleteResult[]* : Método do *Web service* que aciona o método *DeleteSync* do DAO referente a tabela sincronizada, excluindo do banco de dados central os registros excluídos no dispositivo.
- *GetAllUpdated(SyncSession syncSession, string className) : object[]* : Método do *Web service* que aciona o método *GetAllUpdated* do DAO referente a tabela sincronizada e obtém os registros modificados ou criados no banco de dados central durante o período de desconexão, independente se quem realizou as inserções ou modificações dos registros foi o próprio dispositivo que está sincronizando. O método é invocado pelo *Server Adapter* nas sincronização do tipo *refresh*.
- *GetDeleted(SyncSession syncSession, string className) : int[]* : Método do *Web service* que aciona o método *GetDeleted* do DAO referente a tabela sincronizada e obtém os registros excluídos no banco de dados central durante o período de desconexão.
- *GetNewIdSync(string classname) : int* : Método do *Web service* que retorna e incrementa em uma unidade o valor da âncora de sincronização referente a tabela sincronizada.
- *GetUpdated(SyncSession syncSession, string className) : object[]* : Método semelhante ao *GetAllUpdated*, com a diferença de que não retorna os registros criados ou modificados pelo próprio dispositivo que está realizando a sincronização. Utilizado nas sincronizações do tipo *incremental* para evitar o efeito eco, apresentado na Seção 4.3.2.1.
- *Insert(SyncSession syncSession, List<object> ls, bool createElementsKey) : Insertion-Result[]* : Método utilizado para inserir registros no banco de dados central. Retorna uma lista de objetos do tipo *InsertionResult* que, conforme detalhado na Seção A.1, é utilizado para substituir os LUIDs pelos GUIDs no banco de dados local dos dispositivos.
- *Update(SyncSession syncSession, List<object> ls) : UpdateResult[]* : Método utilizado para aplicar no banco de dados central as modificações efetuadas no banco de dados local dos dispositivos durante o período de desconexão.

Figura A.15: Classe *BaseSyncWebService*

### A.3.2 Interface *ISyncServerDAO*

A interface *ISyncServerDAO*, apresentada na figura A.16, é utilizada pelo *Web service* para interagir com o banco de dados central. Os DAOs (*Data Access Objects*) referentes às tabelas sincronizadas devem implementar esta interface.

Figura A.16: Interface *ISyncServerDAO*

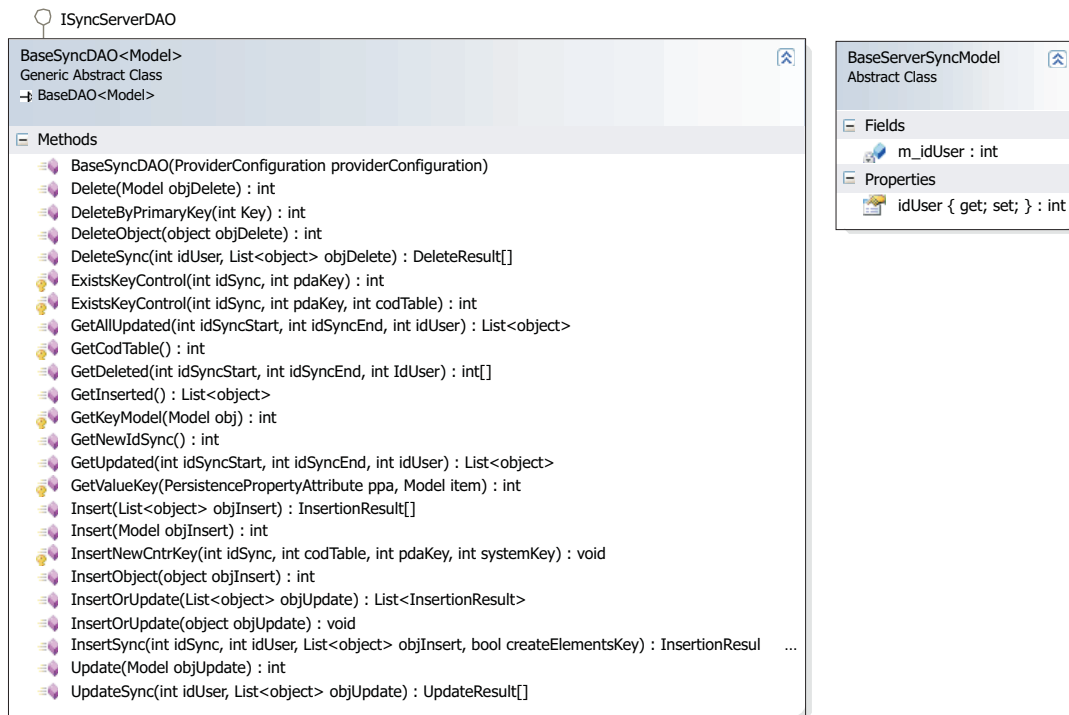
### A.3.3 *ServerSyncDAO*

O *ServerSyncDAO* implementa a interface *ISyncServerDAO*, apresentada na Seção A.3.2, e permite que o *Web service* de sincronização tenha acesso ao banco de dados central.

Deve ser definido um *ServerSyncDAO* para cada tabela sincronizada. O mSync disponibiliza duas opções de classes bases, detalhadas a seguir, que são aplicáveis a uma grande variedade de situações, mas, caso seja necessário, um programador pode implementar sua própria *ServerSyncDAO*, desde que a mesma implemente a interface *ISyncServerDAO*.

#### **BaseSyncDAO**

A figura A.17 exibe a estrutura da classe abstrata *BaseSyncDAO*. Ele pode ser utilizada como base para o DAO das tabelas que suportem sincronizações do tipo *incremental*.

Figura A.17: Classe *BaseSyncDAO*

Os métodos disponibilizados na *BaseSyncDAO* implementam as funcionalidades de rastreamento de modificações no servidor, identificação e tratamento de erros / conflitos e mapeamento de LUIDs em GUIDs, conforme discutido no Capítulo 4. A classe exige que os modelos utilizados sejam especializações da classe *BaseServerSyncModel*, que também é apresentada na figura A.17.

### BaseSimpleSyncDAO

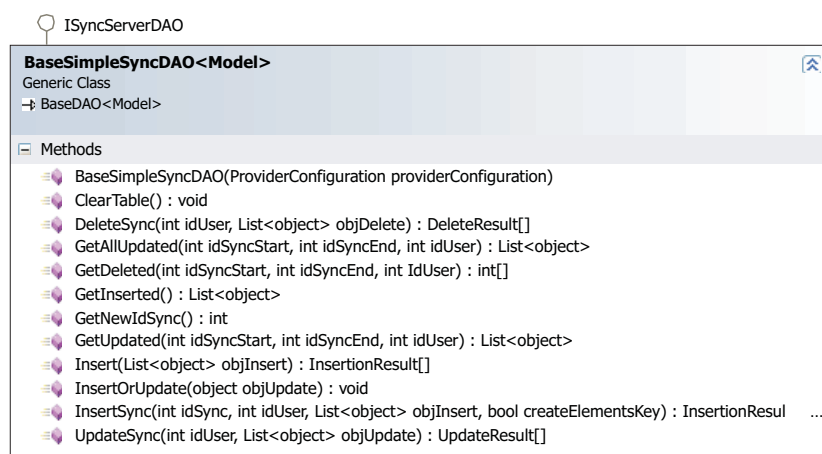
A classe *BaseSimpleSyncDAO*, apresentada na figura A.18, pode ser utilizada como base para os DAOs das tabelas que suportem apenas sincronizações do tipo *refresh*. Esta classe base não oferece suporte para as operações de rastreamento de modificações e conversão de identificadores dos registros.

## A.4 Requisitos mínimos de *hardware* e *software*

O mSync foi implementado utilizando como base o *framework* .NET (.NET Framework (2008)), possuindo os mesmos requisitos mínimos de *hardware* e *software* exigidos por este *framework*.

O módulo cliente, executado no dispositivo, é constituído pelas DLLs mSyncCommon e mSyncClient, desenvolvidas sobre o *.NET Compact Framework 2.0* (ver seção 2.3.2), e seus requisitos mínimos são apresentados a seguir:

- Sistema operacional *Windows Mobile 2003; Windows Mobile 2003 SE; Windows Mobile*

Figura A.18: Classe *BaseSimpleSyncDAO*

5.0; Windows Mobile 6.0; Windows CE 4.2; Windows CE 5.0; Windows Embedded CE 6.0 ou superior.

- O módulo cliente pode ser utilizado em conjunto com bancos de dados móveis compatíveis com a tecnologia ADO .NET (ADO .NET (2008)).
- Entre 4 MB e 5 MB de espaço de armazenamento disponível, dependendo do tipo do processador.

O módulo servidor é constituído pelas DLLs mSyncCommon, baseada no *.NET Compact Framework 2.0*, e mSyncServer, baseado no *.NET Framework 2.0*, possuindo os requisitos mínimos listados a seguir:

- Sistema operacional *Windows 2000 Service Pack 3; Windows 98; Windows 98 Second Edition; Windows ME; Windows Server 2003; Windows XP Service Pack 2* ou superior.
- O módulo servidor pode ser utilizado em conjunto com bancos de dados compatíveis com a tecnologia ADO .NET (ADO .NET (2008)).
- Espaço em disco: mínimo 280 MB (x86), 610 MB (x64).
- O *Web service* implementado no *framework* é compatível com o servidor Web IIS 6.0 ou superior (*Microsoft Internet Information Services*).

## A.5 Flexibilidade com o mSync

O mSync permite que o seu comportamento seja modificado para se adaptar às exigências específicas das aplicações. A flexibilização do mSync pode ser realizada de duas formas:

- Sobrescrevendo-se métodos das classes do *framework*. Um exemplo é apresentado no estudo de caso 2, Seção 5.2.3.1, onde os métodos *GetUpdate* e *GetAllUpdate* da classe

*BaseSyncDAO* foram sobrescritos para que apenas os registros que atendessem determinados requisitos fossem armazenados na memória local dos dispositivos.

- Criando-se classes completamente novas, desde que as mesmas implementem as interfaces exigidas pelo *framework*. Um exemplo possível é uma aplicação que não utilize *Web services* para integração entre os dispositivos e o servidor e, para isto, defina novas classes para substituir o *GenericServerAdapter* e o *SyncWebService* (ver Seção 4.5) disponibilizados pelo *framework*.

# Referências Bibliográficas

- Access Developer Network (2008). Access developer network. Disponível em: <http://www.accessdevnet.com/>. Acesso em 22/08/2008.
- ADO .NET (2008). Dev center - ado.net. Disponível em: <http://www.microsoft.com/brasil/msdn/adonet/Default.aspx>. Acesso em: 07/08/2008.
- Agarwal, S. (2008). Characteristic polynomial interpolation for fast data synchronization web site. Disponível em: <http://nislalab.bu.edu/nislalab/projects/cpisynd/index.htm>. Acesso em: 23/05/2008.
- Agarwal, S.; Starobinski, D. e Trachtenberg, A. (2002). On the scalability of data synchronization protocols for pdas and mobile devices. *Network, IEEE*, 16:22–28.
- Apple (2008). iphone dev center. Disponível em: <http://developer.apple.com/iphone/>. Acesso em: 22/05/2008.
- Black Berry Developers (2008). Black berry developers. Disponível em: <http://na.blackberry.com/eng/developers/>. Acesso em: 22/08/2008.
- Blaha, M. R.; Premerlani, W. J. e Rumbaugh, J. E. (1988). Relational database design using an object-oriented methodology. *Commun. ACM*, 31(4):414–427.
- Butrico, M.; Cohen, N.; Givler, J.; Mohindra, A.; Purakayastha, A.; Shea, D. G.; Cheng, J.; Clare, D.; Fisher, G.; Scott, R.; Sun, Y.; Wone, M. e Zondervan, Q. (2000). Enterprise data access from mobile computers: An end-to-end story. In *RIDE '00: Proceedings of the 10th International Workshop on Research Issues in Data Engineering*, p. 9, Washington, DC, USA. IEEE Computer Society.
- Canalys (2008). Smart mobile device shipments q4. Disponível em: <http://www.canalys.com/pr/2008/r2008021.htm>. Acesso em: 21/05/2008.
- Castro, C. Q. e Pakkala, T. (2007). Version vectors based synchronization engine for mobile devices. In *PDCN'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference*, pp. 309–314, Anaheim, CA, USA. ACTA Press.
- Castro, P.; Giraud, F.; Konuru, R.; Purakayastha, A. e Yeh, D. (2004). A programming framework for mobilizing enterprise applications. In *WMCSA '04: Proceedings of the Sixth*

- IEEE Workshop on Mobile Computing Systems and Applications*, pp. 196–205, Washington, DC, USA. IEEE Computer Society.
- Chauhan, V. (2006). Bandwidth efficient string reconciliation using puzzles. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1217–1225. Member-Sachin Agarwal and Senior Member-Ari Trachtenberg.
- Choi, M.-S.; Kim, Y.-K. e Chang, J. (2005). Transaction-centric split synchronization mechanism for mobile e-business applications. In *DEEC '05: Proceedings of the International Workshop on Data Engineering Issues in E-Commerce*, pp. 112–118, Washington, DC, USA. IEEE Computer Society.
- Citro, S.; McGovern, J. e Ryan, C. (2007). Conflict management for real-time collaborative editing in mobile replicated architectures. In *ACSC '07: Proceedings of the thirtieth Australasian conference on Computer science*, pp. 115–124, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.
- Coda, P. e Mayfield (1998). *Projeto de sistemas em Java: construindo aplicativos e melhores applets*. Makron Books.
- Coda, P. e Mayfield (2000). *Padrões de projeto: soluções reutilizáveis de software orientado a objetos*. Bookman.
- Cohen, N. H. (2000). A java framework for mobile data synchronization. In *CooplS '02: Proceedings of the 7th International Conference on Cooperative Information Systems*, pp. 287–298, London, UK. Springer-Verlag.
- Davidson, S. B.; Garcia-Molina, H. e Skeen, D. (1984). Consistency in a partitioned network: A survey. Technical report, Ithaca, NY, USA.
- DB2 Everyplace (2008). Db2 everyplace. Disponível em: <http://www-01.ibm.com/software/data/db2/everyplace/>. Acesso em: 04/09/2008.
- de Araujo, R. B. (2003). *Computação Ubíqua: Princípios, Tecnologias e Desafios*, pp. 45–115. XXI Simpósio Brasileiro de Redes de Computadores, Natal.
- Dong, L. e Bingjun, Y. Y. (2005). A mobile subset technic of mobile db in xml. In *Mobile Technology, Applications and Systems, 2005 2nd International Conference on*, p. 5. IEEE Computer Society.
- Eden, A. e Mens, T. (2006). Measuring software flexibility. *Software, IEE Proceedings*, 153(3):113–125.
- Fayad, M. e Schmidt, D. C. (1997). Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38.

- Fischer, M. J. e Michael, A. (1982). Sacrificing serializability to attain high availability of data in an unreliable network. In *PODS '82: Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pp. 70–75, New York, NY, USA. ACM.
- Gamma, E.; Helm, R.; Johnson, R. e Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
- Gollmick, C. (2003). Replication in mobile database environments: A client-oriented approach. In *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, p. 980, Washington, DC, USA. IEEE Computer Society.
- Grott, M. C.; de Sousa, F. C. e Hugo, M. (2004). Reutilização de soluções com patterns e frameworks na camada de negócio. *XIII Seminário de Computação - Universidade Regional de Blumenau*.
- Hansmann, U.; Nicklous, M. S. e Stober, T. (2001). *Pervasive computing handbook*. Springer, New York, NY, USA.
- Helal, S.; Hammer, J.; Zhang, J. e Khushraj, A. (2001). A three-tier architecture for ubiquitous data access. In *AICCSA '01: Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, p. 177, Washington, DC, USA. IEEE Computer Society.
- Kangasharju, J.; Lindholm, T. e Tarkoma, S. (2007). Xml messaging for mobile devices: From requirements to implementation. *Comput. Netw.*, 51(16):4634–4654.
- Kangasharju, J.; Tarkoma, S. e Raatikainen, K. (2003). Comparing soap performance for various encodings, protocols, and connections. In *Personal Wireless Communications, volume 2775 of Lecture Notes in Computer Science*, pp. 397–406. Springer-Verlag.
- Karlsson, J. S.; Lal, A.; Leung, T. Y. C. e Pham, T. (2001). Ibm db2 everywhere: A small footprint relational database system. In *Proceedings of the 17th International Conference on Data Engineering*, pp. 230–232, Washington, DC, USA. IEEE Computer Society.
- Keller, W. (1998). Object/relational access layers - a roadmap, missing links and more patterns. In *In Proceeding of EuroPlop 1998, Irsee*, pp. 1–25.
- Lam, F.; Lam, N. e Wong, R. (2002). Efficient synchronization for mobile xml data. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pp. 153–160, New York, NY, USA. ACM.
- Lee, B.-Y.; Kim, T.-W.; Kim, D.-W. e Choi, H. (2002). Data synchronization protocol in mobile computing environment using syncml. *High Speed Networks and Multimedia Communications 5th IEEE International Conference on*, pp. 133–137.

- Lee, B.-Y.; Lee, G.-H. e Kim, Y.-S. (2007). Modeling and analysis of syncml server system using stochastic petri nets. In *ICWMC '07: Proceedings of the Third International Conference on Wireless and Mobile Communications*, p. 60, Washington, DC, USA. IEEE Computer Society.
- Leonard Kawell, J.; Beckhardt, S.; Halvorsen, T.; Ozzie, R. e Greif, I. (1988). Replicated document management in a group communication system. In *CSCW '88: Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, p. 395, New York, NY, USA. ACM.
- Li, L.; Li, J. J.; Li, L. e Zhang, T. (2004). Research and realization of consistent maintenance of the mobile data replication based on the user semantics rules and transaction engine. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 1, pp. 133–139. IEEE.
- Lindholm, T. (2003). Xml three-way merge as a reconciliation engine for mobile data. In *MobiDe '03: Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, pp. 93–97, New York, NY, USA. ACM.
- Loureiro, A. A.; Nogueira, J. M. S.; Ruiz, L. B.; de Freitas Mini, R. A.; Nakamura, E. F. e Figueiredo, C. M. S. (2003). *Redes sensores sem fio*. Capítulo 4 Livro texto de mini-cursos do XXI Simpósio Brasileiro de Redes de Computadores (SBRC).
- Lu, Q. e Satyanarayanan, M. (1994). Isolation-only transactions for mobile computing. *SIGOPS Oper. Syst. Rev.*, 28(2):81–87.
- Microsoft (2008a). Windows embedded ce overview. Disponível em: <http://www.microsoft.com/windows/embedded/products/windowsce/default.mspx>. Acesso em: 22/05/2008.
- Microsoft (2008b). Windows mobile overview. Disponível em: <http://www.microsoft.com/brasil/windowsmobile>. Acesso em 22/05/2008.
- MicrosoftSyncServices (2008). Synchronization services for ado.net. Microsoft Technical Preview for Synchronization Services for ADO.NET.
- Miller, K.; Gee, C.; Inaba, R.; Ozyer, T.; Lo, A. e Alhajj, R. (2004). Synchronization of mobile xml databases by utilizing deferred views. In *Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on*, pp. 186–191. IEEE Computer Society.
- .NET Compact Framework (2008). .net compact framework - developer center. Disponível em: <http://msdn.microsoft.com/en-us/netframework/aa497273.aspx>. Acesso em: 07/08/2008.
- .NET Framework (2008). .net framework developer center. Disponível em: <http://msdn.microsoft.com/en-us/netframework/default.aspx>. Acesso em: 07/08/2008.

- Oh, S.; Bulut, H.; Uyar, A.; Wu, W. e Fox, G. (2005). Optimized communication using the soap infoset for mobile multimedia collaboration applications. In *Collaborative Technologies and Systems, 2005. Proceedings of the 2005 International Symposium on*, pp. 32–29, Indiana Univ., IN, USA. IEEE.
- Oh, S. e Fox, G. C. (2005). Hhfr: A new architecture for mobile web services principles and implementations. Technical report.
- Oracle Database Lite (2008). Oracle database lite 10g. Disponível em: <http://www.oracle.com/technology/products/lite/>. Acesso em: 07/08/2008.
- Oracle FAQ's (2008). Oracle faq's: How to avoid primary key collision on multiple sites? Disponível em: <http://www.orafaq.com/node/1406>. Acesso em: 24/10/2008.
- Paulish, D. e Carleton, A. (1994). Case studies of software-process-improvement measurement. *IEEE Computer Society Press*, 27:50–57.
- Petersen, K.; Spreitzer, M.; Terry, D. e Theimer, M. (1996). Bayou: replicated database services for world-wide applications. In *EW 7: Proceedings of the 7th workshop on ACM SIGOPS European workshop*, pp. 275–280, New York, NY, USA. ACM.
- Phatak, S. H. e Nath, B. (2004). Transaction-centric reconciliation in disconnected client-server databases. *Mob. Netw. Appl.*, 9(5):459–471.
- Poulin, J. (1994). Measuring software reusability. *Proc. of Third International Conference in Software Reuse: Advances in Software Reusability*, pp. 126–138.
- Preguiça, N.; Martins, J. L.; Cunha, M. e Domingos, H. (2003). Reservations for conflict avoidance in a mobile database system. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 43–56, New York, NY, USA. ACM.
- Ratner, D.; Reiher, P. e Popek, G. J. (2004). Roam: a scalable replication system for mobility. *Mob. Netw. Appl.*, 9(5):537–544.
- Ren, L. e Song, J. (2002). Data synchronization in the mobile internet. *Computer Supported Cooperative Work in Design, 2002. The 7th International Conference on*, pp. 95–98.
- Ren, L.; Song, M. e Song, J. (2004). A novel data type for the protocol of data synchronization,. *Computer Supported Cooperative Work in Design*, 1:532–535.
- Rocha, R. (2006). *Minidicionário da Língua Portuguesa*. Scipione, São Paulo.
- Saito, Y. e Shapiro, M. (2005). Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81.
- Sales, F. M. (2008). Gda project. Disponível em: <http://www.globogeo.com/gda/>. Acesso em: 16/10/2008.

- Serrano-Alvarado, P.; Roncancio, C. e Adiba, M. (2004). A survey of mobile transactions. *Distrib. Parallel Databases*, 16(2):193–230.
- Service Data Objects Specification (2008). Service data objects specification. Disponível em: <http://www.ibm.com/developerworks/library/j-sdo/>. Acesso em: 07/08/2008.
- SOAP Specifications (2008). Soap specifications. Disponível em: <http://www.w3.org/TR/soap/>. Acesso em: 22/06/2008.
- SQL Anywhere (2008). Sql anywhere synchronization. Disponível em: <http://www.sybase.com/detail?id=1053529>. Acesso em: 04/09/2008.
- SQL Server 2005 Home (2008). Microsoft sql server 2005. Disponível em: <http://www.microsoft.com/SQL/default.mspx>. Acesso em: 07/08/2008.
- SQL Server Mobile (2008). Microsoft sql server compact 3.5. Disponível em: <http://www.microsoft.com/sql/editions/compact/default.mspx>. Acesso em: 07/08/2008.
- Symbian (2008). Symbian os - the open mobile operating system. Disponível em: <http://www.symbian.com/>. Acesso em: 22/08/2008.
- Symbian Brasil (2008). Symbian brasil. Disponível em: <http://www.symbianbrasil.com>. Acesso em: 22/05/2008.
- SyncML Protocol (2001). Syncml representation protocol. Disponível em: <http://www.openmobilealliance.org>. Acesso em: 22/07/2008.
- Tanenbaum, A. S. (2003). *Redes de Computadores*. CAMPUS.
- Tarkoma, S.; Kangasharju, J.; Lindholm, T. e Raatikainen, K. (2006). Fuego: Experiences with mobile data communication and synchronization. *The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1–5.
- Terry, D. B.; Theimer, M. M.; Petersen, K.; Demers, A. J.; Spreitzer, M. J. e Hauser, C. H. (1995). Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pp. 172–182, New York, NY, USA. ACM.
- Tolia, N.; Satyanarayanan, M. e Wolbach, A. (2007). Improving mobile database access over wide-area networks without degrading consistency. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pp. 71–84, New York, NY, USA. ACM.
- Trachtenberg, A.; Starobinski, D. e Agarwal, S. (2002). Fast pda synchronization using characteristic polynomial interpolation. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3:1510–1519.

- Uchôa, E. M. A. (1999). Framework para integração de sistemas de bancos de dados heterogêneos. Tese de Doutorado.
- Vieira, R. (2007). *Professional SQL Server 2005 Programming*. Wiley Publishing, 15 edição.
- VisualStudio (2008). Visual studio 2008 development system. Disponível em: <http://msdn.microsoft.com/pt-br/vstudio/products/default.aspx>. Acesso em: 07/09/2008.
- W3C (2008a). Extensible markup language (xml). Disponível em: <http://www.w3.org/XML/>. Acesso em: 21/05/2008.
- W3C (2008b). Wap binary xml content format. Disponível em: <http://www.w3.org/TR/wbxml/>. Acesso em: 21/05/2008.
- Wang, A.; Reiher, P.; Bagrodia, R. e Popek, G. (1999). A simulation evaluation of optimistic replicated filing in a mobile environment. In *Performance, Computing and Communications Conference, 1999. IPCCC '99. IEEE International*, pp. 43–51. IEEE.
- Web Services Activity (2008). Web services activity. Disponível em: <http://www.w3.org/2002/ws/>. Acesso em: 22/06/2008.
- Weiser, M. (1999). *The computer for the 21st century*, volume 3, pp. 3–11. ACM.
- Weiser, M. e Brown, J. S. (1997). The coming age of calm technology. In Denning, P. J. e Metcalfe, R. M., editores, *Beyond Calculation: The Next Fifty Years of Computing*, pp. 75–85. Springer, New York.
- Ye, Q.; Sloane, A. M. e Verity, D. R. (2005). Analysis, specification and generation of mobile computer data synchronisation. In *ICMB '05: Proceedings of the International Conference on Mobile Business*, pp. 499–506, Washington, DC, USA. IEEE Computer Society.
- Zhang, J.; Helal, A. S. e Hammer, J. (2003). Ubidata: ubiquitous mobile file service. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pp. 893–900, New York, NY, USA. ACM.