Bibliografia

[1] ACHARYA, A.; BADRINATH, B. R.. **Checkpoint distributed applications on mobile computers**. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON PARALELL AND DISTRIBUTED INFORMATION SYSTEMS, p. 73–80, 1994.

[2] APPLETON, B.. **Patterns and software: Essential concepts and terminology**. In: TUTORIAL INTRODUçãO A SOFTWARE PATTERN, XI SIMPóSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, Ceará, Brasil, Outubro 1997.

[3] BADRINATH, B. R.; ACHARYA, A. ; IMIELINSKI, T.. **Impact of mobility on distributed computations**. Operating Systems Review, 27(2):15–20, 1993.

[4] BAILEY, J.; GEORGEFF, M. ; KEMP, D.. **Active databases and agent systems - a comparison**. In: PROCEEDINGS OF SECOND INTERNATIONAL WORKSHOP ON RULES IN DATABASE SYSTEMS (RIDS), p. 342–356, 1995.

[5] BARBARÁ, D.. **Mobile computing and databases - a survey**. Knowledge and Data Engineering, 11(1):108–117, 1999.

[6] BERNSTEIN, P. A.; NEWCOMER, E.. **Principles of Transaction Processing**. Morgan Kaufmann, 1997.

[7] BRAYNER, A.. **Transaction Management in Multidatabase Systems**. Informatik - Shaker Verlag, 1999.

[8] BRAYNER, A.; FILHO, J. A. M.. **Amdb: An approach for sharing mobile databases in dynamically configurable environments**. In: ANAIS DO XVII SIMPóSIO BRASILEIRO DE BANCO DE DADOS, p. 12–26, 2002.

[9] BREITBART, Y.; GARCIA-MOLINA, H. ; SILBERSCHATZ, A.. **Overview of multidatabase transaction management**. VLDB Journal, 1(2):181–293, 1992.

[10] BUCHHOLZ, S.; ZEIGERT, T.; SCHILL, A. ; HELD, A.. **Transaction processing in a mobile computing environment with alternating client hosts**. In: PROCEEDINGS OF RESEARCH ISSUES IN DATA ENGINEERING (RIDE), p. 1–8, 2000.

[11] BUSHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P. ; STAL, M.. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley and Sons, 1996.

[12] CHRYSANTHIS, P. K.; RAMAMRITHAM, K.. **Acta: a framework for specifying and reasoning about transaction structure and behavior**. In: PROCEEDINGS OF ACM SIGMOD, p. 194–203, 1990.

[13] CHRYSANTHIS, P. K.; RAMAMRITHAM, K.. **Database Transaction Models for Advanced Applications**, capítulo ACTA: The SAGA Continues, p. 349–397. Morgan Kaufmann, 1992.

[14] CHRYSANTHIS, P. K.. **Transaction processing in mobile computing environment**. In: PROCEEDINGS OF IEEE WORKSHOP ON ADVANCES IN PARALLEL AND DISTRIBUTED SYSTEMS, p. 77–83, 1993.

[15] DA SILVA, S. D.. **Sistemas de Bancos de Dados Heterogêneos: Modelo de Execução da Gerência de Transações**. Tese de Doutorado, Departamento de Informática - Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio, 1994.

[16] DA COSTA CÔRTES, S.; LIFSCHITZ, S.. **Banco de dados para um ambiente de computação móvel**. In: ANAIS DO XXIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAçãO, volume XXII JAI, p. 95–144, 2003.

[17] DA SILVA, V. T.. **From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language**. Tese de Doutorado, Departamento de Informática - Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio, 2004.

[18] DUNHAM, M. H.; HELAL, A.. **Mobile computing and databases: Anything new?** In: PROCEEDINGS OF SIGMOD RECORD, p. 5–9, 1995.

[19] DUNHAM, M. H.; HELAL, A. ; BALAKRISHNAN, S.. **A mobile transaction model that captures both the data and movement behavior**. Mobile Networks and Applications, 2(2):149–162, 1997.

[20] ELMASRI, R.; NAVATHE, S. B.. **Fundamentals of Database Systens - Third Edition**. Addison-Wesley, 2000.

[21] ELMASRI, R.; NAVATHE, S. B.. **Sistemas de Banco de Dados - Fundamentos e Aplicações - Tradução da terceira edição**. Livros Técnicos e Científcos Editora - LTC, 2002.

[22] ENDLER, M.; DA SILVA E SILVA, F. J.. **Requisitos e arquiteturas de software para computação móvel**. In: PROCEEDINGS OF I WORKSHOP DE SISTEMAS DE INFORMAçãO DISTRIBUíDOS PARA AGENTES MóVEIS (WORKSIDAM), São Paulo - Brazil, 2000. IME/USP.

[23] FRANK, L.. **Atomicity implementation in mobile**. In: PROCEEDINGS OF DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA) - CONFERENCE, p. 105–113, 1999.

[24] GEORGEFF, M.; RAO, A.. **Agent Technology - Foundations, Applications, and Markets**, capítulo Rational Software Agents - From Theory to Practice, p. 139–160. Springer, 1998.

[25] GORE, M.; GHOSH, R.. **Recovery of mobile transaction**. In: PROCEEDINGS OF DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA) - CONFERENCE, p. 23–27, 2000.

[26] GRAY, J.; HELLAND, P.; O'NEIL, P. ; SHASHA, D.. **The dangers of replication and a solution**. In: PROCEEDINGS OF ACM SIGMOD (ASSOCIATION FOR COMPUTING MACHINERY - SYMPOSIUM SPECIAL INTEREST GROUP ON MANAGEMENT OS DATA), p. 173–182, 1996.

[27] HUANG, Y.; WOLFSON, O.. **Object allocation in distributed databases and mobile computers**. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE), p. 20–29, 1994.

[28] HUANG, Y.; SISTLA, P. ; WOLFSON, O.. **Data replication for mobile computers**. In: PROCEEDINGS OF SIGMOD CONFERENCE, p. 13–24, 1994.

[29] IBM. **Db2 everyplace**. Url (http://www-3.ibm.com/software/data/db2/everyplace/), IBM, Junho 2004.

[30] IMIELINSKI, T.; BADRINATH, B. R.. **Mobile wireless computing: Solutions and challenges in data management**. Technical report, Rutgers University, U.S., 1992.

[31] IMIELINSKI, T.; BADRINATH, B. R.. **Data management for mobile computing**. SIGMOD Record, 22(1):34–39, 1993.

[32] JENNINGS, N. R.; WOOLDRIDGE, M. J.. **Agent Technology - Foundations, Applications, and Markets**, capítulo Applications of Intelligent Agents, p. 3–28. Springer, 1998.

[33] JEONG-JOON; SUH, Y.-H.; LEE, D.-I.; JUNG, S.-W.; JANG, C.-S. ; KIN, J.-B.. **Casting mobile agents to workflow systems: On performance and scalability issues**. In: PROCEEDINGS OF DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA) - CONFERENCE, p. 254–263, 2001.

[34] LAM, K.-Y.; KUO, T.-W.; TSANG, W.-H. ; LAW, G. C. K.. **Transaction shipping approach for mobile distributed real-time databases**. In: PROCEEDINGS OF DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA), p. 932–941, 1999.

[35] LAM, K.-Y.; KUO, T.-W.; TSANG, W.-H. ; LAW, G. C. K.. **Concurrency control in mobile distributed real-time database systems**. Information Systems, 25(3):261–286, 2000.

[36] LARMANN, C.. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process**, volume Second Edition. Prentice Hall PTR, 2001.

[37] LU, Q.; SATYANARAYANAN, M.. **Isolation-only transactions for mobile computing**. Operating Systems Review, 28(2):81–87, 1994.

[38] LU, Q.; SATYANARAYANAN, M.. **Improving data consistency in mobile computing using isolation-only transactions**. In: PROCEEDINGS OF FIFTH IEEE HOTOS TOPICS WORKSHOP, 1995.

[39] MADRIA, S. K.; BHARGAVA, B.. **A transaction model for improving data availability in mobile computing**. In: PROCEEDINGS OF DISTRIBUTED AND PARALLEL DATABASE, 2001.

[40] MELO, R. N.; SILVA, S. D. ; TANAKA, A.. **Banco de Dados em Aplicações Cliente-Servidor**. Editora Infobook, 1997.

[41] MICROSOFT. **Microsoft SQL Server (SQL Server CE)**. Url (http://www.microsoft.com/sql/ce/default.asp/), Microsoft, Junho 2004.

[42] NWANA, H.; NDUMU, D.. **Agent Technology - Foundations, Applications, and Markets**, capítulo A Brief Introduction to Software Agent Technology, p. 29–47. Springer, 1998.

[43] ORACLE. **Oracle9ias mobile and wireless**. Url (http://www.oracle.com/ip/deploy/ias/mobile/index.html/), Oracle, Junho 2004.

[44] ÖZSU, M. T.; VALDURIEZ, P.. **Principles of Distributed Database Systems**. Prentice Hall, 1999.

[45] PITOURA, E.; BHARGAVA, B.. **Dealing with mobility: Issues and research challenges**. Technical Report CSD-TR-93-070, Pudue University, USA, 1994.

[46] PITOURA, E.; BHARGAVA, B.. **Revising transaction concepts for mobile computing**. In: PROCEEDINGS OF FIRST IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, p. 164–168, 1994.

[47] PITOURA, E.; BHARGAVA, B.. **Building information systems for mobile environments**. In: PROCEEDINGS OF THIRD INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, p. 371–378, 1994.

[48] PITOURA, E.; BHARGAVA, B. K.. **Maintaining consistency of data in mobile distributed environments**. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, p. 404–413, 1995.

[49] PITOURA, E.; SAMARAS, G.. **Data Management for Mobile Computing**, volume 10. Kluwer Academic Publishers, 1998.

[50] PITOURA, E.; BHARGAVA, B. K.. **Data consistency in intermittently connected distributed systems**. Knowledge and Data Engineering, 11(6):896–915, 1999.

[51] PITOURA, E.; CHRYSANTHIS, P. K.. **Mobile and wireless database access for pervasive computing**. In: IEEE ICDE 2000 - TUTORIAL, 2000.

[52] PRADHAN, D.; KRISHNA, P. ; VAIDYA, N.. **Recovery in mobile wireless environment: Design and trade-off analysis**. In: PROCEEDINGS OF THE 26TH INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, p. 16–25, 1996.

[53] PRAKASH, R.; SIHGHAL, M.. **Low-cost checkpointing and failere recovery in mobile systems**. In: PROCEEDINGS OF IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 1996.

[54] PREE, W.. **Meta-patterns a means for capturing the essentials of reusable object-oriented design**. In: PROCEEDINGS OF EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP94), p. 150–162, Bologna, Itália, July 1994.

[55] PREGUICA, N. M.; BAQUERO, C.; MOURA, F.; MARTINS, J. L.; OLIVEIRA, R.; DOMINGOS, H. J. L.; PEREIRA, J. O. ; DUARTE, S.. **Mobile transaction management in mobisnap**. In: PROCEEDINGS OF Advances in Databases and Information Systems (ADBIS) - Database Systems for Advanced Applications (DASFAA), p. 379–386, 2000.

[56] SSU, K.-F.; YAO, B.; NEVES, N. F. ; FUCHS, W. K.. **Adaptive checkpointing with storage management for mobile environments, manuscript**, 1998.

[57] SAMARAS, G.; PITSILLIDES, A.. **Client/intercept: a computional model for wireless environments**. In: PROCEEDINGS OF 4 TH INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS (ICT'97), April 1997.

[58] SATYANARAYANAN, M.. **Fundamental challenges in mobile computing**. In: PROCEEDINGS OF SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, p. 1–7, 1996.

[59] SATYANARAYANAN, M.. **Pervasive computing: Vision and challenges**. Journal of IEEE Personal Communications, p. 10–17, 2001.

[60] SAYGN, Y.; ÖZGÜR ULUSOY ; ELMAGARMID, A. K.. **Association rules for supporting hoarding in mobile computing environments**, 2001.

[61] SERRANO-ALVARADO, P.; RONCACIO, C. L. ; ADIBA, M.. **Analyzing mobile transactions support for dbms**. In: PROCEEDINGS OF DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA), p. 595–600, 2001.

[62] SILBERSCHATZ, A.; KORTH, H. F. ; SUDARSHAN, S.. **Database System Concepts**, volume Third Edition. McGraw-Hill, 1997.

[63] SYBASE. **SQL anywhere 9 studio database**. Url (http://www.sybase.com/products/anywhere/), Sybase, Junho 2004.

[64] OMG. **Uml: Unified modeling language specification, version 2.0, available at: http://www.omg.org/uml/, accessed in: February 14th 2004**.

[65] UCHÔA, E. M. A.. **Framework para Integração de Sistemas de Bancos de Dados Heterogêneos**. Tese de Doutorado, Departamento de Informática - Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio, 1999.

[66] VIGNA, G.. **Mobile agents: Ten reasons for failure**. Technical report, Reliable Software Group - Department of Computer Science - University of California, Santa Barbara, 2004.

[67] VILLATE, Y.; PITOURA, E.; ILLARRAMENDI, A. ; ELMAGARMID, A. K.. **Extending the data services of mobile computers by external data lockers**. In: PROCEEDINGS OF DATABASE AND EXPERT SYSTEMS APPLICATIONS (DEXA) - WORKSHOP, p. 160–164, 2000.

[68] WALBORN, G. D.; CHRYSANTHIS, P. K.. **Supporting semantics-based transaction processing in mobile database applications**. In: PROCEEDINGS OF SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, p. 31–40, 1995.

[69] WALBORN, G. D.; CHRYSANTHIS, P. K.. **PRO-MOTION: Management of mobile transactions**. In: PROCEEDINGS OF SELECTED AREAS IN CRYPTOGRAPHY, p. 101–108, 1997.

[70] WALBORN, G. D.; CHRYSANTHIS, P. K.. **Transaction processing in pro-motion**. In: PROCEEDINGS OF 1999 ACM SYMPOSIUM ON APPLIED COMPUTING (SAC), p. 389–398, 1999.

[71] YEO, L. H.; ZASLAVSKY, A.. **Submission of transactions from mobile workstation in a cooperative multidatabase processing environment**. In: PROCEEDINGS OF CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 1994.

A
Algumas Questões Importantes Quanto ao Funcionamento do Ambiente de Computação Móvel

Este anexo apresenta um conjunto de questões importantes quanto ao funcionamento do Ambiente de Computação Móvel. Inicialmente serão destacados os estados que um equipamento móvel pode ter em suas operações, em seguida que operações são necessárias para o seu funcionamento nos momentos de desconexões, quais as implicações da fraca conectividade das redes sem fio sobre os clientes móveis e a implicação da mobilidade dos clientes. Finalmente, como é realizado o processo de recuperação em caso de falhas ocorridas no processamento em alguns sistemas móveis e seus protocolos de *checkpoint*. Uma conceituação mais abrangente pode ser encontrada em [49] e [16].

A.1
Estados de Operação de um Equipamento (Host) Móvel

Em um *sistema distribuído não-móvel*, seus equipamentos ou estão totalmente conectados ou totalmente desconectados. Já no ambiente móvel, existem vários graus de desconexão, que vão desde a desconexão total até a fraca desconexão[1]. Como resultado dessa característica do ambiente móvel, um host móvel pode operar em vários estados. Além disso, para conservar bateria, um host móvel pode operar no estado *cochilo* (*doze mode*). A figura A.1, extraída de [45], resume os diferentes estados de operação que um host pode ter.

Uma outra característica dos estados de operação dos host móveis é que eles são previsíveis. Enquanto nos ambientes distribuídos não-móveis essa característica de desconexão não é freqüente, no ambiente móvel a maioria das desconexões pode ser detectada. Assim, um protocolo especial pode ser projetado para manusear a desconexão. Além disso, visto que as desconexões

[1] Quando um terminal está conectado ao resto da rede com uma pequena largura de banda.

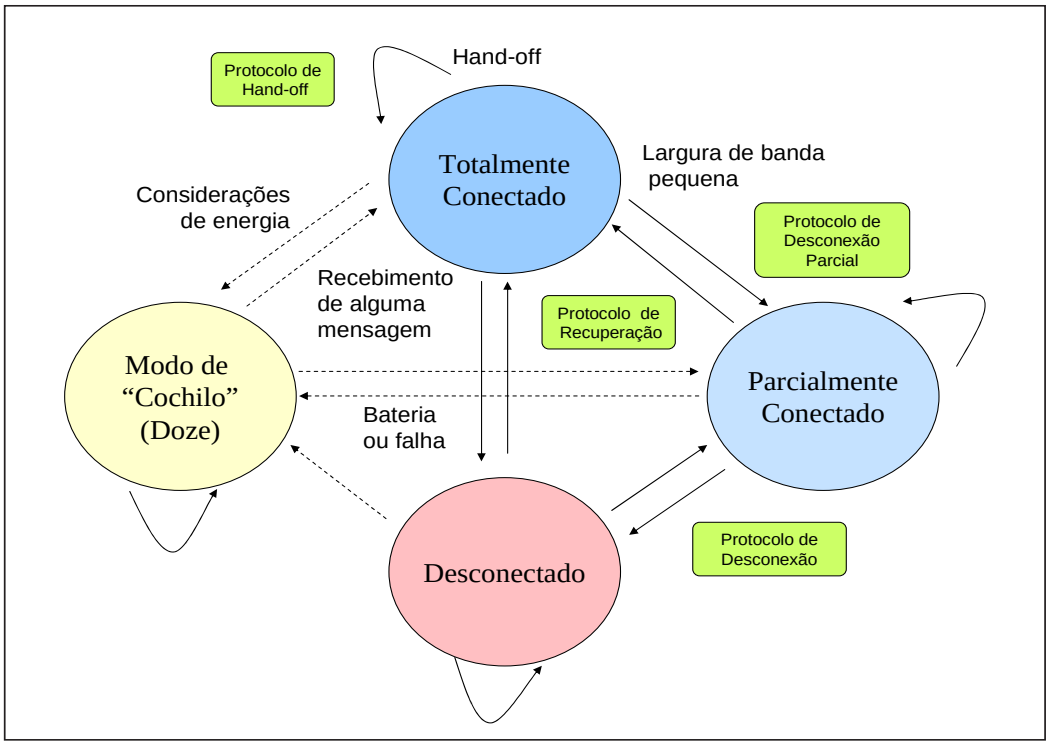

Figura A.1: Estados de operação de um host móvel

são freqüentes, idealmente um host móvel deve operar de forma autônoma mesmo durante a desconexão total.

Um host móvel pode executar um protocolo de desconexão antes de se desligar fisicamente da rede. Esse protocolo visa garantir a transferência suficiente de dados para que o mesmo continue a operar durante o estado de desconexão. Um host móvel pode trocar seu estado de *parcialmente conectado* para o estado *parcialmente desconectado*. Enquanto permanecer nesse estado, toda a comunicação com a rede deve ser limitada. Finalmente, enquanto um host móvel está operando em um modo, ele pode entrar em uma nova *célula*. Nesse caso, toda a informação pertinente ao estado do host móvel deve ser transferida para a estação de base da nova célula, através da execução de um protocolo de *hand-off*.

A.2 Estados de Operações Desconectadas de um Host Móvel

Os *hosts móveis* (ou clientes móveis) são constantemente desconectados de suas redes de comunicação. Um host móvel desconectado não pode enviar nem receber mensagens. Assim, um host móvel que se desconecte no meio da execução de um processo (algoritmo) pode causar a suspensão da tarefa em

execução até a sua nova conexão com a rede fixa. Desconexão no ambiente móvel é diferente de falha, pois ela pode ser voluntária ou involuntária. As desconexões são eleitas (*elective*). Assim, um host móvel pode informar ao sistema, *a priori*, uma iminente desconexão e, então, executar um protocolo específico para desconexão antes do seu desligamento físico da rede de comunicação. Desconexões são constantemente esperadas, o que as torna uma característica típica do ambiente móvel [3].

Um outro estado dos hosts móveis com o objetivo de reduzir o consumo das baterias, principalmente em palmtops e laptops, é o *cochilo* (*doze mode*). Nesse estado, a velocidade do *clock* do processador é reduzida e nenhuma operação do usuário é executada. O host móvel simplesmente aguarda, passivamente, o recebimento de mensagens enviadas pelos demais componentes da rede, retornando ao modo normal (de execução) quando uma mensagem é recebida. Como no caso da desconexão, essa é uma operação voluntária. Entretanto, as implicações são diferentes. No modo *cochilo*, o host móvel é alcançável pelo resto do sistema e, assim, pode ser induzido, pelo sistema, a retomar o seu modo de operação normal. Diferentemente dos dois tipos de desconexões descritos, a conexão com o sistema *somente* pode ser retomada pelo host móvel [3].

A idéia por trás do suporte às operações desconectadas é simples. Quando uma desconexão é percebida antecipadamente, os itens de dados são transferidos para o cliente móvel (host móvel), para possibilitar a sua operação durante o período de desconexão. A carga antecipada dos dados para garantir a sobrevivência durante a desconectividade que está por vir é chamada de **hoarding**. Segundo [49], a desconexão pode ser descrita como a transição entre três estados:

1. Carga Antecipada de Dados (*Hoarding*);
2. Operações Desconectadas; e
3. Reintegração.

A figura A.2, extraída de [51], resume os diferentes modos de operação em que um host móvel pode estar.

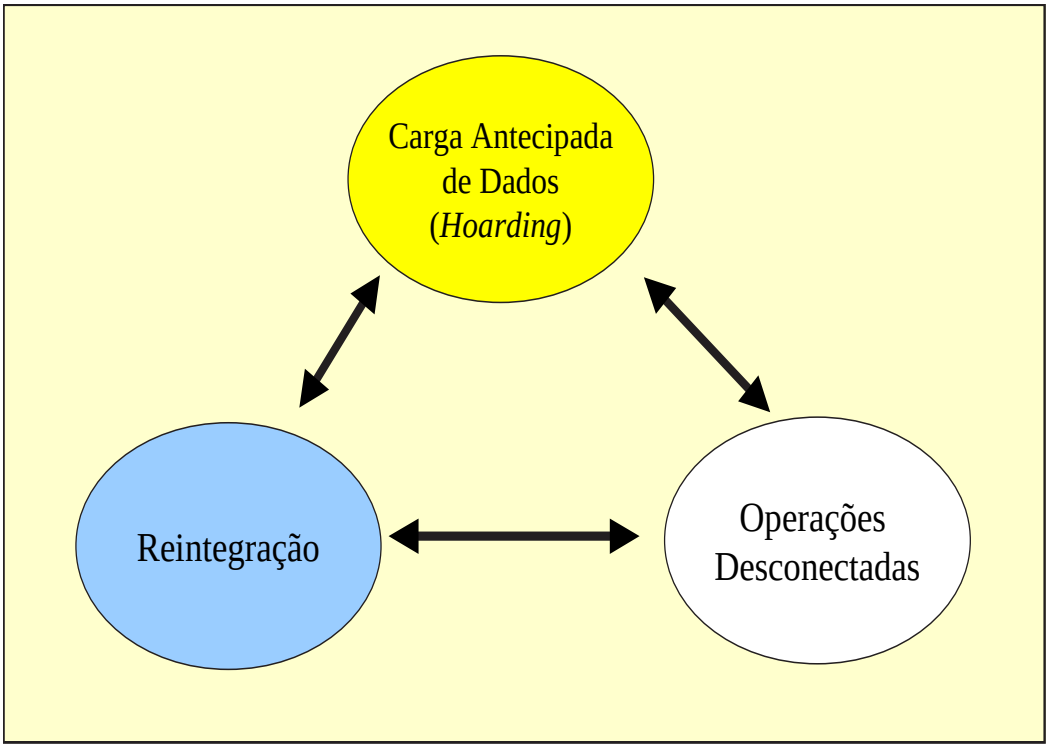

Figura A.2: Estados de operações desconectadas de um host móvel

A.2.1 Carga Antecipada de Dados

Antes da desconexão, o host móvel está no estado de *Carga Antecipada de Dados (Hoarding)*. Nesse estado, os itens de dados necessários para operação são carregados na unidade móvel. Os itens podem ser simplesmente realocados (mudados de endereço) ou movimentados do host fixo para a unidade móvel. Entretanto, fazendo isso, esses itens de dados se tornarão inacessíveis para outros sites. Como alternativa, os itens de dados podem ser replicados ou mantidos na memória cache da unidade móvel. O tipo de dados transferidos para o host móvel depende da aplicação e do modelo de dados que o sustenta. Por exemplo, no caso de *sistemas de arquivos*, os dados podem ser arquivos, diretórios ou volumes de disco. No caso de *SGBDs*, os dados podem ser relações ou visões. No caso de sistemas de *browsing na Web*, os dados podem ser páginas. No caso do *modelo de objetos*, os objetos de dados podem carregar consigo informações adicionais, tais como o conjunto de operações permitidas. No caso do *modelo baseado em agentes móveis*, os agentes podem ser transmitidos através da rede para serem executados no cliente móvel. Para desconexões previsíveis, como, por exemplo, a motivada pela entrada do cliente em uma região fora da sua área original de cobertura e, portanto, mais cara, os dados podem ser carregados imediatamente antes da desconexão [49].

Uma questão complicada quando um cliente móvel está nesse estado é como antecipar suas necessidades futuras de dados. Uma abordagem simplista é permitir que o usuário especifique, explicitamente, que itens de dados necessita carregar para suas aplicações. Uma outra abordagem é usar as informações sobre o histórico dos acessos aos dados para predizer a provável necessidade futura dos dados. Que dados carregar depende também da aplicação que o sistema irá usar. Por exemplo, pode precisar de arquivos para um processador de texto ou uma tabela atualizada de preços. Além das questões anteriores, uma outra situação complexa para a *carga antecipada dos dados* ocorre quando os dados carregados são compartilhados por vários clientes ou sites, que necessitam continuar acessando e modificando os dados com o host móvel desconectado. Assim, avaliando a probabilidade de operações conflitantes por ocasião da escolha de que itens de dados devem ser carregados antecipadamente, pode-se melhorar a efetividade das operações desconectadas. Em [60], existe uma abordagem muito interessante para especificação da carga antecipada de dados utilizando-se uma técnica de mineração de dados (*data mining*) chamada de *regras de associação*.

A.2.2 Operações Desconectadas

Quando acontece uma ruptura de conexão com a rede fixa, a unidade móvel entra no estado de *desconexão*. Enquanto estiver nesse estado, ela somente pode usar dados locais. Solicitações de dados que não estão disponíveis localmente não podem ser atendidas e retornam com indicação de erro. Essas pendências podem ser organizadas em uma fila para serem atendidas, quando ocorrer a próxima conexão. Aplicações com solicitações de dados não-atendidas também podem suspender sua execução ou continuar trabalhando em outro processamento que não dependa dos dados solicitados.

Existem duas abordagens para reparar as atualizações dos dados compartilhados: a *abordagem pessimista* e a *abordagem otimista*. Na abordagem pessimista, as atualizações são executadas somente em um site, utilizando bloqueio (*lock*) ou alguma forma de verificação dos dados de entrada e saída (*check-in/check-out*). Na abordagem otimista, as atualizações são permitidas em mais de um site, com a possibilidade perigosa de conflito entre as operações. As atualizações nas unidades móveis são registradas (*logged*) na memória estável do cliente. As informações são armazenadas no *log (diário)* do sistema. O tipo de informação guardada afeta a eficácia da reintegração das atualizações quando da reconexão, assim como a eficácia da otimização do *log*. A otimização do

log, que o mantém em um tamanho pequeno, é muito importante, pois esse log é armazenado na memória do cliente móvel e é propagado para a aplicação das atualizações e reintegração, durante a reconexão. Operações de otimização no *log* podem ser realizadas durante as operações, quando a unidade móvel está desconectada; incrementalmente, cada vez que uma nova operação é inserida no log; ou como um passo de pré-processamento, antes da propagação ou aplicando o log na próxima reconexão [49].

A.2.3 Reintegração

Quando acontece uma nova conexão com a rede fixa, a unidade móvel entra no estado de *reintegração*. Neste estado, as atualizações do host móvel são reintegradas com as atualizações de outros sites, reexecutando o seu log no host fixo. Questões referentes à concorrência e seriação das transações são efetivadas em função de cada sistema em particular, visando resolver os problemas de atualização.

A Tabela A.1, extraída de [49], apresenta os principais problemas e questões relacionados ao estado de desconexão de uma unidade móvel.

A.3 Desconexão e os SGBDs

Existem muitos problemas que permanecem sem solução no que se refere à questão da carga de dados antecipada (*hoarding*) em banco de dados. A primeira questão em aberto consiste em determinar qual deve ser a granularidade dos dados para a carga antecipada. Por exemplo, no caso dos SGBDs relacionais, pode ser uma tupla, um conjunto de tuplas ou toda uma relação. Uma abordagem lógica seria carregar os dados em função das consultas realizadas, por exemplo, carregando antecipadamente de forma progressiva (*prefetching*) apenas os dados que constituem a resposta a uma determinada pergunta. Isso corresponde a carregar, na unidade móvel, uma *visão materializada*. Uma outra questão consiste em decidir que dados carregar. Em termos de visões, significa identificar quais devem ser as visões materializadas ou especificar quais consultas necessitam das cargas relativas às visões. Usuários podem, explicitamente, especificar quais são as consultas que precisam de carga antecipada. Uma outra alternativa consiste na avaliação do comportamento das consultas anteriormente realizadas, para que o próprio sistema tente identificar prováveis

Estado	*Problemas e Questões*	*Abordagens*
Hoarding	Unidade de carga	- Depende do sistema (por exemplo, arquivos ou fragmentos de banco de dados)
	Que itens carregar?	- Especificado pelo usuário - Usar as informações sobre o histórico dos acessos aos dados - Dependente do tipo da aplicação
	Quando executar a carga dos dados?	- Antes da desconexão - Em uma base regular
	Solicitação para dados não-disponíveis localmente	- Retorna uma exceção/erro - Solicita uma fila para futuros serviços
Desconexão	O que pôr no log?	- Valores dos dados - Timestamps - Operações
	Quando otimizar o log?	- Nas operações locais (incrementação) - Antes da integração
	Como otimizar o log?	- Depende do sistema
Reintegração	Como integrar?	- Reexecutando as operações do log
	Como resolver os conflitos?	- Usar semântica das aplicações - Soluções automáticas - Ferramentas para ajudar o usuário

Tabela A.1: Principais problemas e questões relacionadas ao estado de desconexão de uma unidade móvel

futuras necessidades dos usuários. Em ambos os casos, o sistema carregaria, automaticamente, o conjunto de dados mais comumente usado ou referenciado por último nas consultas dos usuários.

Existem algumas particularidades que se deve considerar quanto ao problema de ruptura (divisão) das redes, particularmente na identificação do local ideal para se executar algum processo. A ruptura da rede não corresponde a um comportamento de falha, visto que desconexões são comuns na computação móvel, exigindo um tratamento diferenciado dos sistemas distribuídos tradicionais [49].

Uma tendência comum é fazer a confirmação (*commit*) da transação de forma temporária na unidade móvel, tornando seus resultados visíveis apenas para uma próxima transação na mesma unidade móvel. Por exemplo, um vendedor de bebidas pode ter carregado, em seu host móvel, um valor de estoque para cada tipo de bebida, e a cada venda realizada essa quantidade é atualizada, diminuindo a quantidade vendida do respectivo saldo de estoque. Na reconexão, ocorre um processo de certificação, durante o qual toda transação temporariamente confirmada (*committed*) no host móvel é validada contra uma

aplicação ou por um critério de exatidão definido pelo sistema. Se o critério for satisfeito, então a transação é confirmada. Por exemplo, o critério no exemplo anterior poderia ser existir bebidas em estoque para entrega no prazo combinado entre o vendedor e o comprador, uma vez que o estoque total da empresa é fornecido diariamente a todos os vendedores. Caso o critério não seja satisfeito, a transação deve ser cancelada, reconciliada ou compensada. Isso realmente se torna um problema, pois essas ações podem ter efeito em cascata sobre as transações temporariamente confirmadas no host móvel. Para resolver esse problema [37] e [38] introduziram o conceito de Isolation-only Transactions (IOTs), e [26], o conceito de transações no esquema de *Two-Tier Replication* para computação móvel.

A.4
Fraca Conectividade

Fraca conectividade é a conectividade fornecida por redes de comunicação de dados lentas e/ou monetariamente caras para seu usuário. Além disso, nessas redes, a conectividade é oferecida em curtos períodos de tempo. A fraca conectividade causa várias limitações que não estão presentes quando a conectividade é normal, estimulando a revisão de vários protocolos. Uma outra característica da computação móvel é a variação da intensidade da conectividade. Conectividade, em computação móvel, varia no custo, na largura de banda e na segurança/confiança. Muitas propostas para tratamento da fraca conectividade examinam essas características e fornecem suporte para as operações. Nesses sistemas, operações desconectadas são exatamente a forma de operação no extremo total da falta conectividade. O objetivo da maioria das propostas para tratamento da fraca conectividade é a prudência no uso da largura de banda. Elas aceitam trocar a confiança (exatidão) do dado transmitido pela redução dos custos na comunicação [49].

Fraca Conectividade e os SGBDs

Abordagens para tratar a fraca conectividade em SGBDs objetivam minimizar a comunicação e sobreviver a pequenas desconexões. Entretanto, devido à complicada dependência entre os itens de dados do banco de dados, o problema é complexo. Um host móvel pode desempenhar vários papéis no ambiente do banco de dados distribuído. Por exemplo, ele pode simplesmente submeter operações para serem executadas em um servidor da rede fixa ou pode realizar o processamento no próprio local. Assim, é necessário permitir operações

autônomas durante a desconexão, o que complica a gerência dos dados e pode causar um aumento inaceitável (*overhead*) na comunicação. Colocar os dados no host móvel também causa novos desafios no projeto físico do banco de dados, tais como a fragmentação apropriada do banco de dados e a alocação dos fragmentos no host móvel e fixo. O fragmento também pode ser replicado em alguns sites ou em todos os sites, ou mantido como uma única cópia. Em geral, manter os dados como uma única cópia dos dados no host móvel não é apropriado em função da disponibilidade e segurança dos mesmos [49].

O controle de concorrência, nos casos de transações que envolvem tanto host móveis quanto fixos, é complicado. Transações que acessam dados tanto nos hosts móveis quanto nos fixos, via conexões sem fio, terão um maior *overhead*. Por exemplo, no caso do protocolo de controle de concorrência pessimista, que requer que a transação adquira bloqueios (*locks*) em vários sites, as transações podem ser canceladas se pedirem bloqueios a sites que estão desconectados ou pedirem bloqueios a sites bloqueados por transações desconectadas. Por outro lado, técnicas tais como *timestamps* podem levar a um grande número de transações canceladas (abortadas) em função da lentidão das operações na rede. Para evitar os atrasos decorrentes da lentidão das conexões sem fio, um modelo de transações, chamado de *Mobile Open Nested Transactions*, foi proposto por [14].

A.5 Mobilidade

Um sistema distribuído que inclua elementos móveis é dito dinâmico. Unidades móveis se comunicam diretamente com diferentes estações de base, enquanto se movimentam dentro e para fora de sua área, entrando em uma nova célula. A *mobilidade* também significa que os clientes se movem enquanto executam atividades de leitura/gravação para realizar suas funcionalidades. Por essa razão, alguns dados ou algumas tarefas também precisam se movimentar, juntamente com as unidades móveis, principalmente por questões de desempenho. Muitos sites se tornam indisponíveis durante a desconexão. Além disso, o custo da comunicação, na computação móvel, é diferente do custo da computação fixa, primeiro porque existe a necessidade de se determinar exatamente a localização do host móvel e segundo porque é baseado na duração da conexão e não no número das mensagens enviadas. Assim, os algoritmos para distribuição de dados e tarefas dos sistemas distribuídos precisam ser revisados para a computação móvel.

Muitos parâmetros se modificam, com o tempo, no ambiente da computação móvel. Assim, deixar os dados alocados em locais fixos pode não ser apropriado. Em [27] é feita uma distinção entre os custos de replicação, na computação fixa e na computação móvel, com base no pressuposto de que, na computação móvel, o custo de entrada/saída de dados se torna insignificante, em função de outras características da comunicação sem fio. Também é introduzido um algoritmo de alocação dinâmica de dados que tem a propriedade de, sempre que um processador lê uma cópia, salvar essa cópia em um banco de dados local. Esse algoritmo dinâmico demonstra ser superior ao esquema de alocação estática.

Em [28] é estudado o caso em que computadores móveis acessam dados de forma on-line em hosts fixos. O problema discutido é se é benéfico, em termos de custo de comunicação, guardar uma cópia dos dados localmente no host móvel. Intuitivamente, guardar uma cópia local é apropriado para itens de dados que são freqüentemente lidos, quando comparados com a sua taxa de gravação.

A.6 Recuperação de Falhas

Recuperação, em ambiente móvel, tem se concentrado em gravar *checkpoints* (pontos de verificação) de consistência global para a aplicação distribuída móvel. O processamento de recuperação para transações móveis, dentro do contexto desses algoritmos de *checkpoints*, não está ainda bem amadurecido. As características do ambiente móvel que são levadas em consideração na definição dos esquemas de *checkpoints* são as seguintes:

- *A mobilidade dos hosts móveis de célula para célula*, que influencia na decisão de encontrar o melhor lugar para armazenar o próximo local do *checkpoint*;
- *A disponibilidade de armazenamento estável no host móvel*, que determina o grau de participação do host móvel no processo de *checkpoint*;
- *A largura de banda*, que, juntamente com a disponibilidade de armazenamento estável, refina a participação do host móvel no *checkpoint* e afeta a freqüência de sua realização.

A.6.1 Checkpoints no Ambiente Móvel

Em sistemas distribuídos, a recuperação de falhas é baseada nos protocolos de *checkpoints*. Esses protocolos, periodicamente, armazenam o estado da aplicação mídia persistente (seguro). Após uma falha, a aplicação usa esses *checkpoints* para desfazer (*roll back*) até o último ponto salvo e reiniciar a execução. Um estado global é armazenado e usado pelo protocolo para recuperar a aplicação. O estado global inclui o estado de cada processo participante na aplicação distribuída e, possivelmente, muitas mensagens. Para uma correta recuperação, o protocolo deve conservar recuperável o estado global da aplicação. Maiores detalhes sobre protocolos em sistemas distribuídos podem ser encontrados em [44].

Em [1], um *checkpoint global* é definido como um *checkpoint local* consistente para cada host móvel/processo participante da aplicação. Considerando que os hosts móveis enviam e recebem mensagens, [56] apresentou as seguintes condições de recuperação e consistência:

- **Consistência**: Um *checkpoint global* é consistente se as seguintes condições são satisfeitas: para cada mensagem *m*, se o *recebedor da mensagem (m)* é incluído no *checkpoint global*, então o *emissor da mensagem (m)* também é incluído no *checkpoint global*.
- **Recuperabilidade**: Para evitar a perda de mensagens *em trânsito*, que foram enviadas mas não recebidas por qualquer outro processo, se o *checkpoint global* contiver o *emissor da mensagem (m)* mas não contiver o *recebedor da mensagem (m)*, então o *protocolo de checkpoint* deve salvar a *mensagem (m)* também.

Os algoritmos de *checkpoint* são classificados em duas categorias: *coordenados* e *não-coordenados*. *Os protocolos coordenados* necessitam que cada participante coordene seus *checkpoints locais* para garantir a recuperação e a consistência do *checkpoint global*. *Os protocolos não-coordenados*, por sua vez, permitem aos participantes *checkpoints* independentes em seu estado local. Durante a recuperação, um esforço de coordenação é necessário para selecionar um *checkpoint* para cada participante e criar o *checkpoint* global consistente.

Devido à mobilidade, pequena largura de banda e desconexões, ambos os tipos de protocolos, em sua forma original, são inadequados para a computação móvel. *Protocolos coordenados* requerem controle de mensagens enviadas a diferentes hosts móveis para sincronizar o processo de *checkpoint*. Isso envolve um custo de pesquisa para encontrar a localização do host móvel.

O processo pode ser ainda mais complicado, pois o host móvel pode se movimentar para outra célula antes de o processo de *checkpointing* ter sido concluído. Os esquemas coordenados também são afetados pela desconexão. Durante a desconexão, os *checkpoint locais* dos hosts móveis ficam inacessíveis para o algoritmo de sincronismo do *checkpoint*, tornando assim a execução problemática. *Protocolos não-coordenados* permitem aos participantes móveis o estado de *checkpoint* sem troca de mensagens. Isso é adequado para o ambiente móvel, porque permite que o processo de *checkpointing* continue durante a desconexão. Entretanto, mensagens necessitam ser trocadas durante a recuperação, para encontrar o *checkpoint global* ou pegar informações sobre outros participantes [49].

A.6.2 Protocolos de Checkpoints Móveis

Um pequeno número de adaptações para os protocolos de *checkpoint* coordenados e não-coordenados tem sido desenvolvido para lidar com as limitações do ambiente de computação móvel. Esses protocolos podem ser classificados com base em seu grau de adaptabilidade e se os armazenamentos estáveis são considerados um lugar relativamente seguro para armazenar o estado local da aplicação. Assim, os *checkpoints* são classificados como *soft* ou *hard* com base nos *tipos de falhas* que podem ocorrer. *Falhas soft* são aquelas que não causam danos permanentes aos host móveis, como, por exemplo, descarga de bateria ou problemas no sistema operacional. Já as *falhas hard* são aquelas que causam danos permanentes aos host móveis. *Falhas hard* são manuseadas pelos *checkpoints hard*, que são armazenados na rede fixa, enquanto as *falhas soft* são tratadas pelos *checkpoint soft*, que são armazenados localmente no host móvel. *Checkpoints* armazenados localmente são fáceis de serem criados e, sobretudo, permitem que o host móvel continue funcionando durante a desconexão [49].

A Tabela A.2, extraída de [49], apresenta características dos grupos de protocolos em função do grau de conectividade.

Entre os trabalhos de destaque, constam [1] e [52], que apresentam três protocolos não-coordenados que consideram os armazenamentos em disco dos hosts móveis instáveis e, conseqüentemente, inapropriados para armazenar o estado de um participante. [56] e [53], entretanto, consideram que os hosts móveis têm um armazenamento estável relativamente seguro e, assim, podem participar no processo de *checkpoint* como um host fixo.

	Conectado	**Desconectado**	**Fraca Conectividade**
Tipo de Login	Imediato ou periódico	Periódico	Periódico
Razão dos *Checkpoints*	$hard = soft$	Somente soft	$soft > hard$
Tipo de Coordenação	Coordenado ou não-coordenado	Não-coordenado	Não-coordenado
Clientes com poucos recursos (armazenamento estável inseguro)	Equilibrado ou minimizado o uso do cliente	Maximizado o uso do cliente (somente memória do cliente)	Maximizado o uso do cliente
Clientes com muitos recursos (armazenamento estável seguro)	Equilibrado ou maximizado o uso do cliente	Maximizado o uso do cliente (somente memória do cliente)	Maximizado o uso do cliente

Tabela A.2: Conectividade e *checkpoint*

B
Revisão Conceitual para Formalização do Modelo de Transações Proposto

A seguir, será apresentado o framework ACTA para formalização de algumas definições de uso geral e especificação dos tipos de dependências entre transações, ambas relevantes à compreensão do modelo proposto neste trabalho [6] e [13].

B.1
Formalização de Algumas Definições de Uso Geral

Definição 1: Sejam **obj**$_p$ e **obj**$_k$ dois objetos definidos em um sistema de computação. Dizemos que **BD** é um Banco de Dados se e somente se:

1. BD $= \bigcup_{i=1}^{n}$ obj$_i$
2. $\forall$ i $\leq$ m, j $\leq$ n, i$\neq$j: obj$_i$ $\bigcap$ obj$_j$ $= \emptyset$

Assim, pode-se dizer que um banco de dados é uma coleção de objetos.

Definição 2: A invocação de uma operação sobre um objeto é chamada de **evento de objeto**. O tipo de um objeto define as operações e os eventos possíveis para o objeto. Será usado o termo **op**$_t$**[obj]** para denotar o evento correspondente à invocação da operação **op** sobre o objeto **obj** pela transação **t** e **E-Obj**$_t$ para denotar o conjunto de eventos que pode ser invocado[1] pela transação **t** (por exemplo, **op**$_t$**[obj]** $\in$ **E-Obj**$_t$**)**.

Definição 3: O efeito de uma operação **op** invocada pela transação **t** sobre um objeto **obj** é permanente no banco de dados quando **op**$_t$**[obj]** é confirmado (*commit*).

[1] O sentido do termo "invocar um evento"é o mesmo de "causar a ocorrência do evento".

Definição 4: O efeito de uma operação **op** invocada pela transação **t** sobre um objeto **obj** é cancelado, no banco de dados, quando $\mathbf{op}_t$**[obj]** é cancelada (*abort*).

Definição 5: A invocação de uma primitiva do gerente de transações é denominada **evento significativo**. Um modelo de transações define os eventos significativos que são necessários para as transações aderirem ao modelo. O conjunto de eventos significativos de uma transação **t** é denotado por $\mathbf{ES}_t$.

Definição 6: Eventos de Iniciação, denotados por $\mathbf{EI}_t$, correspondem ao conjunto de *eventos significativos* que podem ser invocados para iniciar a execução de uma transação **t**, desde que $\mathbf{EI}_t \subset \mathbf{ES}_t$.

Definição 7: Eventos de Terminação, denotados por $\mathbf{ET}_t$, correspondem ao conjunto de *eventos significativos* que podem ser invocados para terminar a execução de uma transação **t**, desde que $\mathbf{ET}_t \subset \mathbf{ES}_t$.

Definição 8: A execução de uma transação **t** é uma ordem parcial de eventos $\mathbf{E}_t$, com uma relação de ordenação $\angle_t$, onde:

1. $E_t \subseteq (\text{E-Obj}_t \bigcup ES_t)$
2. $\angle_t$ denota a ordem temporal em que ocorrem os eventos invocados por t.

E_t contém eventos que são tanto eventos de objetos permitidos de serem invocados por **t** como eventos significativos relacionados a **t**.

Definição 9: Um histórico **H**, de uma execução concorrente de um conjunto de transações **T**, contém todos os eventos associados com as transações em **T** e indica a ordem (*parcial*) em que esses eventos ocorrem. $\mathbf{H}_{atual}$ é usado para denotar a história dos eventos que ocorrerem até um determinado ponto no tempo.

Definição 10: O predicado $\epsilon \rightarrow \epsilon$' é verdadeiro se o evento ϵ precede o evento ϵ' no histórico **H**. Do contrário, é falso. Assim, $\epsilon \rightarrow \epsilon$' implica que $\epsilon \in$ H e ϵ' $\in$ H.

Definição 11: (ϵ' $\in$ H) $\Rightarrow$ Condição$_H$, onde $\Rightarrow$ denota implicação, especifica que o evento ϵ pode pertencer ao histórico **H** somente se a **Condição**$_H$ é

satisfeita. Em outras palavras, a **Condição**$_H$ é necessária para ϵ estar em **H**. A **Condição**$_H$ é um predicado que envolve os eventos em **H**.
Considere (ϵ' $\in$ H) $\Rightarrow$ ($\epsilon \rightarrow \epsilon$'). Isso indica que o evento ϵ' pode pertencer ao histórico **H** somente se o evento ϵ ocorrer antes do evento ϵ'.

Definição 12: Condição$_H$ $\Rightarrow$ ($\epsilon \rightarrow \epsilon$') especifica que se a **Condição**$_H$ se mantém, ϵ estará no histórico **H**. Em outras palavras, a **Condição**$_H$ é suficiente para que ϵ esteja em **H**.
Considere ($\epsilon \rightarrow \epsilon$') $\Rightarrow$ ($\alpha \in H$); isso significa que, se o evento ϵ ocorre antes do evento ϵ', então o evento α pertence ao histórico **H**.

Definição 13: Um *conjunto de dependências*, denotado por **ConjDep**, é um conjunto de dependências entre transações, desenvolvido durante a execução concorrente de um conjunto de transações **T**. Assim, **ConjDep** está relacionado a um histórico **H**. **ConjDep**$_{atual}$ é usado para denotar o conjunto de dependências até um ponto no tempo.

Definição 14: Cada transação T possui um conjunto de visões sobre o banco de dados, denotado por ViewSet, que é o conjunto dos objetos potencialmente visíveis para a transação T em um determinado ponto no tempo.

B.2
Tipos de Dependências entre as Transações

Ao longo de sua execução, transações solicitam primitivas à gerência de transações, tais como *Begin, Commit e Abort*. Esses são considerados os *eventos significativos* de uma transação, os quais são definidos pela semântica da transação aderente ao modelo e são denotados por ϵ_t.

Transações também solicitam operações sobre objetos do banco de dados, chamados de *eventos de objetos*. Essas operações são denotadas por $op_t[obj]$, que corresponde à solicitação de uma operação *op* sobre um objeto *obj* pela transação *t*.

A execução concorrente de um *conjunto de transações T* está representada em seu histórico *H*, no qual estão registrados todos os *eventos significativos* e os *eventos de objetos* das transações. Em *H* estará registrada a ordem (parcial) em que esses eventos ocorrem. Essa ordem parcial é consistente com a ordem dos eventos de cada transação individual *t* no conjunto *T*. O predicado $\epsilon \rightarrow \epsilon'$ é verdadeiro se ϵ precede o evento ϵ' no histórico *H*. Esse predicado

é falso quando não existe a precedência. Assim, $\epsilon \rightarrow \epsilon'$ implica que $\epsilon \in H$ e $\epsilon' \in H$.

Cada transação *t*, em execução, está associada com a Visão ($View_t$), que é um subhistórico visível a *t*, em algum momento do tempo. Essa *Visão* determina os objetos e seus estados visíveis pela transação *t*. A visão é a projeção da história, em que os eventos projetados satisfazem algum predicado, tipicamente no histórico atual H_{atual}. Então, a ordenação parcial dos eventos na visão é preservada.

A ocorrência de um evento pode ser restritiva em função de:

1. Um evento ϵ *só pode ocorrer* após um outro evento ϵ';
2. Um evento ϵ *só pode ocorrer* se uma condição *c* for verdadeira;
3. A condição *c* pode requerer a ocorrência de um evento ϵ.

As exigências de exatidão impostas às transações que executam concorrentemente em um banco de dados podem ser expressas em termos das propriedades dos históricos de resultados[2]. Tendo em vista que o modelo de transações proposto neste trabalho se baseia, principalmente, na dependência entre as transações, é apresentada a seguir uma revisão da formalização dos tipos de dependência extraída de [13].

Sejam t_i e t_j duas transações e *H* um histórico finito que contém todos os eventos pertencentes a t_i e t_j. Assim, podem-se definir os seguintes tipos de dependências:

- **Dependência de confirmação** (t_j *CD* t_i, onde CD significa *Commit Dependency*) - se ambas as transações t_i e t_j são confirmadas (*commit*), então a confirmação de t_i precede a confirmação de t_j.

$$(Commit_{t_j} \in H) \Rightarrow ((Commit_{t_i} \in H) \Rightarrow (Commit_{t_i} \rightarrow Commit_{t_j}))$$

- **Dependência forte de confirmação** (t_j *SCD* t_i, onde SCD significa *Strong-Commit Dependency*) - se t_i é confirmada (*commit*), então a transação t_j também é confirmada (*commit*).

$$(Commit_{t_i} \in H) \Rightarrow (Commit_{t_j} \in H)$$

[2] O histórico dos resultados armazena os eventos significativos e as operações realizadas pelas transações.

- **Dependência de cancelamento** (t_j *AD* t_i, onde AD significa *Abort Dependency*) - se t_i é cancelada (*abort*), então t_j também é cancelada (*abort*).

  $(Abort_{t_i} \in H) \Rightarrow (Abort_{t_j} \in H)$

- **Dependência fraca de cancelamento** (t_j *WD* t_i, onde WD significa *Weak-abort Dependency*) - se t_i é cancelada (*abort*) e t_j ainda não foi confirmada (*commit*), então t_j é cancelada (*abort*). Em outras palavras, se t_j é confirmada e t_i é abortada, então a confirmação de t_j precede o cancelamento de t_i no histórico *H*.

  $(Abort_{t_i} \in H) \Rightarrow (\neg(Commit_{t_j} \rightarrow Abort_{t_i}) \Rightarrow (Abort_{t_j} \in H))$

- **Dependência de término** (t_j *TD* t_i, onde TD significa *Termination Dependency*) - t_j não pode ser confirmada (*commit*) ou cancelada (*abort*) até que t_i também seja confirmada ou cancelada.

  $(\epsilon' \in H) \Rightarrow (\epsilon \rightarrow \epsilon')$ onde $\epsilon \in \{Commit_{t_i}, Abort_{t_i}\}$ e $\epsilon' \in \{Commit_{t_j}, Abort_{t_j}\}$

- **Dependência de exclusão** (t_j *ED* t_i, onde ED significa *Exclusion Dependency*) - se t_i é confirmada e t_j estiver em execução, então t_j é cancelada (ambas as transações t_i e t_j não podem ser confirmadas).

  $(Commit_{t_i} \in H) \Rightarrow ((Begin_{t_i} \in H) \Rightarrow (Abort_{t_j} \in H))$

- **Dependência de confirmação forçada no cancelamento** (t_j *CMD* t_i, onde CMD significa *force-CoMmit-on-abort Dependency*) - se t_i é cancelada, então t_j é confirmada.

  $(Abort_{t_i} \in H) \Rightarrow (Commit_{t_j} \in H)$

- **Dependência de início** (t_j *BD* t_i, onde BD significa *Begin Dependency*) - a transação t_j não pode iniciar a sua execução até que a transação t_i

tenha iniciado a sua execução.

$$(Begin_{t_j} \in H) \Rightarrow (Begin_{t_i} \rightarrow Begin_{t_j})$$

- **Dependência serial** (t_j *SD* t_i, onde SD significa *Serial Dependency*) - a transação t_j não pode iniciar a sua execução até que a transação t_i tenha sido confirmada ou cancelada.

$$(Begin_{t_j} \in H) \Rightarrow (\epsilon \rightarrow Begin_{t_j}) \text{ onde } \epsilon \in \{Commit_{t_i}, Abort_{t_i}\}$$

- **Dependência de início na confirmação** (t_j *BCD* t_i, onde BCD significa *Begin-on-Commit Dependency*) - a transação t_j não pode iniciar a sua execução até que a transação t_i seja confirmada.

$$(Begin_{t_j} \in H) \Rightarrow (Commit_{t_i} \rightarrow Begin_{t_j})$$

- **Dependência fraca de início na confirmação** (t_j *WCD* t_i, onde WCD significa *Weak-begin-on-Commit Dependency*) - se t_i é confirmada, t_j pode iniciar a sua execução após a confirmação de t_i.

$$(Begin_{t_j} \in H) \Rightarrow ((Commit_{t_i} \in H) \Rightarrow (Commit_{t_i} \rightarrow Begin_{t_j}))$$

- **Dependência de início no cancelamento** (t_j *BAD* t_i, onde BAD significa *Begin-on-Abort Dependency*) - a transação t_j não pode iniciar a sua execução até que a transação t_i seja cancelada.

$$(Begin_{t_j} \in H) \Rightarrow (Abort_{t_i} \rightarrow Begin_{t_j})$$

C
Diagrama do Modelo Lógico de Dados Relacional do Exemplo de Vendas e Interfaces da Aplicação Móvel

Este Anexo tem por objetivo apresentar o diagrama lógico relacional do modelo de dados utilizado como exemplo neste trabalho, bem como as interfaces de uma aplicação móvel que consulta e atualiza os dados desse modelo.

C.1
Diagrama Lógico Relacional do Exemplo de Vendas

A figura C.1 apresenta o *Modelo Lógico Relacional do Banco de Dados* utilizado no estudo de caso. Esse modelo possui as seguintes representações:

- As tabelas estão representadas nos retângulos com seus respectivos atributos;
- Os atributos que participam da chave primária de cada tabela estão sublinhados e assinalados com pk;
- Os atributos que participam das chaves estrangeiras em cada tabela estão assinalados com fk;
- O formato de cada atributo está indicado ao lado do mesmo.

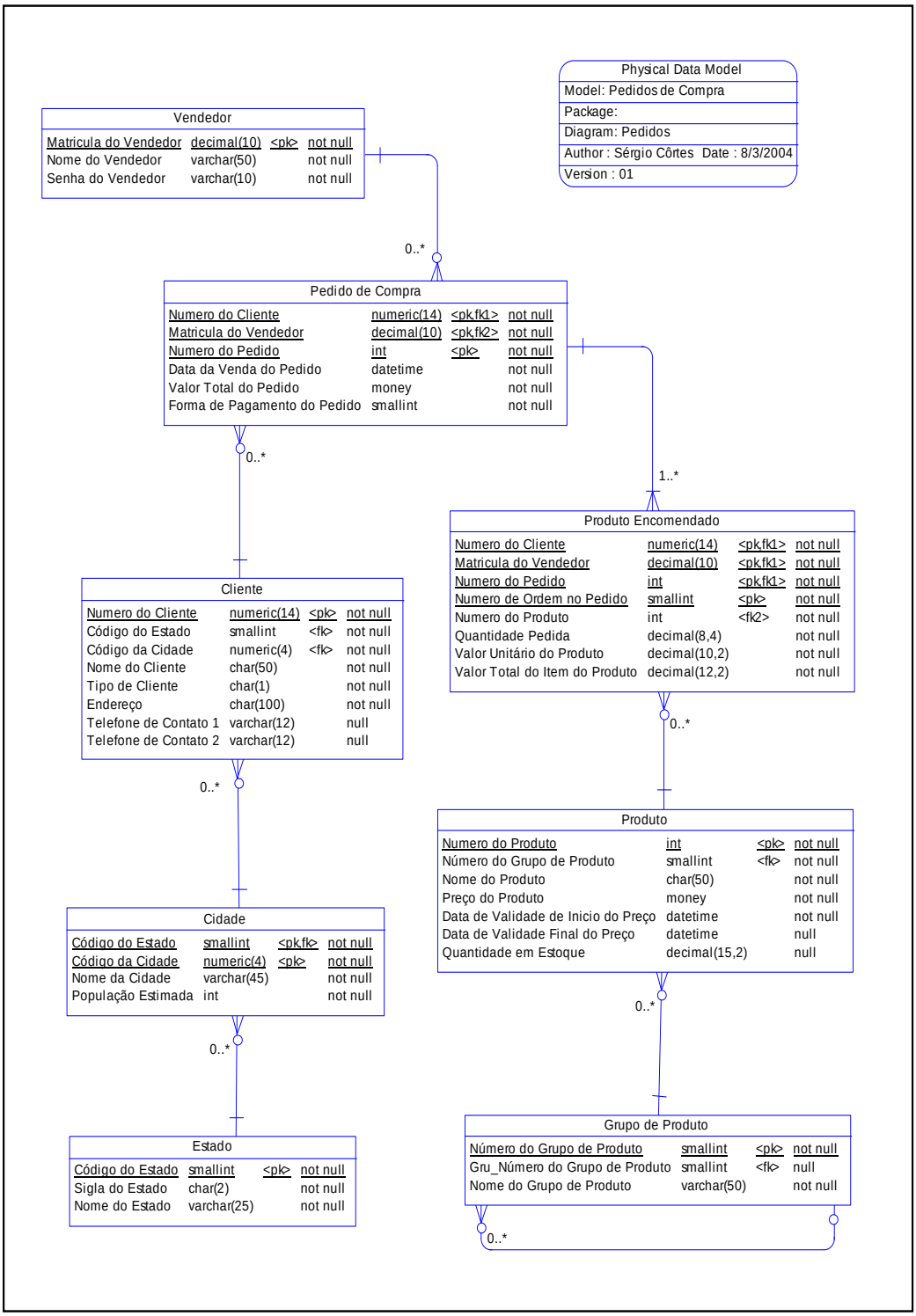

Figura C.1: Modelo Lógico Relacional do Banco de Dados utilizado no estudo de caso

C.2
Interfaces da Aplicação Móvel

A seguir são apresentadas as interfaces de uma aplicação móvel em que um vendedor, atendendo à solicitação de um cliente, faz um pedido de compras com cinco produtos. Inicialmente o vendedor se identifica na aplicação (figuras C.2 e C.3), depois identifica o cliente e a forma com que o cliente deseja efetuar o pagamento pela compra (figuras C.4 e C.5), a seguir seleciona os produtos que o cliente deseja comprar, informando as suas quantidades (figuras C.6 e C.7) e, finalmente conclui a venda ao cliente, após a *aplicação móvel* calcular o valor total da compra (figura C.8).

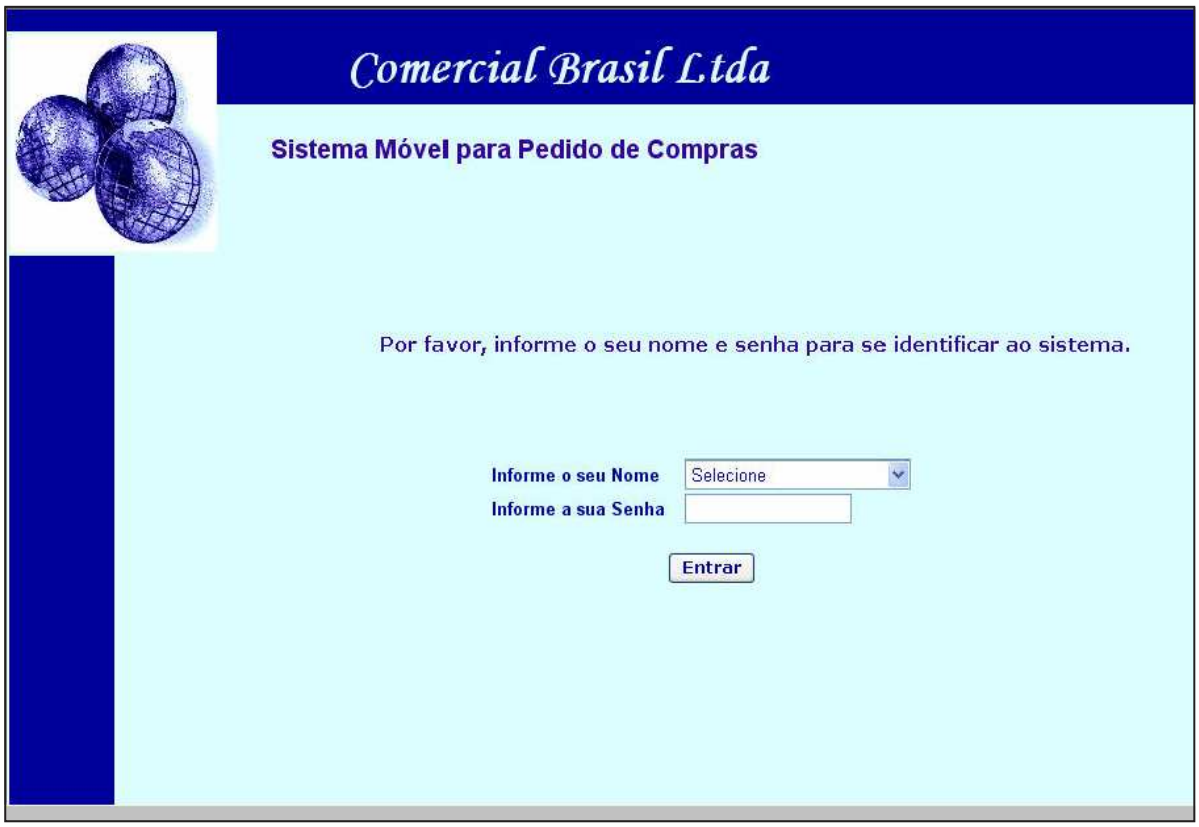

Figura C.2: Interface 01 - início da aplicação

Figura C.3: Interface 02 - identificação do vendedor

Figura C.4: Interface 03 - início do pedido de compra

Figura C.5: Interface 04 - identificação do cliente e da forma de pagamento da compra

Figura C.6: Interface 05 - seleção dos produtos para venda e suas quantidades

Figura C.7: Interface 06 - seleção dos produtos vendidos e suas quantidades

Figura C.8: Interface 07 - conclusão da venda ao cliente