Um Método Baseado em Redes de Petri para Modelagem de Bancos de Dados em Tempo-Real

Maria Lígia Barbosa Perkusich

Tese de Doutorado submetida à Coordenação dos Cursos de Pós-Gradução em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Maria de Fátima Queiroz Vieira Turnell, Ph.D.
Orientadora

Campina Grande, Paraíba, Brasil

P447m Perkusich, Maria Ligia Barbosa
Um metodo baseado em redes de petri para modelagem de bancos de dados em tempo-real / Maria Ligia Barbosa Perkusich. - Campina Grande, 2000.
178 f. : il.

Tese (Doutorado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Petri 2. Banco de Dados 3. Tempo-Real 4. Tese I. Turnell, Maria de Fatima Queiroz Vieira II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 681.3.02(043)

UM MÉTODO BASEADO EM REDES DE PETRI PARA BANCO DE DADOS EM TEMPO REAL

MARIA LÍGIA BARBOSA PERKUSICH

Tese Aprovada em 24.03.2000

Maria de Fatima Queiroz Vieira Turnell

MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFPB
Orientadora

ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFPB
Componente da Banca

ULRICH SCHIEL, Dr.rer.nat., UFPB
Componente da Banca

ALBERTO HENRIQUE FRADE LAENDER, Ph.D., UFMG
Componente da Comissão

JOSÉ REINALDO SILVA, Ph.D., USP
Componente da Comissão

CAMPINA GRANDE - PB
Março - 2000

Dedicatória

Dedico esta tese ao meu esposo Angelo, aos meus filhos Mirko e Andrei e aos meus pais Zefinha e Leocádio (in memorian).

Agradecimentos

A Deus que sempre me fortaleceu e ajudou.

Aos meus queridos filhos Mirko e Andrei pelo amor, incentivo, apoio e compreensão.

Ao meu querido esposo Angelo pelo amor, ajuda, confiança e compreensão, sem os quais seria impossível a realização desta tese.

A Professora Maria de Fátima Turnell pelo incentivo e orientação desta tese.

Aos Membros da Banca Examinadora pela atenção e ricas sugestões.

A minha mãe pela compreensão e amor.

Aos meus irmãos Liginey, Laudilene e Luciano pelo apoio.

A minha prima Païzinha pela ajuda, especialmente no início deste trabalho.

A Ângela da COPELE pela amizade e atenção.

Ao Pedro da COPELE pelo apoio constante e atenção.

Ao Professor Antônio Marcos Nogueira, coordenador da COPELE, pelo apoio incondicional demonstrado.

A CAPES pelo apoio financeiro, através de Bolsa de Estudo.

Aos meus amigos Tomaz e Jussara pelo incentivo e amizade.

Aos meus amigos do LabPetri, Adriano, Ana Carla, Dalton, Érica, Jorge, Killer e Sandro que muito colaboraram para a realização desta tese.

A Claudia da cantina pela amizade.

A todos que direta ou indiretamente colaboraram na execução desta tese.

Resumo

Os métodos disponíveis para a descrição do modelo conceitual dos sistemas de bancos de dados convencionais não são apropriados para a descrição do modelo conceitual dos sistemas de bancos de dados em tempo-real, já que os mesmos não oferecem mecanismos para a representação das restrições temporais presentes nestes sistemas. Também, a maioria dos modelos existentes concentra-se na representação das propriedades estáticas dos dados. No entanto, sistemas complexos, tais como bancos de dados em tempo-real também requerem a modelagem das propriedades dinâmicas dos dados. Então, faz-se necessário o desenvolvimento de métodos para o projeto do modelo conceitual dos sistemas de bancos de dados em tempo-real que permitam a modelagem dos aspectos estáticos e dinâmicos de tais sistemas.

O principal objetivo desta tese é definir um método com base formal para a modelagem de sistemas de bancos de dados em tempo-real, no contexto de sistemas flexíveis de manufatura e automação industrial.

A aplicação do método resulta em dois modelos: um *modelo de objetos*, que é usado para modelar os aspectos estáticos de um sistema, e um *modelo de processos*, que é usado para modelar os aspectos dinâmicos de um sistema. O modelo de objetos é baseado no modelo de objetos do OMT (*Object Modelling Technique*) e no modelo RTSORAC (*Real-Time Semantic Objects Relationships and Constraints*). O modelo de processos é baseado numa classe de redes de Petri de alto-nível denominada ECPN (*Extended Coloured Petri Nets*), desenvolvida nesta tese.

Os modelos descritos em ECPN precisam ser analisados. Assim, mostramos como as ECPN podem ser transformadas em redes de Petri coloridas hierárquicas (HCPN). Desta forma, todos os procedimentos de análise de HCPN podem ser aplicados as ECPN.

Abstract

The methods available to describe the conceptual model of conventional database systems cannot be directly applied to describe the model of real-time database systems, since these models do not provide mechanisms to represent the temporal restrictions that exist in such systems. Also, most of the available models focus on the representation of static properties of the data. On the other hand, complex systems, such as real-time databases, require that dynamic properties for data and information be modeled as well. Therefore, it is necessary the development of methods for the design of real-time databases supporting the modeling of both static and dynamic aspects of systems.

The main objective in this work is to define a method with formal basis for the modeling of real-time database systems, in the context of flexible manufacturing systems and industrial automation in general.

The application of such a method results in two models: *an object model*, that is used to model the static aspects of a system, and a *process model*, used to model the dynamic aspects of a system. The object model is based on the object model of the OMT (*Object Methodology Technique*), and on the real-time object oriented data model RTSORAC, (*Real Time Semantic Objects Relationships and Constraints*), whereas the process model is based on a high level Petri net named ECPN (*Extended Coloured Petri Nets*), developed in this work.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Atualmente podemos encontrar várias aplicações que envolvem o processamento de grande volume de dados e acesso com restrições temporais a esses dados. Entre essas aplicações estão os sistemas de telefonia, sistemas de gerenciamento de redes de computadores e controle de fábricas automatizadas. Geralmente esses sistemas precisam interagir com o ambiente continuamente para obter os dados, que podem ter um tempo de vida útil limitado, e utilizá-los no processamento de atividades que devem produzir respostas em tempo-real. Outra característica desses sistemas é a necessidade do compartilhamento de dados entre várias aplicações.

Sistemas de gerenciamento de bancos de dados (SGBD) tradicionais disponibilizam mecanismos para implementação de algumas das funções requeridas por esses sistemas, tais como, coordenação de transações concorrentes e acesso consistente a dados compartilhados. Entretanto, os SGBD tradicionais não disponibilizam mecanismos que possibilitem tratar os requisitos de tempo inerentes aos sistemas acima mencionados [Ram93; SSH99]. Por outro lado, os sistemas de gerenciamento de bancos de dados para aplicações em tempo-real, ou simplesmente, sistemas de gerenciamento de bancos de dados em tempo-real (SGBD-TR) apresentam as características necessárias para expressar e manter dados e transações com restrições temporais [OS95; Ram93]. Estas características adicionais presentes nos SGBD-TR introduzem a necessidade de desenvolver novas técnicas para modelagem conceitual, controle de concorrência, escalonamento, entre outras [DiP95a; DiP95b; DW97; SSH99].

As seções seguintes têm o objetivo de motivar o leitor com relação ao problema de

pesquisa na área de bancos de dados para aplicações em tempo-real (BDTR) abordada nesta tese. Além disso, apresentam-se sucintamente os conceitos e ferramentas utilizadas para abordarem o problema da modelagem conceitual de BDTR.

1.1 Motivação

Nos últimos anos as pesquisas na área de BDTR têm se intensificado, tanto por parte dos pesquisadores da área de bancos de dados, quanto por parte dos pesquisadores da área de sistemas em tempo-real (STR) [OS95; Ram93; SSH99].

O objetivo dos pesquisadores da área de bancos de dados é compartilhar os benefícios da tecnologia de bancos de dados para solucionar os problemas de gerenciamento de dados em STR. Por outro lado, os pesquisadores de STR são atraídos pela oportunidade que os SGBD-TR fornecem para aplicar escalonamento guiado pelo tempo e algoritmos de alocação de recursos. Portanto, considerando os mecanismos da tecnologia de bancos de dados para tratar com aplicações que envolvem grande volume de dados, e da necessidade dos STR em lidar com muitos dados em várias aplicações, seria benéfico se essas duas tecnologias fossem integradas. De maneira similar, os avanços obtidos pela tecnologia dos STR para processar atividades com restrições temporais poderiam ser explorados para tratar com as transações com restrições temporais nos BDTR. Entretanto, como discutido por Graham [Gra92], Özsoyoglu e Snodgrass [OS95], Ramamritham [Ram93], e Stankovic [SSH99] a simples integração destas tecnologias (conceitos, mecanismos, ferramentas) não é suficiente para desenvolver um SGBD-TR. Enquanto muitas das técnicas usadas em STR de um lado, e SGBD de outro, podem ser aplicadas aos SGBD-TR, muitas diferenças existem, que requerem adaptações das abordagens usadas nas duas áreas, ou mesmo, o desenvolvimento de novas abordagens. Questões como modelagem conceitual, controle de concorrência, escalonamento, recuperação, entre outras, estão sendo consideradas e pesquisadas para SGBD-TR [BLS97; Jon98; KS96; OS95; SSH99]. Nossa pesquisa concentra-se no problema de modelagem conceitual de BDTR.

Atualmente as pesquisas na área de modelagem conceitual de sistemas complexos têm se concentrado principalmente na abordagem orientada a objetos. Essa abordagem absorve e permite a aplicação controlada de conceitos de abstração, encapsulamento, modularidade, entre outros. Mecanismos de abstração são importantes pois permitem

que o projetista concentre-se em aspectos essenciais de uma aplicação enquanto ignora detalhes. Encapsulamento proporciona a separação da especificação externa da implementação interna, além de permitir que características, tais como restrições lógicas e temporais, métodos concorrentes, entre outras, para um objeto, sejam declaradas em um único lugar. Modularidade promove coerência, compreensão e simetria para organizar um sistema em grupos de objetos relacionados [BP98].

Vários métodos para modelagem conceitual utilizando a abordagem orientada a objetos foram propostos [Boo94; Dou98; RJB99; Rum91; SWG94]. Recentemente, a linguagem UML foi adotada pela OMG (*Object Management Group*) como uma notação padrão para modelagem orientada a objetos. A linguagem UML define um conjunto de diagramas que são usados para descrever diferentes aspectos ou visões de um sistema. Por exemplo, os diagramas de classes descrevem a estrutura estática de um sistema como um conjunto de classes que se relacionam entre si, enquanto que os diagramas de estado (*statecharts*) [Har87] descrevem o comportamento de uma classe como mudanças em seu estado interno resultante do recebimento de eventos externos. Os *statecharts* utilizados na UML usam uma notação similar aos *statecharts* de Harel, mas a semântica dos diagramas é consideravelmente diferente. Na UML, a semântica dos *statecharts* de Harel foi modificada para a semântica *run-to-completion*, mostrada na Seção 3.3.2, que é mais simples que a original e assegura que estímulos externos sejam recusados até que o processamento interno seja completado, mas por outro lado exclui a concorrência intra-objetos [GGW99a; GGW99b]. Este fato torna os *statecharts* da UML inapropriados para muitas aplicações, incluindo vários tipos de STR, onde a sobrecarga de execução seria inaceitável.

1.2 Modelo Conceitual

O termo *modelo conceitual* vem da área de bancos de dados, tendo sido estabelecido na década de 70, por um grupo de trabalho da associação norte-americana de normas (ANSI) [TK78]. A característica básica de um ambiente de bancos de dados é o compartilhamento de dados entre diversas classes de usuários com requisitos funcionais diferentes. Portanto, o banco de dados é o meio de comunicação entre os diversos tipos de usuários. No entanto, para que esta comunicação seja confiável e eficaz, é necessário

que todos os usuários tenham o mesmo entendimento do significado dos dados representados no banco de dados. Para suprir esse entendimento, inicialmente, o modelo conceitual foi definido como sendo o elemento central de toda descrição do significado dos dados representados no banco de dados.

A função do modelo conceitual foi sendo ampliada gradativamente, de documentação central de um banco de dados, para o modelo que serve de ponto de partida para a implementação do sistema de bancos de dados. Inicialmente, a maior parte das atividades de pesquisa relacionadas ao modelo conceitual se concentrou nas características estáticas do banco de dados, ou seja, sobre a descrição dos estados corretos do banco de dados. Esses modelos descrevem apenas os dados e seus relacionamentos e são denominados *modelos orientados a dados*, como por exemplo o Modelo Entidade/Relacionamento [EN94]. Mais tarde, com a utilização do modelo conceitual também no projeto das aplicações sobre o banco de dados, surgiu a necessidade de incluir no modelo conceitual também as características funcionais, ou seja, como os valores de saída são derivados dos valores de entrada, vistos como fluxo de dados entre as operações. Esses modelos descrevem *"o que"* cada função faz e são denominados *modelos orientados a comportamento/função*, como por exemplo os Diagramas de Fluxo de Dados - DFD [EN94]. Hoje, devido à complexidade inerente a muitos sistemas, o modelo conceitual pode servir também para descrever as propriedades dinâmicas das transações que envolvem o banco de dados, de forma que possa dar suporte à fase de implementação do sistema, facilitando e agilizando as etapas de análise dos requisitos e implementação do sistema. Esses modelos descrevem as mudanças nas entidades e nos seus relacionamentos dependentes do tempo. Eles descrevem *"como"* as funções devem ser realizadas pelo sistema e são denominados *modelos dinâmicos ou orientados a processos*, como, por exemplo, as redes de Petri [Mur89] e os *statecharts* [Har87]. Assim, o modelo conceitual de um sistema é estabelecido por diferentes paradigmas/visões: modelagem dos dados, modelagem do comportamento (funcional) e modelagem dos processos (dinâmica).

Um modelo conceitual é, portanto uma representação gráfica/matemática das principais propriedades estáticas, funcionais e dinâmicas de um sistema que pode ser utilizado diretamente na sua implementação.

Uma linguagem de modelagem conceitual deve permitir que todas as propriedades desejáveis para o sistema sejam capturadas pela descrição do modelo, sem que proprie-

dades indesejáveis sejam incluídas. Além disso, a linguagem de modelagem deve fornecer modelos facilmente compreensíveis pelos usuários e projetistas, e deve fornecer modelos exatos e não ambíguos, o que implica que a linguagem deve ter algum tipo de formalismo matemático que permita a análise e verificação das propriedades modeladas. Portanto, o desenvolvedor de uma linguagem de modelagem deve confrontar com uma negociação entre expressividade e entendimento [BP98]. Uma linguagem simples é coerente, mas pode não capturar todas as propriedades do sistema, enquanto que uma linguagem complexa, com base formal, pode permitir uma especificação mais precisa, mas pode inibir o amplo entendimento do problema.

Através da análise do modelo conceitual, um sistema real pode ser estudado sem o perigo, custo ou inconveniência da manipulação de seus elementos. Quando se tratando de sistemas complexos, o processo de análise do modelo precisa ser automatizado. Portanto é fundamental a utilização de modelos matemáticos, os quais possibilitam submeter os modelos a procedimentos automáticos de análise e verificação. Tais procedimentos permitem detectar uma série de deficiências que podem estar presentes no modelo conceitual, tais como contradições, ambigüidades, redundância, incompletude, entre outras. No contexto desta tese, a base formal utilizada são as redes de Petri [Mur89].

1.3 Redes de Petri

Redes de Petri são uma ferramenta gráfica e matemática que permitem a modelagem de diversos tipos de sistemas [Mur89]. O uso do formalismo de redes de Petri para a modelagem conceitual, análise, simulação e controle de sistemas complexos vem sendo defendido e utilizado amplamente na academia e na industria por um grande número de desenvolvedores e pesquisadores citejensen92. Elas apresentam as seguintes características:

- São uma ferramenta com capacidade para modelagem hierárquica, com fundamentação matemática bem desenvolvida, que pode ser usada para análise.
- Permitem que diferentes propriedades de sistemas concorrentes, tais como conflito, concorrência, bloqueio perpétuo, sincronismo, e exclusão mútua, entre outras, sejam verificadas;

- Têm uma representação gráfica que pode ser usada como documentação e que facilita a interação entre desenvolvedores e/ou usuários do sistema sem que necessariamente precisem defrontar-se com a especificação formal subjacente;
- Podem ser simuladas, de modo que não só a estrutura do sistema, mas também o comportamento dinâmico da especificação pode ser observado pelos projetistas e usuários do sistema com base nas sessões de simulação;

Devido às características acima mencionadas, a modelagem de sistemas através de redes de Petri suporta o desenvolvimento no nível conceitual, permitindo manter a coerência com elementos e termos próprios do domínio do problema nos níveis mais abstratos. Sua faceta gráfica é extremamente importante quando a questão é prover mecanismos para a construção de sistemas reais e não apenas um modelo teórico com o qual se deseja apenas raciocinar matematicamente sobre sistemas-exemplo.

Sendo as redes de Petri uma linguagem com formalismo matemático, modelos descritos com tal linguagem podem ser submetidos a procedimentos automáticos de análise e verificação, através dos quais podem ser detectadas inconsistências, erros ou falta de suporte aos requisitos estabelecidos na descrição do sistema.

Para uma introdução detalhada a esses e outros aspectos relacionados a redes de Petri o leitor pode referir-se à [Mur89; Pet81]. No Capítulo 5 apresenta-se uma revisão dos conceitos básicos relacionados a redes de Petri, entretanto esta introdução esta orientada a fornecer um embasamento teórico para os temas que serão abordados neste trabalho.

1.4 Manufatura Integrada por Computador (MIC)

Ultimamente tem-se notado o uso intensivo de computadores e sistemas inteligentes em diferentes funções em empresas de manufatura. No entanto, devido à evolução *bottom-up* da automação industrial, geralmente os diferentes subsistemas automatizados são unidades isoladas [DAJ94; Zho97].

MIC é uma filosofia de manufatura que visa o uso de computadores em todas as fases de produção de uma empresa de manufatura, desde o projeto do produto, planejamento da produção e controle das operações, até atividades como marketing, pedidos dos consumidores, etc. Geralmente MIC é definida como a automação e integração de várias

funções e entidades de uma empresa de manufatura através do uso de computadores e recursos existentes [DAJ94; Zho97].

O conceito de MIC visa preencher a lacuna entre os vários subsistemas de uma empresa através do fluxo de informação coordenada entre eles. Este conceito indica o uso de computadores para executar as diversas funções necessárias dentro de uma empresa de manufatura, e conseqüentemente o uso de um banco de dados que será compartilhado entre os diversos subsistemas, possibilitando assim a integração total da empresa.

1.5 Sistemas Flexíveis de Manufatura

SGBD-TR podem ser aplicados a diferentes tipos de sistemas pertencentes ao domínio de automação industrial e, em particular, a sistemas de manufatura. Estas aplicações envolvem a aquisição de dados do ambiente e o subseqüente processamento desses no contexto de informações passadas. A dependência do processamento de informações correntes e passadas, associada à crescente complexidade dos sistemas de manufatura e à inerente quantidade de dados, exige a utilização de ferramentas que possibilitem garantir a consistência temporal das respostas produzidas. No contexto desta tese considera-se os Sistemas Flexíveis de Manufatura (SFM) como o domínio de aplicação.

Um SFM é um sistema complexo, baseado em tecnologia de grupo[1], para manufaturar uma variedade de produtos.

Um SFM é estruturado como um conjunto de células, onde cada célula possui um conjunto de recursos de produção (máquinas, ferramentas, etc.), um conjunto de recursos de transporte (robôs, esteiras, etc.), e um conjunto de depósitos de material (matéria-prima, subprodutos, produtos finalizados). No Capítulo 7 apresentamos um exemplo para uma célula de manufatura.

No contexto deste trabalho considera-se a estrutura de controle hierárquico para uma empresa de manufatura dividida em cinco níveis de abstração, compartilhando um banco de dados. Como mostrado na Figura 1.1, tem-se no lado esquerdo os diferentes níveis de abstração e no lado direito o banco de dados. Observa-se que a operação correta do

[1] Tecnologia de grupo é uma filosofia de manufatura onde peças similares são identificadas e agrupadas, para obter-se vantagens de suas similaridades nas fases de projeto e manufatura destas peças.

sistema de controle global depende da interação entre esses diferentes níveis [PPC96; PPS95a; PPS95b].

O *nível de planejamento* é o nível mais alto na hierarquia. Ele coordena as diferentes atividades no sistema de manufatura e gerencia as atividades de planejamento em longo prazo.

O *nível de escalonamento* implementa mecanismos de decisão a fim de definir a ordem de produção para cumprir as metas de planejamento.

O *nível de supervisão*, executa alocação de máquinas e outros dispositivos do processo de manufatura de forma a atender as decisões do nível de escalonamento.

A principal função executada pelo *nível de coordenação* é coordenar os diferentes recursos, de forma que a execução de um conjunto de operações seja correta. Essas operações podem ser executadas concorrentemente.

No *nível de comando local* a principal função é o controle direto dos dispositivos associados ao processo controlado, tais como, robôs, máquinas, etc.

O banco de dados é o componente central do sistema, no sentido de que ele fornece suporte a todos os níveis da hierarquia. Os dados armazenados são gerados nos diversos níveis e contêm informações sobre o estado geral do sistema controlado (células, máquinas, transportes, peças, escalonamentos, detalhes de configuração, fornecedores, etc.). Cada nível usa o banco de dados para comunicar-se com os demais. Os dados gerados em um nível podem disparar eventos em outros níveis. Os dados têm diferentes granularidades de tempo, por exemplo, no nível mais baixo (comando local) o tempo é usualmente expresso em milissegundos, enquanto que nos níveis mais altos (supervisão e planejamento) o tempo pode ser expresso em minutos, horas, meses e até em anos. Isso é devido, principalmente, ao fato de que cada nível tem diferentes requisitos temporais para os dados armazenados no banco de dados, como mostrado na Figura 1.1.

Considerando os atuais requisitos de um sistema de manufatura, compreendendo tanto o sistema de informação como o sistema de controle, a necessidade de integração, via um banco de dados, entre os diferentes níveis de abstração é cada vez mais vital para o gerenciamento otimizado dos recursos de produção. A disponibilidade de dados, tanto de planejamento e engenharia de produção, quanto de controle de processos, é condição necessária para satisfazer os requisitos de produção.

Dessa forma, é desejável que a modelagem conceitual do sistema de controle de

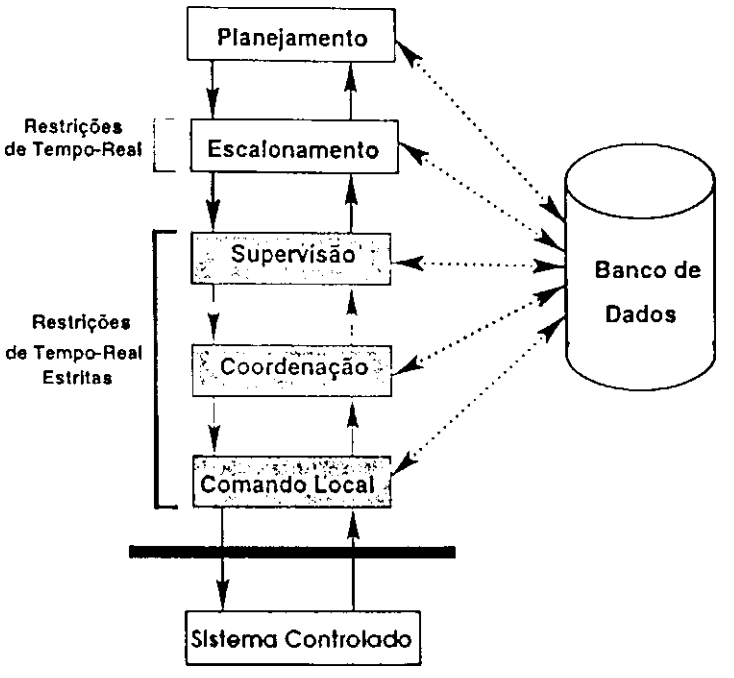

Figura 1.1: Níveis em um sistema de controle hierárquico

processos, incluindo os diferentes níveis de abstração, de forma integrada ao banco de dados, seja considerada com base em um método unificado. Dessa forma, evita-se a necessidade de desenvolver interfaces com o objetivo de adequar e integrar os diferentes modelos com base em métodos diferentes.

Assim, é de fundamental importância para satisfazer o requisito de integração, e o necessário rigor de ferramentas formais no desenvolvimento de sistemas complexos, tais como sistemas de controle em automação industrial, a busca de um método unificado com base formal que seja aplicável tanto no contexto de banco de dados quanto no sistema de automação industrial.

1.6 Objetivos do Trabalho

Neste trabalho introduzimos um método de modelagem com forte base formal para BDTR. Este método é baseado na técnica OMT (*Object Modelling Tecnique*) [Rum91; BP98], no modelo de dados RTSORAC (*Real-Time Semantic Objetcs Relationships and Constraints*) [DiP95b; DW97; PWPD96], e numa nova classe de redes de Petri de alto-nível denominada ECPN (*Extended Coloured Petri Nets*).

Além da definição de um conjunto sistemático de procedimentos para obter o mo-

delo conceitual de um BDTR, incorporamos um conjunto de extensões a uma classe de redes de Petri baseadas em objetos, denominada G-CPN [Gue97; GPdF97; GdFP97; GdFP00] resultando nas redes ECPN. As principais extensões que incorporamos a G-CPN incluem:

- Introdução e formalização de restrições lógicas e temporais.
- Introdução e formalização do tratamento da execução concorrente dos métodos de um objeto.

A linguagem formal para descrição de sistemas ECPN disponibiliza ao projetista de um BDTR mecanismos para tornar a descrição mais compacta e mecanismos de abstração que permitem manter o foco do projeto em certas partes do modelo.

Ainda, um modelo descrito em ECPN necessita ser analisado. Para tanto, introduzimos um procedimento e um conjunto de algoritmos que possibilitam transformar um modelo ECPN em uma rede de Petri Colorida Hierárquica (HCPN) [Jen92]. Este modelo HCPN é então analisado utilizando cenários e diagramas de seqüências de mensagens, e através da geração e análise do espaço de estados para os cenários. O pacote de ferramentas Design/CPN [Ja96; Ja99] é utilizado em todo o processo de construção e análise do modelo.

O domínio de aplicação exemplificado nesta tese é constituído pelos sistemas de controle em tempo-real hierárquico, tais como SFM. O método introduzido permite a modelagem integrada, tanto dos aspectos puramente de controle de um sistema de automação industrial, como aqueles relativos ao BDTR.

1.7 Organização da Tese

Esta tese está organizada em oito capítulos, além deste. No Capítulo 2 introduzimos os conceitos básicos de SGBD-TR. No Capítulo 3 introduzimos os conceitos básicos de orientação a objetos, além de dois dos principais métodos orientados a objetos: UML e OMT. No Capítulo 4 introduzimos os principais conceitos sobre controle de concorrência semântico. No Capítulo 5 introduzimos os conceitos básicos de redes de Petri e ECPN, que são utilizados como base formal no desenvolvimento desta tese. No Capítulo 6 apresentamos a formalização de ECPN. No Capítulo 7 introduzimos um método orientado

a objetos para modelagem de BDTR base em ECPN. No Capítulo 8 apresentamos um procedimento de análise para sistemas ECPN e exemplificamos os resultados de análise do modelo para o exemplo introduzido no Capítulo 7. Por fim, no Capítulo 9 declaramos as contribuições deste trabalho, bem como sumarizamos os trabalhos futuros.

Capítulo 2

Sistemas de Gerência de Bancos de Dados para Aplicações em Tempo-real

Este capítulo tem o objetivo de introduzir os conceitos básicos de SGBD-TR. Para tal, inicialmente serão introduzidos os conceitos básicos de STR e SGBD convencionais, nas Seções 2.1 e 2.2, respectivamente. Na Seção 2.3 são apresentados os conceitos básicos de bancos de dados orientado a objetos. Na Seção 2.4 os SGBD-TR são introduzidos e suas principais características são apresentadas, além de alguns modelos de dados existentes. Também são apresentadas algumas das principais questões a serem pesquisadas para tais sistemas. O objetivo é embasar a identificação das características e propriedades relevantes, para guiar a definição de um método de modelagem com base formal para suporte a modelagem de BDTR.

2.1 Sistemas em Tempo-real

Sistemas em Tempo-real (STR) são sistemas que tratam com *dados temporais*, isto é, dados que se tornam sem valor depois de um certo intervalo de tempo, e *eventos temporais*, que são processos cujo tempo de execução é fundamental para o desempenho do sistema.

Computação em tempo-real geralmente é associada a processamento em alta velocidade. No entanto, computação em tempo-real não significa simplesmente processamento em alta velocidade. Computação em tempo-real não é computação com alto desempe-

nho, é computação com restrições temporais [Sta88]. Isso significa que as tarefas de um STR devem incluir declarações explícitas sobre o tempo no qual o processamento deve ser realizado. Então, devido à natureza dos dados e dos requisitos de tempo de resposta impostos pelo ambiente, para que uma tarefa em um STR seja executada com sucesso, ela deve ser executada dentro do intervalo de tempo determinado pelo sistema.

Um STR consiste de um sistema controlador e um sistema controlado [Ram93]. Por exemplo, em uma fábrica automatizada, o sistema controlador é composto pelo computador e as interfaces humanas, enquanto que o sistema controlado é a fábrica (ambiente), com seus robôs, estações de montagem, peças, e esteiras. O sistema controlador interage com o sistema controlado com base nos dados disponíveis sobre o ambiente, os quais são obtidos de vários sensores e arquivos, por exemplo, sensor de temperatura e arquivo de peças. É extremamente importante que o sistema controlador obtenha informações sobre o estado real do ambiente, de forma que os resultados obtidos sejam os esperados, de outra forma, estes podem ser catastróficos. Também, os dados obtidos podem ser usados para derivar novos dados. Geralmente a história da interação com o ambiente é armazenada em um banco de dados temporal. O principal objetivo das restrições de tempo dos STR, e a exigência de que as mesmas sejam obedecidas, é a necessidade de que os dados requisitados estejam sempre disponíveis para o sistema controlador, de modo que ele possa sempre tomar decisões certas nos momentos certos.

Em STR, a unidade de execução com restrição temporal é chamada de *tarefa*. As expressões com restrições temporais para as tarefas podem ter a forma de *tempo de início, prazo*[1], e *períodos*, entre outras. Os STR exigem que as restrições temporais para as tarefas possam ser declaradas, cumpridas, e suas violações possam ser tratadas adequadamente [DiP95b]. O tratamento das violações das restrições temporais depende dos requisitos das tarefas, ou seja, de acordo com o modo de tempo-real requerido, quais sejam: *estrita, suave* ou *firme*[2]. Esses requisitos serão discutidos mais adiante.

Para que um STR possa cumprir as restrições temporais, o comportamento das tarefas deve ser predizível, isto é, o sistema deve ter conhecimento das ações que serão executadas, bem como dos recursos necessários. A seguir, cada um dos aspectos das restrições temporais será examinado.

[1] *deadline*

[2] *hard, soft* ou *firm*

Declarando Restrições Temporais

A maioria dos sistemas em tempo-real especifica pelo menos um subconjunto das seguintes restrições temporais:

Uma restrição de *tempo de início mais cedo*[3] especifica um tempo absoluto antes do qual a tarefa não deve começar.

Uma restrição de *tempo de início mais tardio*[4] especifica um tempo absoluto antes do qual a tarefa deve começar. Este tipo de restrição é bastante útil para detectar a violação do prazo mesmo antes dele acontecer.

Um *prazo* especifica um tempo absoluto antes do qual a tarefa deve ser completada.

Uma restrição em forma de *período* especifica o tempo de início mais cedo e o prazo, em intervalos regulares, para repetidas instâncias de uma tarefa. Cada instância da tarefa deve ser executada a qualquer momento dentro do intervalo determinado.

Modos de Tempo-real

Restrições de tempo-real podem ser classificadas como *estrita, suave* e *firme*, dependendo das conseqüências da violação da restrição.

Estrita: uma tarefa é *estrita* quando qualquer resultado produzido após seu prazo é inútil para o sistema. Isso significa que qualquer tarefa *estrita* deve ser abortada quando não puder cumprir seu prazo, independentemente de sua conseqüência.

Suave: uma tarefa é *suave* quando o resultado produzido após seu prazo sempre tem algum valor, que vai perdendo sua utilidade à medida que se distancia de seu prazo.

Firme: uma tarefa é *firme* quando a perda do prazo não gera nenhum efeito ou valor para o sistema. Geralmente estas tarefas são recomeçadas quando perdem seu prazo.

[3] *Earliest Start Time*
[4] *Latest Start Time*

Predição

A capacidade de predizer se as restrições temporais das tarefas serão satisfeitas é fundamental em STR [DiP95b]. No entanto, para um STR poder predizer se suas restrições temporais serão satisfeitas, ele precisa ser capaz de fazer uma análise do comportamento temporal de suas ações. Para isso ele precisa saber quais recursos serão usados e o período de tempo que cada tarefa utilizará os recursos. Para garantir esse comportamento temporal, os valores considerados para utilização dos recursos serão os de piores casos, embora sistemas com restrições temporais do tipo suave possam considerar valores médios que não oferecem garantia total.

Determinar o tempo que uma tarefa utilizará um recurso é geralmente muito difícil. Por exemplo, considere o caso de determinar o tempo de CPU que uma tarefa precisa. Para estabelecer o pior caso de utilização da CPU, todas as possíveis ramificações da tarefa devem ser consideradas para o pior caso, ou seja, o pior caminho, e todos os laços e recursões devem ser considerados para ter um número limitado de instâncias. No entanto, utilização de CPU é apenas um dos recursos que uma tarefa pode precisar. Uma tarefa também pode precisar utilizar memória principal, dispositivos de E/S, etc. Além disso, o uso desses recursos pode ser inter-relacionado e, portanto não poder ser computado de forma isolada. Sendo uma tarefa difícil, apenas estimativas rudimentares de utilização de recursos podem ser determinadas.

Imprecisão

A introdução de restrições temporais adiciona outra característica à computação em tempo-real: ela pode ser *imprecisa*. Isto é, devido às restrições temporais dos dados e tarefas, os resultados produzidos por uma tarefa podem não ser exatos. Em sistemas onde o tempo de execução de uma tarefa é mais importante que o resultado exato, imprecisão pode ser tolerada. Por exemplo, em um sistema de controle de tráfego aéreo, a posição aproximada de um avião no instante certo, pode ser mais apropriada que o valor correto tarde demais. Geralmente a imprecisão permitida deve ser limitada. Considerando o exemplo anterior, a posição aproximada do avião deve ser diferente da verdadeira posição em apenas alguns metros. Observe que os STR consideram as restrições temporais das tarefas mas ignoram problemas de consistência lógica dos dados.

2.2 Sistemas de Gerenciamento de Bancos de Dados

Um banco de dados (BD) é um depósito autodescritivo de dados os quais são armazenados permanentemente em um ou mais arquivos. Um sistema de gerenciamento de bancos de dados (SGDB) é o *software* que gerência o acesso ao BD que pode ser compartilhado por vários usuários e programas. A execução de um programa que acessa ou modifica o conteúdo do banco de dados é chamada *transação* [EN94].

Um SGBD combina muitas características que permitem a descrição dos dados, a manutenção da integridade dos dados, o acesso eficiente aos dados e a execução correta de consultas e transações, mesmo na presença de concorrência e falhas, disponibilizando:

Suporte para gerenciamento dos dados, tal como indexação, permite acesso eficiente aos dados.

Suporte a transação, onde uma transação tem propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), assegura a execução correta das transações concorrentes e a manutenção da integridade dos dados mesmo na presença de falhas.

O objetivo das transações nesses sistemas é encontrar respostas corretas dentro do menor tempo possível.

2.3 Bancos de Dados Orientados a Objetos

Um banco de dados orientado a objetos (BDOO) é uma coleção de *classes* e instâncias dessas classes. As instâncias das classes são referenciadas como *objetos*. Uma classe *encapsula* um *conjunto de atributos* para suas instâncias e um *conjunto de operações* para manipular com as instâncias.

Uma operação é definida em duas partes. A primeira parte, chamada *interface* da operação, especifica o nome da operação e os argumentos (ou parâmetros). A segunda parte, chamada *método*, especifica a *implementação* da operação. As operações podem ser invocadas pela passagem de *mensagens* para um objeto, que inclui o nome da operação e os parâmetros. O objeto então executa o método para aquela operação. Um método pode invocar outros métodos em outros objetos. Este encapsulamento permite

a modificação da estrutura interna de um objeto e a implementação de suas operações sem a necessidade de alterar os programas externos que invocam estas operações. Uma operação pode ser aplicada a diferentes tipos de objetos. Assim uma operação pode ter várias implementações, que dependem dos tipos dos objetos onde serão aplicadas. Esta característica é chamada de *polimorfismo de operações.* Por exemplo, uma operação para calcular a área de um objeto geométrico pode ter diferentes implementações que vão depender do tipo do objeto (triângulo, círculo ou retângulo) [EN94].

Um dos principais objetivos dos BDOO é a manutenção da correspondência direta entre objetos do mundo-real e objetos do banco de dados de forma que os objetos não percam sua integridade e identidade e possam ser facilmente identificados e operados. Assim, um BDOO fornece a cada objeto um *identificador* único que é gerado pelo próprio sistema.

Outros dois conceitos importantes em BDOO são *classificação* e *herança.* Classificação permite a especificação de novas classes (subclasses) a partir de outras classes existentes (superclasses), onde cada subclasse *herda* as propriedades (estrutura e métodos) da superclasse, e podem ter propriedades locais. Assim, cada objeto na subclasse deve também ser membro da superclasse. Essa capacidade permite criar uma hierarquia de classes, onde as subclasses são especializações da superclasse.

Um BDOO armazena objetos *persistentes* permanentemente em armazenamento secundário e permite o compartilhamento desses objetos entre múltiplos programas e aplicações. Isso requer a incorporação de características básicas de gerenciadores de bancos de dados, tais como mecanismos de indexação, controle de concorrência e recuperação.

Uma vez que múltiplos usuários podem freqüentemente acessar muitas classes, a noção de transação pode ser usada para assegurar atomicidade das interações dos usuários. O modelo de transações aninhadas é o modelo natural das transações em um BDOO [DiP95b]. A execução de um método é tratada como uma transação que pode executar operações atômicas, ou invocar outros métodos em outros objetos. Isto é, uma operação em uma transação pode ser uma operação atômica ou outra transação, e assim as transações impõem uma estrutura em árvore. A transação na raiz da árvore é denominada *supertransação*; as outras são denominadas *subtransações.* Uma transação em um nível mais alto na árvore é chamada de *pai* e as subtransações nos níveis mais baixos

são chamadas de *filhos*. Uma subtransação pode ser completada[5] ou desfeita e reinicializada[6]. Quando uma subtransação completa, só terá efeito quando seu pai também completar. O reinicio de qualquer transação na árvore leva todas as subtransações a reinicializarem.

Os conceitos de orientação a objetos serão discutidos com mais detalhes no capítulo 3.

2.4 Sistemas de Gerenciamento de Bancos de Dados em Tempo-real

Sistemas de gerenciamento de bancos de dados para aplicações em tempo-real (SGBD-TR) podem ser vistos como uma integração de sistemas de gerenciamento de bancos de dados (SGBD) e sistemas em tempo-real (STR). Como um SGBD, ele processa transações e garante a integridade dos dados e, como um STR, ele satisfaz às restrições de tempo impostas às transações. Um SGBD-TR pode existir sozinho ou pode ser parte de um grande sistema com vários bancos de dados. Ele tem duas características principais: trata com dados temporais, ou seja, dados que são válidos apenas por períodos de tempo específicos, e algumas de suas transações podem ter restrições temporais. Estas características são bastante úteis para aplicações com tempo crítico que precisam colecionar, modificar e recuperar dados compartilhados.

Assim como em STR, velocidade de processamento não é suficiente para o bom desempenho de SGBD-TR, ou seja, o fato do *hardware* ser veloz não assegura que as transações serão escalonadas apropriadamente para satisfazer suas restrições temporais, nem assegura que os dados usados são temporalmente válidos. Para um SGBD-TR, uma transação que satisfaz suas restrições temporais usando dados obsoletos é considerada incorreta [SSH99].

Destaque-se a distinção entre objetivos de projeto quando são comparados SGBD e SGBD-TR. Enquanto os SGBD são projetados objetivando um bom tempo de resposta, os SGBD-TR são projetados para satisfazer às restrições temporais das transações. Quando uma transação com restrição temporal não pode ser executada no tempo espe-

[5] *commit*

[6] *rollback and restart*

cificado, vários problemas podem ocorrer tanto no ambiente quanto no banco de dados. O ambiente pode ser colocado em risco devido à falta de resposta do sistema durante eventos críticos. A validade dos dados no banco de dados também pode ser comprometida se esses não forem atualizados em tempo de forma a refletir o estado real do ambiente. Enquanto os SGBD são projetados para manter a consistência lógica dos dados e transações, os SGBD-TR são projetados para manter, sempre que possível, a consistência lógica e temporal dos dados e transações. Assim, enquanto o tempo de resposta das transações convencionais é considerado *requisito de qualidade*, ou seja, um tempo de resposta pequeno é desejável, o tempo de resposta das transações em tempo-real é considerado um *requisito de corretude*. Por outro lado, a consistência lógica é um *requisito de corretude* em bancos de dados convencionais, enquanto que para aplicações em tempo-real, é um *requisito de qualidade*, ou seja, é sempre desejável que os dados sejam consistentes [BLS97].

Em um SGBD-TR, quando um dado se torna inconsistente ou uma restrição temporal é violada, então o gerenciador deve disponibilizar meios para detectar esta violação e ações de recuperação devem ser tomadas. Essas ações de recuperação devem ser especificadas como parte da definição do banco de dados ou implementadas pela aplicação que está usando o banco de dados [PWPD96].

2.4.1 Características dos Dados

Alguns dados usados em um SGBD-TR devem refletir o estado real do ambiente da aplicação, portanto a semântica desses dados indica que os valores gravados são válidos apenas por um determinado intervalo de tempo. Um SGBD-TR faz esse intervalo explícito como parte do banco de dados. Assim, os intervalos de tempo especificam a *consistência temporal* dos dados [Ram93; SSH99]. A consistência temporal pode ser medida através da *consistência absoluta* dos dados e através da *consistência relativa* dos dados.

Consistência absoluta entre o estado do ambiente e como ele é refletido no banco de dados. Então, um item de dado deve ser gravado dentro de intervalos de tempo que garantam que ele reflete o estado real do ambiente. Essa medida surge da necessidade de manter a visão do sistema de controle consistente com o real

estado do ambiente. Por exemplo, em um sistema de transporte automatizado, o dado correspondente a um sensor que verifica a velocidade dos veículos, deve ser atualizado periodicamente, por exemplo, a cada cinco minutos. Assim, o valor da velocidade é absolutamente consistente se ele foi gravado nos últimos cinco minutos.

Consistência relativa entre os dados que são usados na computação de outros dados. Então, um conjunto de itens de dados deve ser gravado dentro de um mesmo intervalo de tempo, que possa representar aproximadamente o mesmo instante de tempo. Essa medida surge da necessidade de produzir novos dados a partir de dados gravados em tempos aproximados. Por exemplo, se o sistema de transporte do exemplo anterior computa o novo valor dos níveis de consumo de combustível usando os valores da *velocidade* e a *posição corrente* em que se encontra, é importante que esses valores tenham sido gravados aproximadamente no mesmo tempo, por exemplo, nos últimos trinta segundos, de forma que eles possam refletir o mesmo instante do ambiente, ou a computação não fará sentido.

2.4.2 Características das Transações

As transações de um SGBD-TR podem usar dados temporais e podem ser associadas a restrições temporais tais como prazo, tempo de início, período, entre outras. Desta forma, uma transação em SGBD-TR é considerada correta se ela satisfaz suas restrições temporais e usa dados temporalmente consistentes [SSH99].

Em [Ram93], Ramamritham caracteriza as transações de um SGBD-TR em três diferentes formas, baseada na natureza das transações: a origem das restrições temporais, a maneira como os dados são usados pelas transações, e o significado de executar uma transação antes de seu prazo.

Primeiro, elas são caracterizadas de acordo com as origens das restrições temporais impostas às transações. Algumas restrições vêm dos *requisitos de consistência temporal dos dados* e algumas vêm dos *requisitos impostos pelo ambiente.* O primeiro, geralmente, tem a forma de requisitos de periodicidade. Por exemplo:

```
A cada 20 segundos verifique a temperatura do ambiente.
```

O segundo, geralmente, tem a forma de prazos impostos às transações aperiódicas. Por exemplo:

```
Se temperatura > 1000 graus, dentro de 10 segundos, adicione líquido
    refrigerador ao reator.
```

As transações em tempo-real também são caracterizadas como: *transações de sensores (ou escrita)*, *transações de atualização* e *transações de leitura* [Ram93]. Todas elas podem ter restrições temporais. Transações de sensores são transações de escrita que obtêm o estado do ambiente e escrevem os dados no banco de dados. Elas são tipicamente periódicas. Transações de atualização podem ler e escrever no banco de dados periodicamente ou aperiodicamente. Transações de leitura, lêem dados do banco de dados e podem também ser periódicas ou aperiódicas.

Finalmente, elas também podem ser diferenciadas baseada no efeito de perder seu prazo como *estrita*, *suave* e *firme*, exatamente como para STR.

2.4.3 Propriedades ACID

As transações convencionais devem possuir algumas propriedades, denominadas propriedades ACID [EN94]. Essas propriedades garantem a consistência lógica dos dados e transações, no entanto não consideram os requisitos de consistência temporal existentes nos BDTR.

Para suportar aplicações em tempo-real, as propriedades ACID foram redefinidas para SGBD-TR, de forma a permitir melhor suporte para consistência temporal enquanto mantém suporte para consistência lógica. As novas definições utilizam informação semântica para determinar em que grau as propriedades ACID devem ser cumpridas [DiP95b; BLS97].

Redefinição das Propriedades ACID

As redefinições das propriedades ACID para transações em tempo-real incluem *atomicidade*, *consistência*, *isolamento*, e *durabilidade*.

A atomicidade garante que uma transação é totalmente executada ou nenhum passo dela é executado. Para transações em tempo-real, a execução atômica é seletivamente

aplicada as subtransações que precisam tratar com dados totalmente consistentes, ao invés da transação toda. Também, imprecisão lógica é permitida em BDTR, assim algumas vezes não se faz necessário desfazer toda a transação. Então uma transação pode terminar com um estado inconsistente desde que a imprecisão seja limitada.

Em geral, atomicidade pode não ser de interesse para algumas transações em tempo-real. Por exemplo, considere que uma transação que está atualizando dados sobre o mundo real, perde seu prazo e precisa parar de executar. Geralmente é desnecessário desfazer suas atualizações, já que os valores anteriores também estão desatualizados. Além disso, provavelmente é mais interessante ter um banco de dados parcialmente atualizado que um banco de dados totalmente desatualizado.

A propriedade de consistência garante que a execução de uma transação sempre transforma o estado consistente de um banco de dados em outro estado consistente. Transações em tempo-real devem suportar a definição de mecanismos para expressar uma negociação entre consistência lógica e temporal. Tais mecanismos podem levar a introdução de imprecisão no banco de dados.

Algumas aplicações em tempo-real podem suportar inconsistências no banco de dados. Por exemplo, dados que mudam freqüentemente, tais como a localização de um automóvel em um sistema automatizado de transporte, não precisam ser atualizados a cada mudança, o que poderia consumir bastante tempo de computação e recursos. Geralmente nessas aplicações as atualizações são feitas apenas quando dados atualizados forem necessários.

A propriedade de isolamento garante que as ações de uma transação não sejam visíveis a nenhuma outra transação até que ela seja comprometida. Isto implica que não existem dependências na execução das transações. Em bancos de dados em tempo-real, as transações podem se comunicar e sincronizar com outras transações de forma a executar funções de controle, portanto suas ações podem ser vistas por outras transações mesmo antes desta comprometer.

Alguns dados usados nestas aplicações precisam ser temporalmente consistentes e as transações podem ter restrições temporais. Portanto, os dados gerados por uma transação não comprometida podem ser vistos por outras transações para evitar que os dados tornem-se desatualizados, ou que percam seus *prazos*.

A propriedade de durabilidade garante que as ações de uma transação no banco

de dados sejam permanentes. Em BDTR alguns dados têm validade temporal e não precisam ser gravados.

Um BDTR deve refletir o estado real do ambiente, portanto quando ocorre uma falha no sistema é fácil recriar o estado do ambiente a partir da leitura dos sensores, ao invés de recriar o estado do ambiente a partir de dados gravados antes da ocorrência da falha pois esses dados provavelmente já serão inválidos. Assim, é desnecessário gravar dados que têm validade temporal em BDTR. É importante observar que em sistemas onde o histórico é necessário, os dados precisam ser gravados periodicamente.

2.4.4 Requisitos Gerais para SGBD-TR

Os requisitos gerais para SGBD-TR foram publicados em *Requirements for Military Database Management* citeGor93. Esses requisitos são divididos em nove classes que são documentadas em [Gor93] como segue:

1. *Requisitos gerais*: esta classe especifica os objetivos gerais, tais como, escalabilidade, modularidade, extensibilidade, entre outros, de um gerenciador de banco de dados.

2. *Serviços de gerenciamento básico de bancos de dados*: esta classe especifica os serviços básicos tipicamente oferecidos pelos gerenciadores convencionais de banco de dados, os quais devem ser incluídos nos gerenciadores de banco de dados em tempo-real.

3. *Distribuição*: esta classe trata da distribuição de dados através de bancos de dados homogêneos e fortemente acoplados, que juntos formam um único banco de dados lógico, conhecido como banco de dados distribuído. Ela especifica os requisitos para uma interface de gerenciador de banco de dados distribuídos.

4. *Heterogeneidade*: esta classe trata da distribuição dos dados através de bancos de dados heterogêneos e autônomos. Ela especifica capacidades para acesso a bancos de dados remoto, transações globais, sistemas múltiplos bancos de dados e sistemas de bancos de dados aliados.

5. *Processamento em tempo-real*: esta classe trata da necessidade de processamento em tempo-real dos modos *estrita, suave* e *firme*. Ela especifica capacidades para

gerenciar a atualização de dados com restrições temporais e a execução de transações com restrições de tempo. Estas capacidades habilitam os usuários e/ou programas de aplicação a controlarem o uso de recursos de acordo com as necessidades da aplicação.

6. *Tolerância a falhas*: esta classe trata das necessidades de segurança, disponibilidade e tolerância à falhas. Ela especifica capacidades para gerenciamento da coleção de informações sobre falhas e a formulação das respostas às falhas.

7. *Integridade*: esta classe trata das necessidades de integridade de dados. Ela especifica mecanismos para preservar a integridade dos dados, tais como: domínios, chaves, restrições de integridade referencial, afirmações, disparos e alertas.

8. *Segurança*: esta classe trata da necessidade de segurança multi-nível. Ela especifica requisitos para segurança multi-nível, incluindo rótulos, controle de acesso arbitrário, identificação e fiscalização.

9. *Serviços de gerenciamento de bancos de dados avançados*: esta classe especifica algumas das funções que estão tipicamente associadas a sistemas de gerenciamento de bancos de dados orientado a objetos e sistemas de gerenciamento de dados baseados em conhecimento. Estas funções são necessárias para o gerenciamento de dados complexos.

Esses requisitos são bastante amplos e, como tal, acredita-se que nenhum SGBD-TR seja capaz de fornecer suporte completo para todos eles [Pri95], mas algumas pesquisas estão sendo realizadas para cobrir grande parte desses requisitos [SSH99]. Considerando os objetivos deste trabalho, o interesse está principalmente na classe de processamento em tempo-real. Observe que a ênfase está na busca de um método para modelar BDTR. Deste modo os requisitos para esta classe serão apresentados com mais detalhes a seguir.

2.4.5 Requisitos para Processamento em Tempo-real

Nesta seção apresentam-se os requisitos para processamento em tempo-real das transações de um SGBD-TR.

Modos de Tempo-real

Um SGBD-TR deve fornecer suporte para modos de operação em tempo-real dos tipos *estrita, suave* e *firme*, e suporte para operação convencional.

Transações em Tempo-real

Um SGBD-TR deve fornecer capacidade para os usuários tratarem as transações em tempo-real onde as propriedades ACID são aplicadas seletivamente, e onde restrições temporais são especificadas.

Critério de Controle de Concorrência

Um SGBD-TR deve fornecer a capacidade de se especificar os critérios de controle de concorrência aos recursos compartilhados.

Consistência Temporal

Um SGBD-TR deve fornecer suporte para os usuários especificarem as restrições temporais dos dados e transações, isto é, um intervalo de tempo durante o qual um dado é considerado válido, e o tempo máximo que deve ser utilizado para executar uma transação.

Imprecisão Lógica Limitada

Um SGBD-TR deve permitir imprecisão lógica dos dados, e fornecer capacidade para restringir esta imprecisão, isto é, deve existir algum mecanismo para limitar este valor.

Imprecisão Temporal Limitada

Um SGBD-TR deve permitir imprecisão temporal dos dados, e deve fornecer capacidade para restringir esta imprecisão, isto é, deve existir algum mecanismo para limitar este valor.

Dados em Memória Principal

Um SGBD-TR deve fornecer a capacidade de especificar que certas partes do banco de dados devem residir em memória principal. Também deve ser capaz de manter a

persistência dos dados da memória principal. Este requisito pretende ajudar a fornecer a capacidade de predizer o tempo de acesso aos dados.

Tolerância a Falha Temporal

Um SGBD-TR deve ser capaz de suportar tolerância a falha temporal. Isto é, as violações das restrições temporais das transações e restrições de consistência temporal dos dados são falhas e devem ser tratadas como tais pela capacidade de tolerância à falhas do SGBD-TR. Este requisito indica que se uma restrição temporal for violada, o sistema deve ser capaz de detectar esta violação.

Limites de Utilização de Recursos

Um SGBD-TR deve permitir a especificação do pior caso de utilização de recursos (pelo menos, tempo de CPU, memória e dispositivos) pelas transações. A violação destes limites é falha e deve ser tratada como tal pela capacidade de tolerância a falhas do gerenciador.

Escalonamento em Tempo-real

Um SGBD-TR deve dispor de um escalonador que tente maximizar o número de transações que satisfazem seu prazo enquanto tenta manter tanto a consistência lógica quanto a consistência temporal dos dados.

Este último requisito, *escalonamento em tempo-real*, indica que um banco de dados em tempo-real freqüentemente precisa escolher entre manter consistência lógica ou temporal. Um SGBD tradicional trata apenas de manter a consistência lógica dos dados, e escalona as transações sem se importar quando elas serão executadas, e nem quanto tempo será necessário para executá-la. Por outro lado, em SGBD-TR, as ações podem ter restrições temporais que devem ser satisfeitas para manter as restrições temporais dos dados. Em algumas situações, estas restrições só podem ser satisfeitas se a consistência lógica dos dados for sacrificada. Ou ainda, pode ser que a consistência lógica só possa ser mantida se a consistência temporal for sacrificada. Uma vez que essas situações podem ser freqüentes em aplicações em tempo-real, um compromisso é estabelecido através do conceito de *imprecisão limitada*. Este conceito será discutido no Capítulo 4.

2.5 Problemas em Aberto para SGBD-TR

Apesar de uma quantidade significativa de pesquisas ter sido feita na área de BDTR, este campo está ainda na sua infância. Atualmente, apenas um número relativamente pequeno de pesquisadores está engajado nessa área, sendo necessário aumentar as atividades nessa área, pois muitas questões ainda estão em aberto [SSH99]. Abaixo mostramos algumas das principais questões a serem pesquisadas. Outras questões são discutidas em [SSH99].

Critérios de Corretude

Em bancos de dados convencionais, *seriação* é o critério de corretude primário para escalonamento de transações: O resultado produzido pela execução intercalada de um conjunto de transações deve ser idêntico a um produzido pela execução serial das transações. Embora escalonamentos seriáveis assegurem corretude, em muitas circunstâncias o custo de assegurar seriação em SGBD-TR pode ser muito alto, especialmente para classes de aplicações em tempo-real onde o tempo é essencial. Nesses casos, produzir um resultado aproximado em tempo hábil usando um escalonamento não-seriável pode ser melhor que produzir um resultado muito tarde usando um escalonamento seriável. Um desafio é definir novos e alternativos critérios para corretude de um BDTR e desenvolver métodos que negociem a seriação por tempo.

Aborto e Recuperação de Transações

Aborto e recuperação de transações consome um valioso tempo de processamento e pode afetar outras transações. Por exemplo, protocolos de controle de concorrência otimistas executam detecção e resolução de conflitos apenas no fim do processamento da transação. Quando um conflito é detectado, a transação é abortada e recomeçada. Em SGBD-TR isso pode causar um grande problema porque o aumento do tempo de processamento de uma transação pode fazer com que ela perca seu prazo e põe em risco os prazos de outras transações. Assim, uma tarefa difícil é estabelecer o tempo certo no qual a recuperação pode ser executada.

Modelagem

Vários tipos de dados têm diferentes tipos de propriedades temporais. Pesquisadores devem desenvolver modelos que associem propriedades temporais com tipos de dados e permitam a especificação clara e fácil dos relacionamentos entre consistência e restrições temporais.

Integração de Bancos de Dados em Tempo-real e Ativos

STR são inerentemente reativos, ou seja, eles devem responder a eventos externos ao ambiente assim como a eventos internos disparados por temporizadores ou estados/condições calculados. Nós precisamos desenvolver modelos reativos que considerem restrições temporais e algoritmos eficientes para detectar eventos em tempo de execução.

Análise de Negociação

Um SGBD-TR precisa algumas vezes negociar entre consistência lógica e temporal. Assim, novos mecanismos de controle de concorrência que permitam a negociação entre consistência lógica e temporal precisam ser pesquisados e desenvolvidos.

2.6 Modelos para Bancos de Dados em Tempo-real

Vários modelos têm sido propostos para expressar as características dos BDTR. Esses modelos incorporam vários aspectos dos modelos de bancos de dados tradicionais e vários aspectos de requisitos de consistência temporal. Na seção a seguir, serão apresentados os modelos de Ramamritham [Ram93], e o modelo RTSORAC [PWPD96].

2.6.1 Modelo de Ramamritham

Em [Ram93], é apresentado um modelo de banco de dados em tempo-real relacional, onde os dados podem apresentar restrições temporais absolutas e relativas.

Dados em Tempo-real

Neste modelo, os dados de um banco de dados em tempo-real têm o seguinte formato:

$$d : (valor, avi, timestamp)$$

onde d_{valor} denota o estado corrente de um dado d, $d_{timestamp}$ denota o tempo em que a observação relativa ao dado d foi feita, e d_{avi} denota o intervalo de validade absoluta do dado d. A *consistência temporal absoluta* do dado d é mantida no tempo corrente t se $(t - d_{timestamp}) \leq d_{avi}$.

Um conjunto de dados usado para derivar um novo dado forma um *conjunto de consistência relativa* R. Cada conjunto R é associado a um intervalo de validade relativa denotado por R_{rvi}. A consistência temporal relativa dentro do conjunto é mantida quando o *timestamp* de cada valor do conjunto R está dentro de R_{rvi}, isto é, quando $\forall d, d' \in R$, $d_{timestamp} - d'_{timestamp} \leq R_{rvi}$. O *timestamp* do dado derivado é uma função dos dados que foram usados para deriva-lo.

Então, um valor $d \in R$ tem o estado correto se e somente se:

1. d_{valor} é logicamente consistente - satisfaz todas as regras de integridade.

2. d é temporalmente consistente:

   consistência absoluta: $(t - d_{timestamp}) \leq d_{avi}$

   consistência relativa: $\forall d, d' \in R$, $d_{timestamp} - d'_{timestamp} \leq R_{rvi}$.

Considere o seguinte exemplo: Seja $temperatura_{avi} = 5$, $pressao_{avi} = 10$, $R = \{temperatura, pressao\}$ e $R_{rvi} = 2$. Se o $tempo-corrente = 100$, então $temperatura = (347, 5, 95)$ e $pressao = (50, 10, 97)$ são temporalmente consistentes. No entanto, $temperatura = (347, 5, 95)$ e $pressao = (50, 10, 92)$ não são temporalmente consistentes porque mesmo sendo absolutamente consistentes, são relativamente inconsistentes.

Transações em Tempo-real

Neste modelo, as transações são classificadas de três formas: de acordo com seu uso por uma transação, pela natureza das restrições temporais, e de acordo com o efeito de perder seu prazo.

As transações podem ser de três diferentes tipos: *transações de escrita* apenas escrevem os dados no banco de dados; *transações de atualização* derivam novos dados através de leitura e cálculos, e escrevem os novos dados no banco de dados; e *transações de leitura* apenas lêem dados do banco de dados.

As restrições temporais das transações são originadas dos requisitos temporais dos dados ou dos requisitos impostos pelo ambiente. Por exemplo, uma transação de atualização pode precisar executar a cada 10 segundos, devido ao dado que ela atualiza ter um intervalo de validade absoluta de 10 segundos. Enquanto isso, outra transação pode precisar executar apenas quando uma determinada situação for detectada no ambiente.

O efeito de perder o prazo é a terceira forma de classificar as transações. O modelo usa transações dos tipos *estrita, suave* e *firme*, como descrito na Seção 2.1.

2.6.2 Modelo RTSORAC

O modelo RTSORAC (*Real-Time Semantic Objects Relationships and Constraints*) [PWPD96] incorpora características que suportam os requisitos de um banco de dados em tempo-real em um modelo orientado a objetos. Três componentes são utilizados para modelar as propriedades de um banco de dados orientado a objetos em tempo-real: *objetos, relacionamentos*, e *transações*. Os objetos representam as entidades do sistema, os relacionamentos representam as associações entre os objetos e definem restrições interobjetos dentro do banco de dados. As transações são as entidades executáveis que acessam os objetos e relacionamentos no banco de dados.

Objetos

Um *objeto* consiste de cinco componentes, (N, A, M, R, FC), onde N é o identificador do objeto, A é um conjunto de atributos, M é um conjunto de métodos, R é um conjunto de restrições, e FC é uma função de compatibilidade. Cada um destes componentes será discutido a seguir.

Atributos

Cada atributo é da forma (N, V, T, I), onde N é o *nome* do atributo, V é o *valor* do atributo, T é o *rótulo de tempo*, ou seja, o tempo quando o valor foi gravado, e I é a *imprecisão* acumulada e indica a quantidade de imprecisão que foi introduzida no atributo.

Métodos

Os métodos são utilizados pelas transações para acessar instâncias dos objetos. Cada método tem a forma (N, Arg, Exc, Op, R), onde N é o nome do método, Arg são os argumentos do método, e cada argumento é composto dos atributos do objeto que são usados para passar informações entre os métodos. Arg é dividido em dois subconjuntos: *Arg_entrada* e *Arg_retorno*. *Arg_entrada* é usado para passar os argumentos que serão utilizados pelo método para atualizar os atributos, e *Arg_retorno* é um argumento cujo valor é computado pelo método e retornado para a transação invocadora. Exc é um conjunto de exceções que indicam quando o método não terminou normalmente, Op é um conjunto de operações que representam a implementação do método. Estas operações incluem leitura e escrita do valor de um atributo, laços, etc. R é um conjunto de restrições na execução do método, incluindo restrições temporais absolutas no método todo ou apenas em um subconjunto de operações do método. Ele é da forma $(N, OpSub, Pred, RR)$, sendo N o nome da restrição, $OpSub$, um subconjunto de operações, $Pred$ é um predicado (expressão booleana) que é especificado sobre o subconjunto de $OpSub$ para expressar restrições de precedência, restrições de execução e restrições de tempo, e a regra de reforço RR, que é executada quando *pred* é avaliado como falso, é da forma (Exc, Op, R), onde: Exc é um conjunto de exceções, Op é um conjunto de operações, e R são restrições na execução de RR.

Restrições

O conjunto de restrições permite especificar corretamente o objeto. Cada restrição é da forma $(N, AtrSet, Pred, ER)$, onde N é o nome da restrição, $AtrSub$ é o subconjunto de atributos do objeto, $Pred$ é uma expressão booleana que é especificada usando atributos de $AtrSub$. $Pred$ pode ser usado para expressar requisitos de consistência lógica através do uso do campo *valor* dos atributos. Ele pode expressar requisitos de consistência temporal através do uso do campo *rótulo de tempo* dos atributos. Ele pode expressar limites de imprecisão através do uso do campo *imprecisão* dos atributos. A regra de reforço RR é executada quando o predicado é avaliado como falso e é da forma (Exc, Op, CR). Exc é um conjunto de exceções que a regra de reforço pode indicar, Op é um conjunto de operações que representam as ações da regra de reforço, e CR é um

conjunto de restrições na execução da regra de reforço.

Função de Compatibilidade

A função de compatibilidade FC indica os métodos que podem ser executados concorrentemente. Para cada par ordenado de métodos (m_i, m_j), uma expressão booleana $BE_{i,j}$ é definida. $BE_{i,j}$ pode ser avaliada para determinar se m_i e m_j podem executar concorrentemente. Baseado na semântica da aplicação, a função de compatibilidade pode permitir a execução simultânea de métodos que introduzem imprecisão nos atributos e nos argumentos dos métodos. Portanto, além de permitir a compatibilidade entre duas invocações de métodos, a função de compatibilidade expressa informação sobre a quantidade de imprecisão permitida na invocação concorrente desses métodos.

Relacionamentos

Cada relacionamento representa uma agregação de dois ou mais objetos, e consiste de (N, A, M, R, FC, P, IR). Os primeiros cinco componentes de um relacionamento são idênticos aos componentes de um objeto. P é o conjunto dos objetos que podem participar do relacionamento e IR é o conjunto de restrições interobjetos. P é da forma $(N, TP, Card)$, onde N é o nome do objeto participante, TP é o tipo de participação no relacionamento, e $Card$ é a cardinalidade do participante. As restrições IR são da forma *(N,PartSet,Pred,RR)*, onde *N, Pred* e *RR* são como as restrições para os objetos, e *PartSet* é um subconjunto de *P*. O predicado é expresso usando-se objetos de *PartSet*, permitindo que restrições sejam especificadas sobre múltiplos objetos participantes do relacionamento. As regras de reforço são definidas da mesma forma que para os objetos, ou seja, por *(Exc,Op,R)*. No entanto, as operações em *Op* agora podem incluir invocações de métodos dos objetos participantes no relacionamento.

Transações

Uma transação consiste de cinco componentes (N, MI, L, R, P), onde N é o identificador da transação e MI é o conjunto de solicitações de invocação de métodos, sendo cada solicitação representada por $(M, Arg, temporal)$. O componente M é o identificador do método sendo invocado. Arg é o conjunto de argumentos do método. O campo

temporal especifica se a transação requer que os dados retornados sejam temporalmente consistentes.

O componente L de uma transação é o conjunto de solicitações e liberações de bloqueios[7]. Cada solicitação de bloqueio é associada a uma solicitação de invocação de um método. Uma transação pode solicitar um bloqueio antes de solicitar uma invocação de um método, talvez para assegurar alguma restrição de consistência lógica. Neste caso, a invocação é para uma *futura invocação de método.* A transação também pode solicitar o bloqueio simultaneamente à invocação do método. Neste caso o bloqueio é solicitado para uma *invocação simultânea do método.*

O componente R, é um conjunto de restrições nas operações da transação. Estas restrições são da mesma forma que as restrições especificadas para os métodos, *(N,OpSub,Pred,RR).*

O componente P define as prioridades de uma transação e é usado pelo gerenciador de banco de dados para executar o escalonamento das transações. Cada solicitação de invocação de método por uma transação deve ser executada de acordo com a prioridade da transação.

2.7 Conclusão

Neste capítulo apresentamos os conceitos básicos de STR, SGBD convencionais e SGBD-TR. Apresentamos também os requisitos necessários aos SGBD-TR e alguns modelos de BDTR. Desta forma pode-se identificar as características desejáveis para um método de modelagem com o objetivo de disponibilizar ao projetista mecanismos de alto-nível para descrever os modelos. O objetivo de dotar um tal método de modelagem de mecanismos de alto-nível é melhorar a capacidade de descrição disponível ao projetista, requisito imprescindível para descrição de sistemas complexos.

[7] *locks*

Capítulo 3

Modelagem Conceitual e Orientação a Objetos

Um *modelo* é uma abstração de uma entidade do mundo real. Modelos são construídos omitindo-se detalhes não essenciais, desta forma um modelo é mais fácil de manipular e entender que uma entidade real. No desenvolvimento de um sistema complexo o projetista precisa abstrair diferentes visões do sistema, construir modelos usando notações precisas, verificar se o modelo satisfaz os requisitos do sistema, e gradualmente adicionar detalhes para transformar o modelo em implementação [Rum91].

A orientação a objetos é uma abordagem para construir modelos conceituais de entidades do mundo real usando abstrações. Como tal define mecanismos mentais padronizados através dos quais é possível capturar as entidades contempladas. Neste capítulo apresentamos os principais mecanismos descritivos para caracterizar sistemas de objetos, além das duas principais metodologias baseadas na abordagem orientada a objetos: UML (*Unified Modeling Language*) e OMT (*Object Modeling Technique*).

3.1 Conceitos Fundamentais de Orientação a Objetos

Nas seções seguintes apresentamos os principais mecanismos descritivos da abordagem orientada a objetos [Boo94; Dou98; Rum91].

3.2 Objetos

A noção fundamental que dá embasamento à orientação a objetos é a de que o mundo é explicável em termos dos objetos que o compõe, suas estruturas, seus relacionamentos e seus comportamentos. Assim, *objeto* é a construção fundamental da abordagem orientada a objetos.

Um objeto pode ser visto como uma entidade que apresenta três propriedades fundamentais: *identidade*, *persistência* e *comportamento*.

Por *identidade* deve-se entender que objetos são distinguíveis dos demais, pela sua própria existência e não por eventuais propriedades descritivas que se possam atribuir. Um objeto é formado por um conjunto de atributos, que podem ser usados para distinguir os objetos. Todavia, no mundo real, é possível que dois ou mais objetos tenham o mesmo conjunto de atributos e que, ainda assim, sejam objetos distintos. Para capturar tal fenômeno, a orientação a objetos considera que todo objeto é inerentemente distinguível dos demais. Assim, diz-se que todo objeto tem identidade única.

Através da propriedade de *persistência*, a orientação a objetos assume que os objetos possuem estados. Um estado representa a memória do objeto em relação aos seus atributos, isto é, os últimos valores adquiridos ou computados. Assim, interações entre objetos podem ser refletidas como mudanças em seus estados.

O *comportamento* de um objeto determina as possibilidades de reação de um objeto em resposta a estímulos externos ou internos. O comportamento de um objeto é usualmente determinado por um conjunto de *operações*, que quando ativas podem alterar o seu estado ou gerar novos estímulos. Em geral, estímulos são denominados *mensagens* ou *eventos*.

3.2.1 Abstração

O termo *abstração* em orientação a objetos é usado para formar a expressão *mecanismos de abstração* que se refere a qualquer um dos mecanismos usados para conceber sistemas na orientação a objetos: classificação, herança/especialização, agregação, associação, etc. Isoladamente, o termo abstração consiste no ato de mentalizar apenas as qualidades essenciais de alguma entidade, ignorando suas particularidades e detalhes irrelevantes. Assim, abstração é uma capacidade fundamental do ser humano que permite tratar com

a complexidade de alguns sistemas.

3.2.2 Classificação

Uma *classe* é uma descrição abstrata de objetos que compartilham a mesma estrutura de dados (atributos) e o mesmo comportamento (operações). Uma classe descreve um conjunto possivelmente infinito de objetos. Cada objeto que corresponde à descrição dada é denominado de *instância* da classe. Cada instância da classe tem seu próprio valor para cada atributo, mas compartilha o nome do atributo e as operações com outras instâncias daquela classe. Assim, *classificação* é o mecanismo de abstração que permite a descrição de classes em termos de objetos similares.

3.2.3 Encapsulamento e Interface

A *interface* de um objeto é a visão que ele apresenta para o mundo. O mecanismo de *encapsulamento* consiste na separação da interface de um objeto, acessível por outros objetos, dos detalhes de sua implementação interna, escondida dos outros objetos. Seu objetivo principal é permitir que as partes do sistema sejam independentes de possíveis alterações dos detalhes internos de outras partes. Ela pode ser vista como a separação entre a abstração e a implementação.

3.2.4 Herança

Herança é o mecanismo de abstração usado para descrever uma entidade *receptora* a partir de uma entidade *doadora* apenas pelas diferenças específicas que sejam necessárias. Assim, a entidade receptora, denominada *subclasse*, *herda* as propriedades (atributos e operações) da entidade doadora *superclasse*, sem que com isso a entidade doadora perca tais propriedades.

O mecanismo de herança, na sua forma mais completa, permite que qualquer diferença seja introduzida na subclasse. Desta forma, uma subclasse pode ser obtida pela eliminação e/ou adição indiscriminada de propriedades (atributos ou operações) à superclasse. Outras formas de herança existem, porém está fora do escopo deste trabalho apresentar uma discussão detalhada sobre o assunto. O leitor interessado pode referir-se à literatura sobre o assunto [BB94; BP94; Tai96; Weg88; Weg92; Weg95;

Weg96]. A relação de herança define uma hierarquia entre classes, onde o número de níveis dessa hierarquia é, potencialmente, qualquer valor inteiro positivo. Observe que o mecanismo de herança, além de permitir a descrição de entidades pela inclusão de detalhes, também promove a organização das descrições de classes, o que facilita as atividades de manutenção, dado que este mecanismo inibe a duplicação desnecessária de descrições.

3.2.5 Agregação

Agregação é um mecanismo de abstração que permite descrever uma entidade pela descrição das suas *partes* e pela indicação de como essas partes se relacionam. Desta forma, subentende-se que as propriedades da entidade como um todo podem ser obtidas a partir das propriedades das partes e da forma como são integradas.

É importante observar que, ao usarmos a agregação, os objetos componentes são partes intrínsecas do objeto composto. Assim, o objeto composto é único e indivisível, e nenhum dos objetos componentes pode ser eliminado sem prejuízo para toda a entidade. Devido a isto, a agregação é também conhecida por relacionamento *todo-parte*.

3.2.6 Associação e Ligação

As classes podem ser relacionadas. *Associações* e *ligações* são os meios pelos quais estabelecem-se relacionamentos entre classes.

Uma ligação é uma conexão física ou conceitual entre instâncias de classes. Uma ligação é uma instância de uma associação.

Uma associação descreve um conjunto de ligações com estrutura e semântica comum. Assim, para que dois objetos possam interagir, é necessário que exista uma *ligação* entre eles. E para que dois objetos suportem uma ligação, entretanto, é necessário que uma *associação* seja declarada entre as classes de que são instâncias. Portanto, o mecanismo de associação permite explicitar que objetos das classes em questão podem suportar ligações.

3.2.7 Mensagens

A comunicação entre os objetos é feita através da passagem de *mensagens*. Uma mensagem é uma sinalização de um objeto a outro para que execute uma operação.

3.3 Linguagem de Modelagem Unificada *(Unified Modeling Language - UML)*

UML é uma linguagem de modelagem de terceira geração para a especificação de sistemas orientados a objetos. Recentemente ela foi adotada pelo OMG (Object Management Group) como a linguagem padrão de modelagem de sistemas. Ela é o resultado de pesquisas de um grupo de metodologistas que juntaram em uma única linguagem o melhor da abordagem orientada a objetos [Dou98].

UML oferece uma variedade de diagramas para descrever os diferentes aspectos dos sistemas e distingue entre a noção de modelos e diagramas. Um *modelo* contém todos os elementos sobre o sistema em consideração, independentemente de como esses elementos são visualmente apresentados. Um *diagrama* contém uma visão particular de alguns elementos do modelo e geralmente expõe apenas um subconjunto dos elementos do modelo. Desta forma um modelo pode existir em vários diagramas, e cada diagrama captura uma visão diferente do modelo do sistema. Assim, um modelo UML é um conjunto de diagramas que descreve e documenta a estrutura e o comportamento de um sistema.

Nesta seção apresentamos os principais diagramas suportados em UML, com base em Douglass [Dou98].

3.3.1 Modelo de Contexto e Modelo de Casos de Uso

Normalmente os sistemas em tempo-real necessitam interagir com seu ambiente externo. O conjunto de objetos externos e suas interações com o sistema formam a base para os requisitos de análise do sistema. Em UML, isso é expresso através de dois modelos: *modelo de contexto*[1] e *modelo de caso de uso*[2]. O modelo de contexto é expresso como um

[1] context model
[2] use case model

modelo de objetos no qual o sistema é tratado como um único objeto que envia e recebe mensagens de objetos externos. O modelo de caso de uso apresenta a decomposição das funcionalidades primárias do sistema e os protocolos necessários para satisfazer os requisitos funcionais.

Diagrama de Contexto

O *contexto de um sistema* é um mapa do mundo de interesse para o sistema. No mundo da análise estruturada, o diagrama de contexto representa esta visão do mundo [Dou98]. Assim, um modelo de contexto é representado através de um *diagrama de contexto*. Um diagrama de contexto apresenta um único *objeto sistema* interagindo um ou mais *objetos externos*, onde o objeto sistema inclui qualquer sensor e agentes internos do sistema e os objetos externos são os dispositivos monitorados, controlados, ou que interagem com o objeto sistema. Um diagrama de contexto captura as informações sobre o ambiente do sistema, incluindo os agentes com os quais ele deve interagir. Ele também permite a captura e a caracterização das mensagens e eventos existentes entre o sistema e seu ambiente.

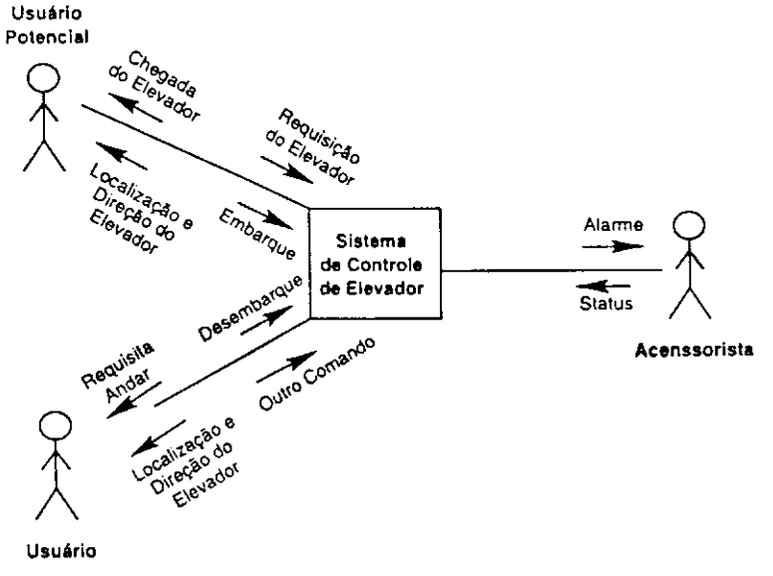

Figura 3.1: Diagrama de contexto para um sistema de controle de elevador

Na Figura 3.1 mostramos um diagrama de contexto para um sistema de controle de elevador conforme introduzido em [Dou98].

	Evento	Resposta do Sistema	Padrão de Direção	Padrão de Chegada	Desempenho e Sincronização	Resposta
1	Potencial usuário solicita elevador	a) Ativa botão b) Seleciona elevador c) Envia elevador selecionadoao andar	ENTRADA	Episódico	Assíerono	a) Botão 0,5 s b) 5 s c) O mais cedo
2	Usuário requisita andar	a) Ativa botão b) Envia elevador para o andar	ENTRADA	Episódico	Assíerono	Botão 0,5 s
3	Usuário seleciona Para na chave Para/Continua	Para elevador	ENTRADA	Episódico	Assíerono	2,0 s
4	Usuário selec. Continua na chave Para/Continua	Move elevador para atender as requisições pendentes	ENTRADA	Episódico	Assíerono	2,0 s
5	Usuário obstrui a porta durante fechamento	A porta é aberta novamente e o temporizador é reinicializado	ENTRADA	Episódico	Assíerono	Detecta 0,5 x
6	Usuário pressiona o botão abrir	Porta permanece aberta, e o temporizador de fechamento é reinicializado	ENTRADA	Episódico	Assíerono	0,5 s
7	Usuário pressiona o botão fechar	Início do fechamento da porta	ENTRADA	Episódico	Assíerono	0,5 s
8	Tempo para fechar a porta expirou	Início do fechamento da porta	ENTRADA	Episódico	Assíerono	0,5 s
9	Usuário pressiona botão de emergência	Notifica operador	ENTRADA	Episódico	Assíerono	1 s
10	Elevdor chega ao andar	a) Desativa botão b) Anuncia chegada c) Inicia ciclo de abertura	ENTRADA	Episódico	Assíerono	a) 1s b) 1s d) 2s
11	Elevador deixa andar	Elevador vai para o destino mais próximo na lista de destinos	ENTRADA	Episódico	Assíerono	1 s
12	Operador comanda liberação de granpos	Elevador vai para o destino mais próximo na lista de destinos	ENTRADA	Episódico	Assíerono	1 s
13	Tensão do cabo falha	a) ativa os grampos para travar b) Notifica via alarme	<interno> SAÍDA	Episódico	Assíerono	a) 0,25 s latência 0,5 s atender b) 1 s

Tabela 3.1: Lista de eventos externos para um sistema de controle de elevador

Lista de Eventos Externos

Em um STR, os eventos externos desempenham um papel fundamental na definição e restrição do comportamento do sistema. Na UML, esses eventos são declarados em uma *lista de eventos externos.* Nesta lista detalham-se os eventos externos que são de interesse para o sistema, possibilitando não só a definição dos eventos, mas também a resposta esperada pelo sistema, seus padrões de chegada, e a origem do evento.

Na Tabela 3.1 mostra-se uma lista de eventos externos para o sistema de controle de elevador. Observe que a lista fornece um número de atributos para capturar as propriedades essenciais dos eventos.

Diagramas de Caso de Uso

Um *diagrama de caso de uso* possibilita representar casos gerais de interação entre o objeto sistema e os objetos externos. O sistema é decomposto em unidades funcionais, chamadas *casos de uso*, que interagem com atores externos e entre si.

Os diagramas de casos de uso possibilitam uma visão ampla das funções primárias do sistema de uma forma simples, ou seja, eles descrevem como o sistema será usado. Esses diagramas podem ser usados para orientar os desenvolvedores na definição dos requisitos para o sistema. Eles podem ser decompostos em outros diagramas de caso de uso e finalmente em cenários que mostram seqüências detalhadas da interação entre objetos.

Cenários são instâncias de casos de uso. Cenários podem definir protocolos elaborados consistindo de mensagens enviadas em uma seqüência particular. Os cenários podem ser diretamente associados a um número de requisitos detalhados do sistema de forma que suas características e implicações são claras.

O relacionamento entre um diagrama de caso de uso e seus protocolos é mostrado na Figura 3.2. O diagrama de caso de uso utiliza o protocolo (e seus eventos e fluxos de dados) para satisfazer às responsabilidades do sistema. Os eventos e fluxos de dados devem ser suficientes para satisfazer os requisitos de uso do sistema.

A Figura 3.2 mostra os limites do sistema como um retângulo, e dentro dele as elipses representam os casos de uso. O sistema interage com os objetos externos fora do sistema. Esses objetos externos conectam-se aos casos de uso, dentro do sistema, através dos protocolos.

Cenários

Como mencionado anteriormente, cenários são instâncias de diagramas de caso de uso. Eles modelam seqüências de mensagens entre objetos colaborando para produzir o comportamento do sistema. No início do processo de análise, os objetos disponíveis para cenários são apenas o sistema e os objetos externos identificados nos diagramas de contexto e caso de uso. Mais tarde, o sistema é decomposto em objetos, e assim o processo de obtenção de cenários é novamente aplicado. Os objetos identificados são decompostos em múltiplos objetos, e as mensagens são decompostas em protocolos consistindo de

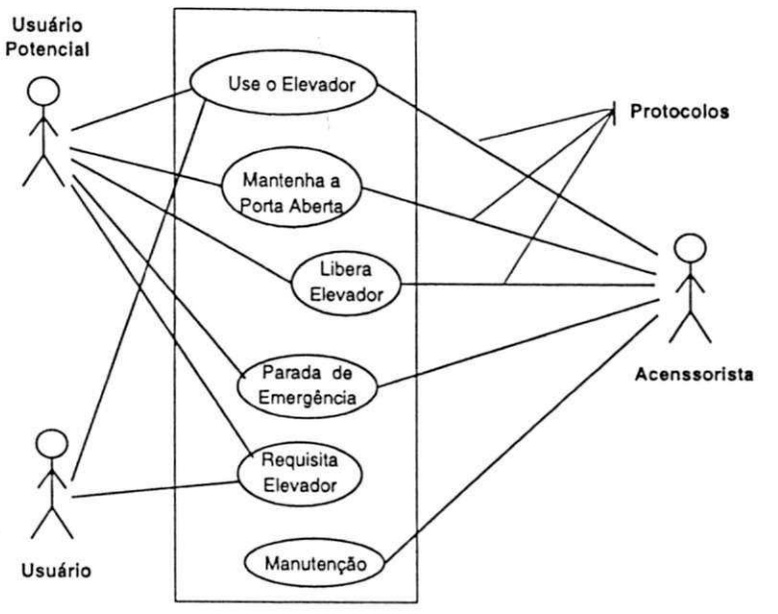

Figura 3.2: Diagrama de caso de uso para um sistema de controle do elevador

muitas mensagens. Assim como o refinamento interativo do modelo de objeto identifica mais classes, o refinamento de cenários adiciona mais detalhes às interações.

Os cenários podem ser representados através de dois tipos de diagramas: *diagramas de seqüência* e *diagramas de colaboração.*

Diagramas de Seqüência

Diagramas de seqüência possibilitam representar a seqüência das mensagens entre os objetos. Na Figura 3.3 mostramos o diagrama de seqüência para o sistema de controle de elevador. Os principais elementos gráficos usados na maioria dos diagramas de seqüência são mostrados nessa figura. As linhas verticais representam objetos, com o nome do objeto acima ou abaixo da linha. As linhas horizontais são mensagens. Cada mensagem parte do objeto origem e termina no objeto destino e tem o nome da mensagem na linha. Essas linhas podem ser usadas posteriormente para especificar operações com uma lista de parâmetros de entrada e valores de retorno.

Em [Dou98] o leitor pode encontrar mais detalhes relativos aos diagramas de seqüência no contexto da UML.

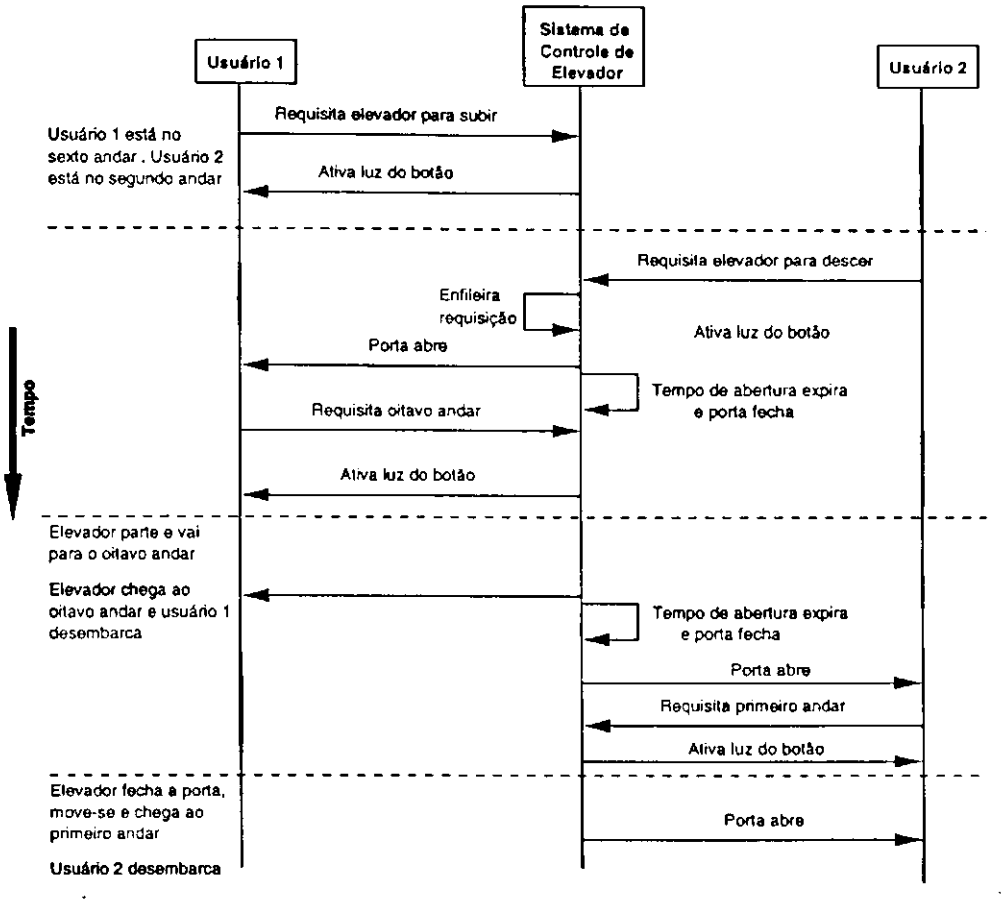

Figura 3.3: Diagrama de seqüência para um sistema de controle do elevador

Diagramas de Colaboração

Os diagramas de colaboração possibilitam ao projetista descrever basicamente as mesmas informações dos diagramas de seqüência. A diferença é que os diagramas de seqüência permitem focalizar a seqüência das mensagens, enquanto que os diagramas de colaboração focalizam a estrutura estática dos objetos do sistema.

Os componentes dos diagramas de colaboração são: objetos, relacionamentos e mensagens. Na Figura 3.4 mostramos o diagrama de colaboração para o sistema de controle de elevador com esses componentes. Os nomes dos objetos identificam os objetos participantes, e as ligações mostram quais os objetos que colaboram entre si. Para que dois objetos se comuniquem via mensagens é necessário uma ligação entre eles. As mensagens são identificadas com identificadores de mensagens. Um número de seqüência precede o identificador para definir o fluxo de mensagens. A direção da mensagem define o remetente e o destinatário da mensagem. Os diagramas de colaboração suportam uma sintaxe muito mais rica do que a apresentada na Figura 3.4. Exemplos podem ser

encontradas em [Dou98].

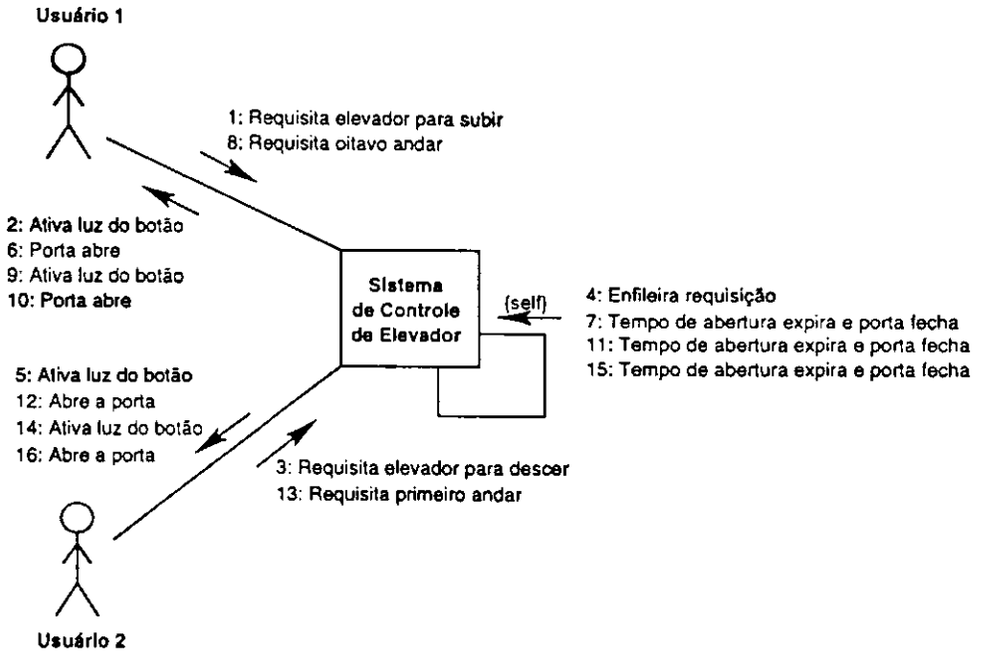

Figura 3.4: Diagrama de colaboração para um sistema de controle de elevador

3.3.2 Análise com UML

Os diagramas apresentados na seção anterior formam a base para a análise do sistema. A análise é dividida em duas etapas: definição da *estrutura estática* do sistema e definição do *comportamento* do sistema.

Definição da Estrutura Estática do Sistema

Nessa etapa identificam-se os principais objetos e classes, e seus relacionamentos dentro do sistema. Esses objetos e classes possuem atributos e comportamento que permitem satisfazer suas responsabilidades. Assim, para cada classe e objeto deve-se também identificar os principais atributos e operações. Os objetos, classes e relacionamentos são então agrupados em *Diagramas de Classes* e *Diagramas de Objetos*. Um diagrama de classes é um esquema que mostra a estrutura estática do sistema em termos de classes e seus relacionamentos. Um diagrama de classes é uma descrição do caso geral na modelagem de um sistema. Diagramas de objetos são idênticos a diagramas de classes, exceto que eles permitem mostrar apenas instâncias de classes. Diagramas de objetos

são usados para descrever exemplos com o objetivo de esclarecer um diagrama de classes complexo.

Na Figura 3.5 mostra-se o diagrama de classes para o sistema de controle de elevador. Ele contém as classes fundamentais, suas associações, e multiplicidade de suas associações. Vários tipos de associações podem ser representados em diagramas de classes. A notação completa para diagramas de classes pode ser encontrada em [Dou98].

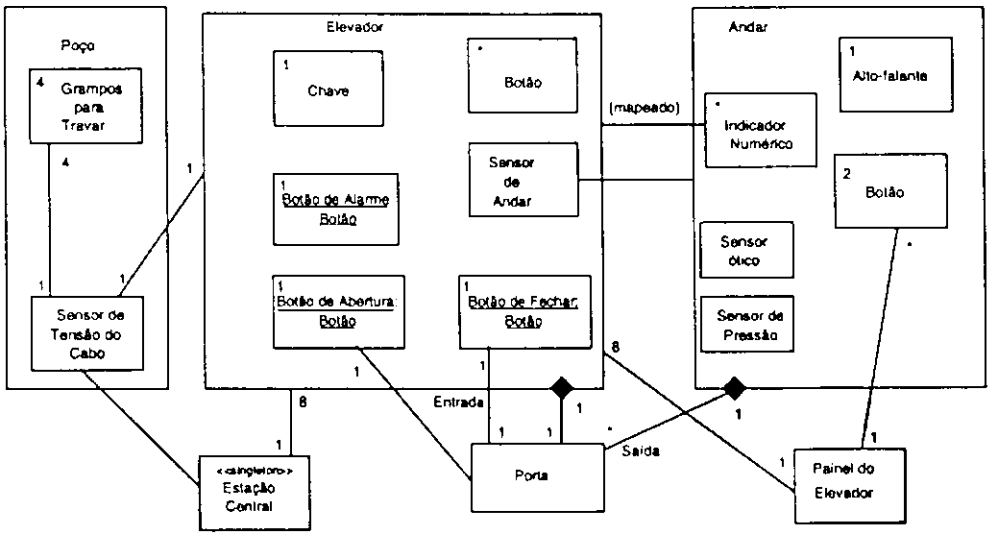

Figura 3.5: Primeira visão para o diagrama de classe de estudo de caso para um sistema de controle de elevador

Definição do Comportamento do Sistema

Nessa etapa define-se o comportamento dinâmico dos objetos do sistema. Na UML o comportamento dos objetos é modelado através de uma classe de diagramas de estado denominada *statecharts* [Har87]. Os diagramas de estado servem como uma conexão entre a estrutura de um sistema orientado a objetos e seu comportamento.

Diagramas de estado relacionam estados e eventos. Um *evento* é algo que acontece em um determinado instante de tempo. Um *estado* é uma abstração dos valores dos atributos e ligações de um objeto. Quando um evento ocorre, o próximo estado depende do estado corrente e do evento. Uma mudança de estado causada por um evento é denominada de *transição.*

Um diagrama de estado é um grafo cujos nós são estados e cujos arcos são transições rotuladas pelo nome do evento. Estados são representados através de retângulos com

bordas arredondadas, que opcionalmente, contêm um nome. Uma transição é representada por uma seta direcionada começando no estado inicial e terminando no estado alvo; o rótulo na seta é o nome do evento que causa a transição. As características essenciais dos diagramas de estado são apresentadas na Figura 3.6

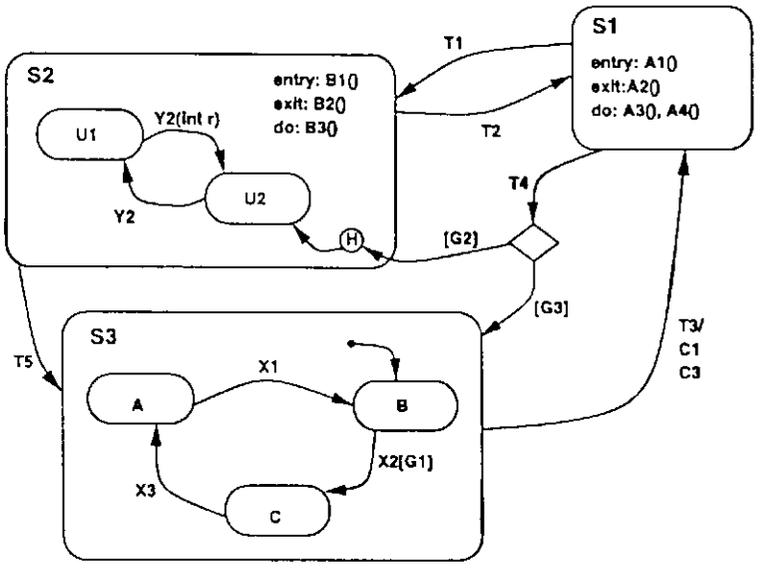

Figura 3.6: *Statechart* básico

Um diagrama de estado especifica a seqüência de estados causada por uma seqüência de eventos. Se um objeto está em um estado e um evento rotulado em uma de suas transições ocorre, então aquela transição dispara e o objeto alcança o estado alvo.

Diagramas de estado podem ser aninhados. O estado mais externo é chamado de superestado e os internos são chamados de subestados. Isso permite o refinamento dos diagramas.

Estados e transições podem ser associados a ações. Estados podem ter *ações de entrada ou saída*, bem como *atividades*. Ações de entrada ou saída são operações que são executadas quando o sistema entra ou sai de um estado, respectivamente. Atividades são executadas enquanto o estado está ativo. Atividades são indicadas no diagrama de estado através da palavra *do*. Assume-se que o tempo para executar as ações é insignificante e que as atividades são executadas durante o tempo em que o estado está ativo.

Transições podem ter parâmetros de entrada, guardas e ações. A sintaxe para transições é:

Nome-do-evento(parâmetros)[guarda]/lista-ações^lista-eventos, onde:

Nome-do-evento: é o nome do evento que dispara a transição.

Parâmetros: lista contendo os dados passados com o sinal do evento.

Guarda: Expressão booleana que deve ser avaliada como VERDADEIRA para a transição ser executada.

Lista-ações: Lista de operações executadas como resultado do disparo da transição.

Lista-eventos: Lista de eventos disparados como resultado do disparo da transição.

O modelo do comportamento dinâmico de um sistema é representado por um conjunto de diagramas de estado que interagem entre si através de eventos compartilhados.

Semântica dos Diagramas de Estado UML

Um diagrama de estado recebe e reage a eventos do ambiente. As reações incluem enviar novos eventos para outros objetos e executar métodos internos do objeto. Os diagramas de estado UML usam a mesma notação dos diagramas de estado *(statecharts)* do Harel [Har87], no entanto a semântica é diferente. A semântica de execução de um diagrama de estado UML é definida em termos de uma máquina hipotética cujos componentes são:

Uma *fila de eventos* que mantém instâncias de eventos até que eles sejam enviados,

Um *mecanismo de envio de eventos* para selecionar e desenfileirar instâncias de eventos da fila de eventos, e

Um *processador de eventos* que processa os eventos enviados.

Os eventos são enfileirados e enviados um por um. Cada objeto irá executar um algoritmo, o *run-to-completion (rtc)*, que envia e executa eventos da fila de eventos. Não existe uma sincronização perfeita no envio de eventos, ou seja, pode se passar uma quantidade de tempo arbitrária a partir do momento que um objeto recebe um evento até que o evento seja finalmente consumido.

A principal tarefa do processador de eventos é determinar quais são as transições habilitadas, ou seja, que podem ser disparadas com base no primeiro evento da fila. É possível que um evento habilite várias transições e que essas transições estejam em conflito, ou seja, elas têm o mesmo estado origem e diferentes estados alvos. Apenas uma das transições conflitantes pode disparar e essa decisão é tomada pelo processador de eventos com base na prioridade da transição ou, se elas têm a mesma prioridade, não deterministicamente. O conjunto de transições habilitadas são agrupadas em um *passo*. As transições no passo são então executadas seqüencialmente, mas o processador de eventos irá considerar o próximo evento na fila apenas quando todas as transições do passo tiverem sido processadas. Observe que a idéia do algoritmo *rtc* é assegurar que um evento só pode ser desenfileirado e enviado depois que todos os eventos prévios tiverem sido completados.

Observe que a semântica do algoritmo *rtc* é bastante especializada para ser útil na modelagem de alguns tipos de aplicações, como por exemplo aplicações em tempo-real que precisem definir concorrência intra-objetos. Também, a semântica do algoritmo *rtc* é sub-especificada, por exemplo, não existe especificação da semântica da fila de eventos, ou seja, se ela é FIFO, ordenada por prioridade, etc. Também a ordem de execução das ações e atividades não é especificada. Suponha que um *statechart* entre em uma hierarquia de estados aninhados, para os quais atividades foram definidas. As atividades do superestado podem ser intercaladas com as atividades do subestado? Elas devem terminar antes das atividades do subestado começar? Estas e outras questões precisam ainda ser esclarecidas na linguagem UML, como declarado em [CSW97].

Mostrando Cenários

UML oferece duas formas para mostrar cenários que são particularmente úteis em sistemas em tempo-real: *diagramas de tempo*, que permitem visualizar restrições temporais, e *diagramas de seqüência*, que possibilitam visualizar a ordem das operações. Os *diagramas de seqüência* foram discutidos na Seção 3.3.1. A seguir descrevemos os diagramas de tempo.

Diagramas de Tempo

Um *diagrama de tempo* é uma representação simples no tempo, onde o tempo é mostrado no eixo horizontal, o objeto no eixo vertical, e o estado é uma faixa horizontal sombreada através do diagrama. Diagramas de tempo são bons para mostrar o comportamento temporal e podem ser usados para analisar o tempo de tarefas periódicas e aperiódicas. Quando usadas dessa forma, alguns elementos mostrados são: períodos, prazos, tempo de início, tempo de execução, entre outros. A forma de um diagrama de tempo é apresentada na Figura 3.7.

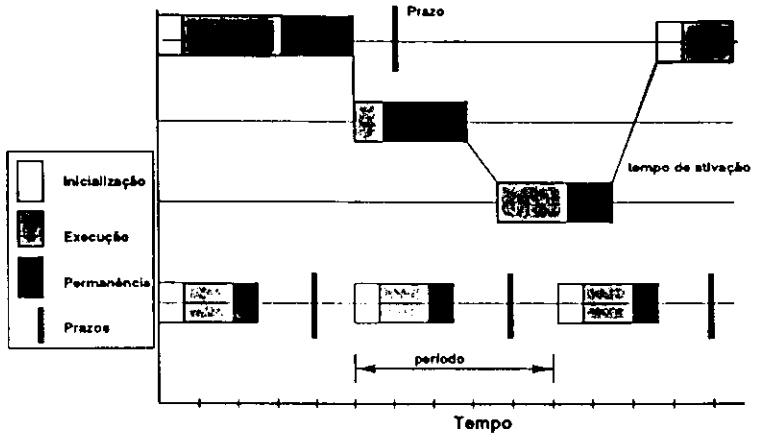

Figura 3.7: Diagrama de tempo para as tarefas

3.4 Técnica de Modelagem de Objetos *(Object Modeling Technique - OMT)*

OMT é uma metodologia orientada a objetos para modelagem de sistemas [Rum91]. Sua aplicação resulta em três tipos de modelos. O *modelo de objetos* representa os aspectos estruturais, estáticos, *os dados* de um sistema. O *modelo dinâmico* representa os aspectos temporais, comportamentais, *de controle* de um sistema. O *modelo funcional* representa os aspectos *funcionais* de um sistema. Os três tipos de modelos separam o sistema em visões ortogonais que podem ser representadas e manipuladas com uma notação uniforme. Os diferentes modelos não são completamente independentes, no entanto cada modelo pode ser examinado e entendido individualmente. As interconexões entre os modelos são limitadas e explicitas. Nesta seção apresentamos cada um desses

modelos.

3.4.1 Modelo de Objetos

Após identificar e caracterizar o sistema a ser modelado, o modelo de objetos é o primeiro a ser construído.

O modelo de objetos descreve a estrutura estática de um sistema, ou seja, os objetos, os relacionamentos entre os objetos, e os atributos e operações que caracterizam cada classe de objetos. O modelo de objetos serve de base para a construção dos modelos dinâmico e funcional. Ele facilita a comunicação com os usuários e serve para documentar a estrutura do sistema. O objetivo da construção do modelo de objetos é capturar os conceitos fundamentais do mundo real que são importantes para uma aplicação.

O modelo de objetos é representado graficamente através de diagramas de classes. Os diagramas de classes usados no modelo OMT são similares aos diagramas usados na UML, discutidos na Seção 3.3, com pequenas variações na notação. No entanto a essência de modelagem é a mesma, portanto não a exemplificaremos aqui.

3.4.2 Modelo Dinâmico

O modelo dinâmico permite descrever os aspectos de um sistema que tratam com tempo e seqüência de operações, tais como, eventos que indicam mudança, seqüência de eventos, estados que definem o contexto dos eventos, e a organização de eventos e estados. O modelo dinâmico possibilita capturar o *controle* do sistema, ou seja, a seqüência das operações que ocorrem no sistema, sem mostrar o que as operações fazem, sobre quem elas operam, ou como elas são implementadas. O modelo dinâmico é representado graficamente através de diagramas de estado. Cada diagrama de estado mostra as seqüências de estados e eventos permitidas para uma classe de objetos. Diagramas de estado podem referenciar outros modelos. As ações no diagrama de estado correspondem a funções no modelo funcional, enquanto que os eventos correspondem às operações no modelo de objeto.

Os diagramas de estado usados em OMT são semelhantes aos diagramas usados no modelo UML, discutidos na Seção 3.3, portanto não os discutiremos aqui.

3.4.3 Modelo Funcional

O modelo funcional descreve os aspectos de um sistema que dizem respeito a transformações de valores, isto é, funções, restrições e dependências funcionais. O modelo funcional permite ao projetista especificar o significado das operações no modelo de objetos e as ações no modelo dinâmico. Permite ainda explicitar *o que* um sistema faz sem se importar em *como* ou textitquando ele faz. O modelo funcional é representado através de diagramas de fluxo de dados (DFD) [EN94].

3.4.4 Relacionamento entre os Modelos

Cada modelo descreve um aspecto do sistema, no entanto contêm referências para os outros modelos. O modelo funcional é uma especificação do que acontece, o modelo dinâmico especifica quando acontece, e o modelo de objetos especifica sobre quem acontece. As operações no modelo de objetos correspondem aos eventos no modelo dinâmico e as funções no modelo funcional. O modelo dinâmico descreve a estrutura de controle dos objetos. Ele mostra decisões que dependem dos valores dos objetos e que causam ações que mudam os valores dos objetos e invocam funções. O modelo funcional descreve funções invocadas por operações no modelo de objetos e ações no modelo dinâmico. As funções operam sobre os dados especificados no modelo de objetos. O modelo funcional também mostra restrições nos valores do objeto.

É importante observar que podem ocorrer ocasionais ambigüidades, uma vez que quando lidando com abstrações nem sempre é possível definir informalmente as fronteiras entre os diferentes objetos [Rum91].

3.5 Conclusão

Neste capítulo apresentamos os conceitos e propriedades fundamentais da abordagem orientada a objetos que dão suporte as metodologias de modelagem baseadas nesta abordagem. Também apresentamos duas das principais metodologias de desenvolvimento de sistemas orientadas a objetos: UML e OMT. Embora UML seja bem mais recente que OMT, a tratamos com mais detalhes, pois esta é atualmente a mais importante linguagem de modelagem de sistemas e tem sido foco de muita pesquisa e desenvolvimento

nos últimos anos.

Atualmente UML é ainda muito pesada. Apenas parte dela é bem entendida; a definição de outras partes precisa ainda ser investigada profundamente para esclarecer seus relacionamentos com os construções centrais da UML, ou seja, os diagramas de classes e os diagramas de estado. Por exemplo, os diagramas de caso de uso e seus diagramas associados, isso é, os diagramas de seqüência e colaboração, são valiosos para usuários e engenheiros de requisitos tentarem elaborar o comportamento desejado do sistema em termos de cenários. No mundo dos diagramas de caso de uso descrevemos um único cenário para todos os objetos relevantes - comportamento interobjeto. Em contraste, um diagrama de estado descreve todo o comportamento para um único objeto - comportamento intraobjeto. Em [Dou98], Harel denomina esta severa diferença de *grande dualidade do comportamento do sistema.* Estamos longe de um bom algoritmo para verificar essa dualidade. Não se sabe ainda como derivar uma visão da outra, ou como testar eficientemente se as descrições presentes nas duas visões são mutuamente consistentes [Dou98].

Capítulo 4

Controle de Concorrência Semântico para SGBD-TR

4.1 Introdução

SGBD são projetados para gerenciar dados persistentes que são compartilhados entre transações concorrentes [BLS97]. O mecanismo de controle de concorrência desses sistemas coordena as ações das transações de forma que elas possam ser processadas concorrentemente enquanto mantém a consistência lógica do banco de dados.

A maioria das técnicas de controle de concorrência tradicionais baseia-se no conceito de *seriação*, para determinar quais as transações que podem executar concorrentemente. Um escalonamento de transações é *seriável* se ele produz o mesmo resultado de um escalonamento serial. Essas técnicas asseguram as propriedades ACID das transações [EN94]. Essas propriedades impõem restrições em como as transações podem ser executadas (restrições de *consistência lógica das transações*), e em como os dados podem ser acessados (restrições de *consistência lógica dos dados*).

Em BDTR, os aspectos temporais dos dados e as restrições de tempo das transações também devem ser considerados. Portanto, uma técnica de controle de concorrência para um SGBD-TR deve buscar manter a *consistência temporal dos dados e transações* [DiP95b]. O requisito de consistência temporal das transações restringe quando uma transação pode executar através da inclusão de restrições temporais tais como prazos, tempo de início mais cedo, tempo de início mais tarde, entre outras. Baseado nestas

restrições, pode-se designar *prioridades* de execução para as transações e o escalonador deve usar essas prioridades para escalonar as transações para execução. O requisito de consistência temporal dos dados restringe quão velho um item de dado pode ser e ainda ser considerado válido. Então a técnica de controle de concorrência de um SGBD-TR deve buscar manter as quatro formas de consistência em um banco de dados: consistência lógica e temporal dos dados e consistência lógica e temporal das transações.

Infelizmente, existem alguns *conflitos* entre restrições lógicas e temporais que dificultam a manutenção das quatro formas de consistência simultaneamente. Por exemplo, para manter a consistência temporal dos dados, uma transação que atualiza um item de dados pode interromper outra transação que esta lendo o mesmo item de dados. Se o item de dados em questão perdeu sua validade temporal, é interessante permitir que a sua consistência temporal seja restabelecida através da execução da transação de atualização. No entanto, esta execução poderia violar a consistência lógica do dado ou da transação de leitura. Então, existe a necessidade de se negociar entre manter consistência temporal ou consistência lógica. Se a consistência lógica for escolhida, então existe a possibilidade que o item de dados torne-se temporalmente inválido, ou que uma transação viole sua restrição temporal. Se por outro lado, consistência temporal for escolhida, a consistência lógica do dado ou da transação pode ser comprometida.

Outro problema que pode ocorrer devido ao conflito entre restrições lógicas e temporais é a *inversão de prioridade*. Isso ocorre quando uma transação é bloqueada por uma transação de menor prioridade porque a transação de menor prioridade mantém bloqueados dados que a transação de maior prioridade está esperando. As restrições de consistência temporal das transações impõem a ordenação da prioridade, mas as restrições de consistência lógica das transações requerem o bloqueio da transação de maior prioridade pela transação de menor prioridade.

Para expressar a negociação entre consistência lógica e temporal, uma técnica de controle de concorrência deve ser capaz de usar conhecimento sobre a aplicação para determinar quais as transações que podem ser executadas concorrentemente [DiP95b]. Tal técnica, denominada *controle de concorrência semântico*, permite maior concorrência em um banco de dados em tempo-real, embora uma sobrecarga seja imposta ao projetista do sistema. O projetista deve conhecer profundamente todas as transações do sistema, para que ele possa definir as compatibilidades entre a execução das transações.

Geralmente, nessas técnicas as propriedades ACID das transações são relaxadas.

Como discutido em [BLS97], transações em tempo-real não precisam ser seriáveis, especialmente transações de atualização que gravam informação sobre o mundo real no banco de dados. No entanto, a conseqüência de relaxar seriação é que *imprecisão* pode ser acumulada no banco de dados e na visão do banco de dados pelas transações. No exemplo anterior, uma transação de atualização interrompe uma transação de leitura de modo a manter a consistência temporal dos dados. Nesse caso, a transação de leitura pode ter uma visão imprecisa do dado porque ela pode ler um valor escrito por uma transação de atualização não comprometida. O valor de um item de dados resultante de um escalonamento de transações é *impreciso* se ele é diferente do valor correspondente resultante de cada possível escalonamento serial das mesmas transações.

4.2 Técnica de Controle de Concorrência Semântico

Uma técnica de controle de concorrência semântico utiliza conhecimento específico sobre uma aplicação de forma a aumentar a concorrência entre as transações de um de banco de dados em tempo-real.

Em [DiP95b] apresenta-se uma técnica de controle de concorrência semântico orientada a objetos, denominada *Técnica de Bloqueio Semântico*, que suporta consistência lógica e temporal dos dados e transações e permite a negociação entre elas. A técnica também controla a imprecisão resultante dessa negociação. Nessa técnica, o controle de concorrência é distribuído entre os vários objetos, e para cada objeto defini-se uma *função de compatibilidade* (FC), que controla o acesso concorrente aos seus métodos. A FC é apresentada com detalhes na Seção 4.4.

Essa técnica suporta consistência lógica de dados e transações através da definição de funções similares às técnicas de controle de concorrência para banco de dados convencionais. Ela suporta consistência temporal dos dados por permitir a negociação entre consistência lógica e consistência temporal e suporta consistência temporal das transações por permitir a especificação de escalonamentos mais flexíveis que aqueles permitidos por seriação. Além disso, essa técnica pode especificar e limitar alguma imprecisão (definida no contexto de *epsilon serializability* [RP95], apresentado na Seção 4.3.1) que possa surgir no sistema devido ao relaxamento da seriação.

Nos últimos anos, muitas outras técnicas para controle de concorrência semântico foram propostas [WA92; YWLS94]. No entanto nenhuma delas suporta a consistência temporal dos dados e a negociação entre consistência lógica e temporal, tal como a técnica de bloqueio semântico, apresentada na Seção 4.5.

4.3 Imprecisão

Em sistemas em tempo-real, muitas vezes resultados imprecisos podem ser aceitáveis para permitir a satisfação das restrições temporais das transações. Isso significa que as técnicas de controle de concorrência para tais sistemas precisam controlar e limitar essa imprecisão. Muitas das técnicas para controle de concorrência são baseadas no critério de corretude *epsilon Serializability* [RP95], apresentado a seguir.

4.3.1 Seriação *Epsilon (Epsilon Serializability - ES)*

ES [RP95] é uma forma de critério de corretude formal que especifica que um escalonamento para transações é correto se o resultado do escalonamento (ambos, valores dos dados e valores dos argumentos de retorno de transações) está dentro de limites especificados, ou seja, ele generaliza seriação por permitir imprecisão limitada no processamento de transações. Para acumular e limitar imprecisão, este critério assume usar apenas dados mensuráveis.

Uma transação especifica a quantidade de imprecisão que ela pode importar e exportar em um item de dado. A quantidade máxima de imprecisão que uma transação t pode importar para um item x é definida como $lim\_imp_{t,x}$, e a quantidade máxima de imprecisão que uma transação t pode exportar para um item x é definida como $lim\_exp_{t,x}$. Para cada item de dados x no banco de dados, lim_max_x é definido como o limite máximo de imprecisão que pode ser escrito em x.

A quantidade de imprecisão importada e exportada por cada transação, assim como as imprecisões escritas nos itens de dados, devem ser acumuladas durante a execução da transação. A quantidade de imprecisão importada pela transação t para um item de dados x é definida por $imprec\_imp_{t,x}$, e a quantidade de imprecisão exportada pela transação t para um item de dados x é definida por $imprec\_exp_{t,x}$. A quantidade de

imprecisão escrita em um item de dados x é definida como $quant\_imp_x$.

As propriedades de segurança definidas para as transações e os dados especificam os limites de imprecisão para transações e dados. Dado um item de dados x, a seguinte propriedade define um dado *seguro*:

$$x : quant\_imp_x \leq lim\_max_x$$

Essa propriedade indica que a imprecisão no item de dados x é aceitável, se ela não for maior que o limite especificado.

Dada uma transação t e um item de dados x, as seguintes propriedades definem uma transação *segura*:

$$(t, x)_{leitura} : imprec\_imp_{t,x} \leq lim\_imp_{t,x}$$

$$(t, x)_{escrita} : imprec\_exp_{t,x} \leq lim\_exp_{t,x}$$

Essa propriedade indica que a imprecisão nos itens de dados lidos ou escritos por uma transação é aceitável se eles não forem maiores que os limites especificados para eles.

Assim, ES é garantido se e somente se todos os dados e transações são seguros.

As definições acima são aplicáveis a sistemas baseados em transações. Essas definições foram adaptadas para sistemas orientados a objetos e são definidas na Seção 4.3.3. Na Seção 4.3.2 defini-se objeto como utilizado no contexto deste trabalho.

4.3.2 Objeto

Um objeto é uma quíntupla $\langle N, AT, MT, R, FC \rangle$, onde:

1. N é um identificador único para um objeto.

2. AT é um conjunto de atributos, e:
   $\forall a \in AT, a = \langle N, VAL, TEMPO, AVI, QUANT\_IMP \rangle$, onde:

   N é o nome do atributo;

VAL é o valor do atributo;

$TEMPO$ é o rótulo de tempo, que determina quando o valor do atributo foi gravado;

AVI é o intervalo de validade absoluta para o atributo, ou seja, determina por quanto tempo o valor do atributo é válido;

$QUANT\_IMP$ é a imprecisão acumulada para o atributo.

3. MT é um conjunto de métodos, e $\forall mt_i \in MT, mt_i$ possui um conjunto de argumentos Arg_e e Arg_r, onde:

   $\forall Arg_e = \langle (N, VAL, TEMPO, AVI, QUANT\_IMP), LIM\_EXP_e \rangle$, onde N, VAL, $TEMPO$, AVI, e $QUANT\_IMP$ são como definidos para AT, e LIM_EXP_e especifica o limite máximo de imprecisão que pode ser passado pelo método.

   $\forall Arg_r = \langle (N, VAL, TEMPO, AVI, QUANT\_IMP), LIM\_IMP_r \rangle$, onde N, VAL, $TEMPO$, AVI, e $QUANT\_IMP$ são como definidos para AT, e LIM_IMP_r especifica o limite máximo de imprecisão permitido no valor retornado.

4. R é um conjunto de restrições lógicas e temporais que definem o estado correto de uma instância de um objeto. Uma restrição é definida como um predicado que pode incluir os campos VAL, $TEMPO$, AVI e $QUANT\_IMP$, de um atributo. Através destes predicados declaram-se as restrições lógicas e temporais, bem como os limites de imprecisão para os atributos.

5. FC é uma função de compatibilidade que utiliza informação semântica sobre os objetos e o estado do sistema para determinar quando dois métodos de um objeto podem ser executados concorrentemente, sendo definida por $FC(m_i, m_j) =$*(expressão booleana)*$\Rightarrow IA$, onde m_i é o método que está executando, m_j é o método que foi invocado e *(expressão booleana)* pode conter predicados envolvendo várias características do objeto ou do sistema, tais como tempo atual e o intervalo de validade absoluto AVI do atributo, limite máximo de imprecisão LIM_IMP e a imprecisão acumulada $QUANT\_IMP$ do atributo, entre outras. IA é definida por uma

expressão que calcula a imprecisão acumulada nos atributos e nos argumentos dos métodos. Na Seção 4.4, a função de compatibilidade é apresentada com detalhes.

4.3.3 Seriação *Epsilon* Orientada a Objetos (SEOO)

Em modelos orientados a objetos os dados são representados por objetos. Um objeto é dito *seguro* se obedece a seguinte propriedade:

$$objeto_o : \forall a \in o_A(a.quant\_imp \leq lim\_max_a)$$

onde o_A é o conjunto de atributos de o e lim_max_a é o limite máximo de imprecisão que pode ser escrito em a.

Assim, se cada atributo em um objeto satisfaz a imprecisão especificada, então o objeto é *seguro*.

Em modelos orientados a objetos as transações operam sobre os objetos através dos métodos dos objetos. Os valores dos itens de dados são obtidos dos objetos através dos argumentos de retorno dos métodos, e são passados para os objetos através dos argumentos de entrada dos métodos. Seja t_{MI} o conjunto de métodos invocados por uma transação t e o_M o conjunto de métodos em um objeto o. Seja $t_{MI} \sqcap o_M$, os métodos de o invocados por t. Uma transação t é *segura* em relação a um objeto o se obedece as seguintes propriedades:

$$(t, o)_{leitura} : \forall m \in (t_{MI} \sqcap o_M) \forall r \in arg\_retorno(m)(r.quant\_imp \leq lim\_imp_r)$$

$$(t, o)_{escrita} : \forall m \in (t_{MI} \sqcap o_M) \forall e \in arg\_entrada(m)(e.quant\_imp \leq lim\_exp_e)$$

Esta propriedade indica que se os argumentos dos métodos invocados por t em um objeto o estão dentro de seus limites de imprecisão, então t é segura para o.

Assim, SEOO é garantido se e somente se todos os objetos e transações dos objetos são seguros.

4.4 Função de Compatibilidade

Uma função de compatibilidade (FC) é um componente de um objeto, definido pelo projetista do sistema, para expressar compatibilidade entre a execução concorrente dos métodos de um objeto. Ela é avaliada em tempo de execução e definida sobre cada par ordenado de métodos de um objeto. Uma FC tem a forma:

$$FC(m_{ati}, m_{inv}) = Expressão\ Booleana \Rightarrow IA$$

onde m_{ati} representa o método que está sendo executado e m_{inv} representa o método que foi invocado. A expressão booleana pode conter predicados envolvendo valores dos argumentos dos métodos, do banco de dados e do sistema em geral. IA é definida por uma expressão que calcula a imprecisão acumulada nos atributos e nos argumentos dos métodos.

O conceito de *conjunto afetado* foi introduzido em [BR88] e é usado como uma base para representar o conjunto de atributos de um objeto que um método lê ou escreve. Cada método m tem um *conjunto de leitura afetado (CLA(m))*, isto é, o conjunto de atributos do objeto que serão lidos pelo método, e um *conjunto de escrita afetado (CEA(m))*, isto é, o conjunto de atributos do objeto que serão escritos pelo método.

Uma FC pode ser utilizada pelo projetista para negociar entre a manutenção de consistência lógica e consistência temporal, e controlar a quantidade de imprecisão aceitável. Portanto, além de expressar compatibilidade entre a execução de dois métodos, uma FC também expressa informação sobre a imprecisão máxima que pode ser introduzida com a execução concorrente de métodos. Existem três origens potenciais de imprecisão que uma FC pode expressar para invocações de métodos concorrentes m_{ati} e m_{inv} [DiP95b]:

Imprecisão no valor de um atributo que está no conjunto de escrita afetado (CEA) de m_{ati} e m_{inv}, isto é, m_{ati} e m_{inv} escrevem o mesmo atributo.

Imprecisão no valor do argumento de retorno de m_{ati}, quando m_{ati} lê atributos escritos por m_{inv}.

Imprecisão no valor do argumento de retorno de m_{inv}, quando m_{inv} lê atributos escritos por m_{ati}.

Para ilustrar, apresentaremos algumas funções de compatibilidade que expressam compatibilidade condicional de execução de métodos.

Exemplo 1

A função de compatibilidade 4.1 expressa uma negociação de consistência lógica por consistência temporal quando o método *LerTemp* (Ler Temperatura) está sendo executado e o método *AtuTemp* (Atualizar Temperatura) é invocado. Sob seriação, estes métodos não poderiam ser executados concorrentemente, pois a visão do atributo *temp* pelo método *LerTemp* seria prejudicada. No entanto, se a validade temporal do atributo *temp* é violada, é importante permitir a execução do método *AtuTemp* para restaurar sua consistência. Porém, os dois métodos só podem ser executados concorrentemente se o valor do atributo *temp* escrito por *AtuTemp* ($A.val$) é próximo do valor corrente do atributo *temp* ($temp.val$). Esta determinação é baseada no valor corrente do atributo *temp* ($temp.val$), no limite máximo de imprecisão permitido para o argumento de retorno L de *LerTemp* (lim_imp_L), na quantidade de imprecisão que *AtuTemp* irá escrever em *temp* através de A ($A.quant\_imp$) e na quantidade de imprecisão existente no argumento de retorno ($L.quant\_imp$). A imprecisão acumulada, resultante da execução concorrente dos dois métodos também é mostrada. Nesse caso, o argumento de retorno L do método *LerTemp* tem um aumento de imprecisão igual à diferença entre o valor do atributo *temp* antes da atualização ($temp.val$) e o novo valor do atributo após a atualização ($A.val$), mais a quantidade de imprecisão que é escrita no atributo *temp* por *AtuTemp*, ($A.quant\_imp$).

$$
\begin{aligned}
FC(LerTemp(L), AtuTemp(A)) = {} & (agora - temp.tempo > temp.avi) \\
& \wedge (|temp.val - A.val| \leq (lim\_imp_L \\
& - (L.quant\_imp + A.quant\_imp))) \\
& \Rightarrow L.quant\_imp = L.quant\_imp \\
& + A.quant\_imp + |temp.val - A.val|
\end{aligned}
\tag{4.1}
$$

Exemplo 2

Na função de compatibilidade 4.2, o método *AtuTemp* atualiza o valor do atributo *temp* com o valor do argumento de entrada A e o método *LerTemp* lê o valor do atributo *temp*. Esses métodos só poderão ser executados concorrentemente se a diferença entre o novo valor da temperatura ($A.val$) e o valor original ($temp.val$) estiver dentro do limite de imprecisão aceito pelo argumento de retorno L do método *LerTemp* (lim_imp_L). Essa determinação é baseada no valor atualizado por *AtuTemp* ($A.val$), no limite de imprecisão permitido para o argumento de retorno de *LerTemp* (lim_imp_L), e na quantidade de imprecisão acumulada no argumento de retorno de *LerTemp* ($L.quant\_imp$). A imprecisão acumulada no argumento de retorno L do método *LerTemp* tem um aumento igual à diferença entre o valor original ($temp.val$) e o novo valor ($A.val$), caso os métodos sejam executados concorrentemente.

$$FC(AtuTemp(A), LerTemp(L)) = |temp.val - A.val| \leq lim\_imp_L - L.quant\_imp) \Rightarrow L.quant\_imp = L.quant\_imp + |temp.val - A.val| \tag{4.2}$$

Exemplo 3

A função de compatibilidade 4.3 mostra como um atributo pode tornar-se impreciso. Duas invocações de um mesmo método *AtuTemp* podem ocorrer concorrentemente se um sensor atualiza o atributo *temp* e um operador humano também solicita a execução de uma operação de escrita nesse mesmo atributo. A função de compatibilidade indica que duas invocações do método *AtuTemp* podem ser executadas concorrentemente desde que as diferenças entre os valores escritos quando executando as duas invocações não excedam a quantidade de imprecisão permitida para *temp*. A determinação de tal imprecisão é baseada nos valores escritos pelas duas invocações do método ($A_1.val$ e $A_2.val$), na imprecisão existente em *temp* ($temp.quant\_imp$) e no limite de imprecisão permitido para *temp* (lim_max_{temp}). A imprecisão acumulada no atributo *temp* tem um acréscimo igual a diferença entre os valores escritos pelas duas invocações, se os métodos forem executados concorrentemente.

$$
\begin{aligned}
FC(AtuTemp_1(A_1), AtuTemp_2(A_2)) = (&|A_1.val - A_2.val| \\
&\leqslant (lim\_max_{temp} - temp.quant\_imp)) \\
&\Rightarrow temp.quant\_imp = temp.quant\_imp \\
&+ |A_1.val - A_2.val|
\end{aligned}
\tag{4.3}
$$

4.5 Mecanismo de Bloqueio Semântico

O mecanismo de bloqueio semântico apresentado em [DiP95b] trata com as ações de *requisição de bloqueio semântico*, *requisição de invocação de método* e *requisição de liberação* de métodos.

Uma transação pode requisitar um bloqueio semântico antes de requisitar uma invocação de um método, isto é, para uma futura requisição de invocação de método, ou simultaneamente à invocação do método. O primeiro caso pode ser útil quando a transação requer que todos os bloqueios sejam concedidos antes de executar algum método.

Os algoritmos para *requisição de bloqueio semântico* e *requisição de invocação de método* desse mecanismo são apresentados nas Seções 4.5.1 e 4.5.2, respectivamente.

4.5.1 Requisição de Bloqueio Semântico

Algoritmo 4.1 Requisição de Bloqueio Semântico para m_{inv}

```
1: Concedido ← VERDADEIRO
2: para todo ((m_ati ∈ BloqueiosAtivos) ∨ ((m_ati ∈ Fila) ∧ (m_ati.prio ⩾ m_inv.prio)))
   faça
3:    se FC(m_ati, m_inv) então
4:       Incrementa imprecisão
5:    senão
6:       Concedido ← FALSO
7:    fim se
8: fim para
9: se Concedido = FALSE então
10:   Enfileire(m_inv) em Fila
11: senão
12:   Adicione m_inv a BloqueiosAtivos
13: fim se
```

Dada uma requisição de bloqueio semântico para m_{inv}, inicialmente a função de compatibilidade $FC(m_{ati}, m_{inv})$ é avaliada para verificar se m_{inv} é compatível com todos os bloqueios ativos (*BloqueiosAtivos*) e com as requisições de bloqueio que estão na fila de prioridade (*Fila*) com prioridade maior que m_{inv} (Passo 3). A fila de prioridade (*Fila*) é mantida para guardar todas as requisições que não podem ser concedidas imediatamente. Quando a FC é avaliada para *verdadeira*, o mecanismo acumula a imprecisão introduzida com a execução concorrente dos métodos m_{ati} e m_{inv} (Passo 4).

Observe que uma requisição de bloqueio semântico para uma invocação futura de método não contém valores para os argumentos no momento da requisição. Os valores para os argumentos só serão passados no momento da invocação do método. Portanto, quando a função de compatibilidade $FC(m_{ati}, m_{inv})$ é avaliada, se m_{ati} ou m_{inv} são para uma futura invocação de método, então qualquer cláusula da FC que envolve os argumentos do método deve ser avaliada para *falso*.

Quando todos os testes para a função de compatibilidade são avaliados para verdadeiro, m_{inv} é colocado na lista de bloqueios ativos (Passo 12). Quando algum teste falha, m_{inv} é colocado na fila de prioridade (*Fila*) para ser testado novamente quando algum bloqueio for liberado (Passo 10).

4.5.2 Requisição de Invocação de Método

Dada uma requisição de invocação de método m_{inv}, o mecanismo inicialmente computa a imprecisão que m_{inv} introduzirá nos atributos que ele escreve (se m_{inv} é um método de escrita) ou em seus argumentos de retorno (se m_{inv} é um método de leitura).

O procedimento de imprecisão inicial (Passo 1) computa a quantidade de imprecisão que m_{inv} escreverá em cada atributo a ($m_{inv}.ImpEscrita(a)$) e que m_{inv} retornará através de cada argumento de retorno r ($m_{inv}.ImpLeitura(r)$). Esses valores são computados usando as quantidades de imprecisão existentes nos atributos ou argumentos de retorno e calculando como o método atualizará esta imprecisão. Este procedimento pode ser criado pelo projetista do objeto ou por uma ferramenta que examina a estrutura de m_{inv} para determinar como o método irá afetar a imprecisão dos atributos que ele escreve ou retorna.

No segundo passo (Passo 2) verificam-se as pré-condições que determinam se a exe-

Algoritmo 4.2 Requisição de Invocação de Método m_{inv}

```
1: ImprecisãoInicial(m_inv)
2: se alguma Precondição falha então
3:    Enfileire m_inv em Fila
4: senão
5:    para todo a ∈ CEA(m_inv) faça
6:       Armazena o valor original de a.quant_imp
7:       a.quant_imp ← m_inv.ImpEscrita(a)
8:    fim para
9:    para todo r ∈ ArgsRetorno(m_inv) faça
10:      Armazena o valor original r.quant_imp
11:      r.quant_imp ← m_inv.ImpLeitura(r)
12:   fim para
13:   se já bloqueado então
14:      Permite a execução de m_inv
15:      Atualiza o Bloqueio Semântico
16:      Verifica Fila
17:   senão
18:      Requisita Bloqueio Semântico
19:      se bloqueio concedido então
20:         Permita a execução de m_inv
21:      senão
22:         para todo a ∈ CEA(m_inv) faça
23:            Recupera o valor original de a.quant_imp
24:         fim para
25:         para todo r ∈ ArgsRetorno(m_inv) faça
26:            Recupera o valor original de r.quant_imp
27:         fim para
28:         para todo argumentos de retorno r de uma invocação de método ativa faça
29:            Recupera o valor original de r.quant_imp
30:         fim para
31:      fim se
32:   fim se
33: fim se
34: Libera Bloqueio
35: Remove m_inv de BloqueiosAtivos
36: Verifica Fila
```

cução de m_{inv} poderia violar a consistência temporal ou limites de imprecisão. As pré-condições são as seguintes:

$$m_{inv}.temporal \Rightarrow \forall a \in CLA(m_{inv}), Agora - a.tempo - Exec(m_{inv}) \leq a.avi \quad \text{(a)}$$

$$\forall a \in CEA(m_{inv})(m_{inv}.ImpEscrita(a)) \leq lim\_max_a \quad \text{(b)}$$

$$\forall r \in ArgsRetorno(m_{inv})(m_{inv}.ImpLeitura(r)) \leq lim\_imp_r) \quad \text{(c)}$$

A pré-condição (a) assegura que se uma transação requer dados temporalmente válidos, então o método invocado só será executado se os dados que ele lê não perdem sua validade temporal durante a execução do método. A pré-condição (b) assegura que um método de escrita só pode executar se ele não introduz imprecisão além da permitida nos atributos que ele escreve. A pré-condição (c) assegura que um método de leitura só executa se ele não introduz imprecisão além da permitida em seus argumentos de retorno.

Se alguma pré-condição falha, m_{inv} é colocado na fila de prioridade ($Fila$) para ser testado novamente quando um bloqueio for liberado (Passo 3). Se todas as condições forem satisfeitas e m_{inv} é um método de escrita, a quantidade de imprecisão é atualizada para cada atributo a que m_{inv} escreve com o valor $m_{inv}.ImpEscrita(a)$. Se m_{inv} é um método de leitura, a quantidade de imprecisão é atualizada para cada argumento de retorno r de m_{inv} com o valor $m_{inv}.ImpLeitura(r)$ (Passos de 5 a 12). Observe que os valores originais de imprecisão dos atributos e argumentos de retorno envolvidos são salvos, de forma a permitir que eles possam ser recuperados caso o bloqueio não seja concedido.

Observe também que as pré-condições podem bloquear uma transação se os dados que ela acessa são muito imprecisos para seus requisitos. Então deve existir alguma forma de se recuperar a precisão do dado de forma que a transação não fique bloqueada definitivamente. Uma forma é criar transações, que podem ser de um sensor ou de um usuário, para escreverem dados precisos nos dados. Assim as transações bloqueadas pela imprecisão dos dados podem ser executadas.

No passo seguinte (Passo 13), verifica-se se m_{inv} já foi ou não bloqueado. Se não, o bloqueio semântico é requisitado, como descrito na Seção 4.5.1 (Passo 18). Caso o bloqueio seja concedido, a execução do método é permitida (Passo 20). Caso contrário,

os valores originais de imprecisão que foram trocados são recuperados (Passo 22). Se o bloqueio para m_{inv} foi concedido antes de sua invocação (bloqueio para invocação futura), a execução do método é permitida (Passo 14). Em seguida o mecanismo executa um procedimento de atualização de bloqueio semântico (Passo 15). Este procedimento atualiza o bloqueio associado com m_{inv}, com os respectivos argumentos. Estes argumentos não estavam disponíveis no momento do bloqueio, como comentado na Seção 4.5.1. Esta atualização é muito importante, pois ela pode aumentar a concorrência entre os métodos, uma vez que uma função de compatibilidade $FC(m_{ati}, m_{inv})$ tem mais chances de ser avaliada para verdadeira quando os argumentos dos métodos estão disponíveis. Após a atualização do bloqueio semântico, as requisições de bloqueio esperando na fila de prioridade (*Fila*) são verificadas para determinar a compatibilidade com o método atualizado (Passo 16).

Finalmente, quando um método termina de executar, o bloqueio correspondente é liberado (Passo 34). Quando um bloqueio é liberado ele é removido da lista de bloqueios ativos (Passo 35) e a fila de prioridade é verificada (Passo 36). Como o bloqueio recentemente liberado pode estar associado com a invocação de um método que recuperou a consistência lógica ou temporal de um atributo, ou o bloqueio pode ter causado alguma incompatibilidade, algumas das requisições da fila de prioridade podem ter agora o bloqueio concedido. Também, as requisições de invocação de método da fila de prioridade podem passar nas pré-condições se a consistência temporal ou precisão foi restaurada para o dado. A fila é verificada em ordem de prioridade e, caso algumas dessas requisições sejam concedidas, elas são removidas da fila de prioridade e adicionada na lista de bloqueios ativos.

4.5.3 Definição da Função de Compatibilidade

Em [DiP95b] são apresentadas duas regras para a definição das funções de compatibilidade utilizadas pela técnica de controle de concorrência semântico. Assim essas regras devem ser seguidas quando se definindo uma FC, com o objetivo de se determinar métodos concorrentes e ao mesmo tempo controlar a imprecisão gerada pela execução desses métodos. Intuitivamente essas regras permitem compatibilidade entre dois métodos, onde um lê e o outro escreve em um mesmo conjunto de atributos, ou ambos escrevem

em um mesmo conjunto de atributos, desde que os limites de imprecisão especificados não sejam violados.

Qualquer imprecisão gerada pela execução concorrente de dois métodos deve ser calculada antes da FC ser avaliada, pelo procedimento de imprecisão inicial (veja Seção 4.5.2).

Sejam a um atributo de um objeto o e m_1 e m_2 dois métodos de o.

Regras para a Função de Compatibilidade

R1: Se $a \in CEA(m_1) \cap CEA(m_2)$, então as funções de compatibilidade para m_1 e m_2, $FC(m_1, m_2)$ e $FC(m_2, m_1)$, devem retornar VERDADEIRO se e somente se elas incluem a cláusula conjuntiva: $|z1 - z2| \leq (lim\_max_a - a.quant\_imp)$, onde $z1$ e $z2$ são os valores escritos em a por m_1 e m_2 respectivamente. Além disto, a função de compatibilidade deve especificar o acúmulo de imprecisão em a, da seguinte forma: $a.quant\_imp = a.quant\_imp + |z1 - z2|$.

R2: Se $a \in CLA(m_1) \cap CEA(m_2)$, então para cada $r \in arg\_retorno(m1)$ seja z o valor corrente de a correspondente ao argumento r do método m_1, seja x o valor escrito em a por m_2, e seja w o valor de r correspondente a x. Então:

a) A função de compatibilidade para m_2 e m_1, $FC(m_2, m_1)$ deve retornar VERDADEIRO se e somente se ela inclui a cláusula conjuntiva: $|z - w| \leq (lim\_imp_r - r.quant\_imp)$. Além disto, a função de compatibilidade deve especificar o acúmulo de imprecisão em r, da seguinte forma: $r.quant\_imp = r.quant\_imp + |z - w|$.

b) A função de compatibilidade para m_1 e m_2, $FC(m_1, m_2)$ deve retornar VERDADEIRO se e somente se ela inclui a cláusula conjuntiva: $|z - w| \leq (lim\_imp_r - (r.quant\_imp + x.quant\_imp))$. Além disto, a função de compatibilidade deve especificar o acúmulo de imprecisão em r, da seguinte forma: $r.quant\_imp = r.quant\_imp + x.quant\_imp + |z - w|$.

A regra R1 indica que se dois métodos m_1 e m_2 são executados concorrentemente, e ambos escrevem em um mesmo atributo a, então a imprecisão que pode ser introduzida em a é no máximo a diferença entre os dois valores que são escritos, $|z1 - z2|$. Para ser

segura, essa diferença não pode ser maior que o limite máximo de imprecisão permitida para a, menos a quantidade de imprecisão corrente em a, $(lim\_max_a - a.quant\_imp)$. A imprecisão acumulada é também refletida em R1.

A função de compatibilidade mostrada em 4.4 mostra uma aplicação da regra R1. Observe que o atributo *temp* está no conjunto afetado do método *AtuTemp*.

$$\begin{aligned} FC(AtuTemp_1(A_1), AtuTemp_2(A_2)) = (&|A_1.val - A_2.val| \leq \\ &(lim\_max_{temp} - temp.quant\_imp)) \\ &\Rightarrow temp.quant\_imp \\ &= temp.quant\_imp + |A_1.val - A_2.val| \end{aligned} \tag{4.4}$$

O valor escrito no atributo *temp* pelo método $AtuTemp_1$ é A_1 e o valor escrito em *temp* pelo método $AtuTemp_2$ é A_2. Então, a função de compatibilidade pode retornar VERDADE apenas se $|A_1.val - A_2.val| \leq (lim\_max_{temp} - temp.quant\_imp)$.

A imprecisão acumulada no atributo *temp* é igual à imprecisão anterior mais a diferença entre A_1 e A_2.

A regra R2 baseia-se no fato de que se dois métodos m_1 e m_2 são executados concorrentemente, e m_1 lê um atributo a e m_2 escreve no mesmo atributo a, então a visão que m_1 tem do atributo a (no argumento de retorno) pode ser imprecisa.

Em R2a, m_2 é o método ativo e m_1 é o método que foi invocado. Nesta regra, a quantidade de imprecisão que pode ser introduzida na visão do atributo em m_1 é no máximo a diferença entre o valor do atributo antes de m_2 escrevê-lo e o valor do atributo após m_2 escrevê-lo $|z - w|$. Essa diferença não pode ser maior que o limite máximo de imprecisão permitida para r (lim_imp_L), menos a quantidade corrente de imprecisão em r $(L.quant\_imp)$. A imprecisão acumulada em r é também refletida em R2a.

A função de compatibilidade 4.5 mostra uma aplicação da regra R2a.

$$\begin{aligned} FC(AtuTemp(A), LerTemp(L)) = &|temp.val - A.val| \leq lim\_imp_L \\ &- L.quant\_imp) \Rightarrow L.quant\_imp \\ &= L.quant\_imp + |temp.val - A.val| \end{aligned} \tag{4.5}$$

Em R2b, m_1 é o método ativo e m_2 é o método que foi invocado. Nesta regra, a quantidade de imprecisão que pode ser introduzida na visão do atributo em m_1 é no máximo

a diferença entre o valor do atributo antes de m_2 escrevê-lo e o valor do atributo após m_2 escrevê-lo $|z - w|$. Essa diferença não pode ser maior que o limite máximo de imprecisão permitido para r (lim_imp_L), menos a quantidade corrente de imprecisão em r, e a quantidade de imprecisão passada pelo argumento de m_2 ($r.quant\_imp + x.quant\_imp$). Observe que neste caso a quantidade de imprecisão que pode ser introduzida em a por m_2 ($x.quant\_imp$) deve ser considerada na função de compatibilidade $FC(m_1, m_2)$. Isso porque o procedimento de imprecisão inicial para m_1 calcula a quantidade de imprecisão que m_1 irá retornar através de r ($m_1.ImpLeitura(r)$) antes de m_2 ser invocado, e então $r.quant\_imp$ não inclui a quantidade de imprecisão que m_2 deve introduzir em a ($x.quant\_imp$). Portanto, esta imprecisão deve ser calculada pela função de compatibilidade. A imprecisão acumulada em r é também refletida em R2b.

A função de compatibilidade mostrada em 4.6 mostra uma aplicação da regra R2b. Observe que a função só será avaliada para VERDADE se a diferença entre o valor do atributo *temp* antes da atualização ($temp.val$) e o valor após a atualização ($A.val$), estiver dentro do limite de imprecisão permitido no argumento de retorno L (lim_imp_L) do método *LerTemp*. O limite de imprecisão permitido deve considerar a quantidade de imprecisão existente no argumento de retorno ($L.quant\_imp$) e a quantidade de imprecisão no argumento usado para atualizar o atributo *temp* ($A.quant\_imp$).

$$\begin{aligned} FC(LerTemp(L), AtuTemp(A)) = (&now - temp.tempo \geqslant temp.avi) \\ &(|temp.val - A.val| \leq (lim\_imp_L \\ &- (L.quant\_imp + A.quant\_imp))) \\ &\Rightarrow L.quant\_imp = L.quant\_imp \\ &+ A.quant\_imp + |temp.val - A.val| \end{aligned} \tag{4.6}$$

Observe que a diferença básica entre R2a e R2b é que em R2a a imprecisão introduzida pelo método de escrita m_2 ($x.quant\_imp$) é considerada pelo procedimento inicial, pois o método de leitura m_1 é invocado depois do método de escrita m_2, e portanto este tem conhecimento do valor a ser escrito em a por m_2. Então, a FC em R2a não inclui a imprecisão introduzida em a ($x.quant\_imp$) pelo método *AtuTemp*, já que esta é considerada pelo procedimento que calcula a imprecisão inicial.

Em [DiP95b] prova-se que estas restrições são suficientes para garantir a seriação

epsilon [RP95]) e, portanto, controlam a imprecisão.

4.6 Conclusão

Neste capítulo apresentou-se uma técnica para controle de concorrência semântico para bancos de dados em tempo-real que suporta consistência lógica e consistência temporal, assim como limita a imprecisão resultante da sua negociação.

Nessa técnica o controle de concorrência é distribuído entre os objetos, ou seja, para cada objeto define-se um conjunto de funções de compatibilidade para controlar a execução concorrente dos métodos. Uma função de compatibilidade especifica quando sacrificar consistência lógica por consistência temporal e o acúmulo e limites de alguma imprecisão lógica.

Apresentou-se também as regras para definir uma função de compatibilidade. A técnica apresentada pode preservar a consistência global e limitar a imprecisão por garantir uma forma de seriação epsilon especializada para bancos de dados em tempo-real orientados a objetos.

Capítulo 5

Introdução e Conceitos Básicos de Redes de Petri e ECPN

Neste capítulo apresentam-se os conceitos básicos relativos a redes de Petri clássicas e a um novo tipo de redes de Petri baseadas em objetos, denominadas ECPN (*Extended Coloured Petri Nets*), que foram desenvolvidas nesta tese. As ECPN serão utilizadas como base formal do método de modelagem desenvolvido nesta tese. Os conceitos serão introduzidos de forma informal. Os aspectos formais serão introduzidos no Capítulo 6.

5.1 Introdução ao Modelo de Redes de Petri Clássicas

Redes de Petri são uma ferramenta matemática amplamente utilizada para a descrição e o estudo de sistemas de processamento de informação que se caracterizam como concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e/ou estocásticos [Mur89]. Além de serem um formalismo matemático, as redes de Petri são uma ferramenta de expressão gráfica e simulação. A faceta gráfica das redes de Petri possibilita a melhor interação entre diferentes indivíduos, reduzindo problemas de comunicação. Como as definições semânticas também são matemáticas, torna-se possível a simulação do comportamento de sistemas modelados por redes de Petri, o que permite a visualização da dinâmica do sistema por parte dos desenvolvedores e usuários do sistema, de modo a reforçar a confiabilidade nos requisitos e na especificação.

A estrutura de uma rede de Petri é um grafo bipartido, direcionado e com pesos. O

termo bipartido indica que os nós do grafo pertencem a duas classes disjuntas, denominadas *lugares* e *transições*, e que os arcos ligam sempre elementos de classes distintas. O termo direcionado é usado para denotar que os arcos têm origem e destino especificados. Os arcos podem ter pesos associados, o que justifica a qualificação *com pesos*. Os pesos dos arcos são valores inteiros positivos que podem ser interpretados como replicações dos arcos. Assim, um arco com peso w denota a existência de w arcos entre os mesmos nós.

Os lugares são convencionalmente elementos passivos nas redes de Petri, ou seja, lugares podem armazenar, mas não podem transformar informações. Em geral eles são utilizados para representar condições, recursos, dados, etc. As transições são os únicos elementos convencionalmente ativos nas redes de Petri. Isso significa que transições podem transformar, mas nunca armazenar informações. Elas são utilizadas para representar ações, atividades, transformações, eventos, etc.

Graficamente, lugares são representados por círculos e transições por retângulos. Um lugar p é entrada para uma transição t se existe um arco direcionado conectando o lugar à transição, neste caso o lugar é um *lugar de entrada*. Um lugar p é saída para uma transição t se existe um arco direcionado conectando a transição ao lugar, neste caso o lugar é um *lugar de saída*.

O grafo direcionado define a estrutura de um sistema representado por uma rede de Petri. A estrutura de um sistema é também denominado *grafo de suporte* ou *estrutura da rede*.

A descrição anterior, no entanto, estabelece apenas a natureza sintática de uma rede de Petri. Como o principal objetivo das redes de Petri é a formalização de aspectos dinâmicos de sistemas, é preciso então que seja possível expressar estados de um sistema. Para definir matematicamente o comportamento de uma rede de Petri novos conceitos precisam ser adicionados. Com esse objetivo, associam-se marcas à estrutura da rede de Petri que denotam o estado em que se encontra o sistema. As marcas são denominadas *fichas* e podem ser associadas apenas aos lugares da rede. As fichas são representadas graficamente por pontos pretos. As fichas são utilizadas para marcar os lugares e determinar o estado em que se encontra o sistema. O estado de uma rede de Petri é denominado *marcação*, de forma que podemos dizer que uma marcação é caracterizada pela distribuição de fichas nos lugares da rede em determinado instante. Assim, uma

marcação é um vetor com o mesmo número de lugares que a estrutura da rede.

O comportamento do sistema modelado pela estrutura da rede pode ser caracterizado pelo movimento de fichas pelos lugares, quando a rede é *executada*. Este movimento de fichas caracteriza o comportamento dinâmico do sistema em termos de estados e suas mudanças. Uma transição deve estar *habilitada* na marcação corrente para poder ocorrer. Uma transição é dita habilitada se todos os lugares de entrada são marcados por, pelo menos, o mesmo número de fichas definido pelo peso associado aos arcos conectando esses lugares à transição. Uma transição habilitada pode ocorrer. Quando uma transição ocorre, o número de fichas associadas aos pesos dos arcos de entrada são removidas dos lugares de entrada, e depositadas nos lugares de saída de acordo com os pesos associados aos arcos saindo da transição, conectando os lugares de saída. Este movimento de fichas pela rede é também conhecido como *jogo de fichas*.

Deste modo a definição de uma rede de Petri clássica pode ser dada pela união de uma estrutura de rede de Petri (um grafo) a uma marcação inicial (um estado).

5.2 Descrição de Sistemas Através de Redes de Petri Clássicas

Em geral, a descrição de sistemas complexos reais através de redes de Petri clássicas torna-se extremamente extensa em termos da estrutura de rede. Percebe-se também que subestruturas semelhantes do modelo precisam ser repetidas, caracterizando certa redundância de expressão. Além disso, as redes clássicas não favorecem a modelagem de aspectos temporais quantitativos dos sistemas, permitindo apenas estabelecer relações de dependência de eventos em termos de expressões temporais qualitativas.

Diversas extensões foram propostas às redes clássicas com o intuito de sanar as limitações citadas. Uma classe de extensões de redes de Petri agrega a possibilidade de tratamento quantitativo de aspectos temporais dos sistemas. Em geral, as ferramentas permitem a associação de atributos de tempo à semântica do modelo. Assim, torna-se possível especificar além da possibilidade da ocorrência de eventos num sistema, as limitações temporais às quais os eventos são associados. A classe de redes de Petri temporal é de especial interesse para as atividades de verificação e avaliação de desempenho de

sistemas de software [PdFC94].

Uma outra classe de extensões de redes de Petri, denominadas redes de Petri de alto-nível, introduz diversos mecanismos para a simplificação dos modelos obtidos, através do enriquecimento do poder de expressão. A principal contribuição deste enfoque é a integração das redes de Petri à Teoria dos Tipos de Dados. Em redes de Petri de alto-nível, as fichas que denotam o estado do sistema podem representar tipos estruturados de dados, ao invés da simples indicação binária que a presença da ficha representa nas redes clássicas. Isto permite que os modelos sejam simplificados através da fatoração de subestruturas semelhantes, que agora poderão reconhecer as fichas sobre as quais atuam. As abordagens de redes de Petri de alto-nível que se destacam são as redes Predicado/Transição (PrT-Nets) [Gen91; Gen89] e as redes de Petri Coloridas (CPN) [Jen87; Jen92; Jen97; Jen98b].

Entretanto, apesar da introdução das redes de Petri de alto-nível, a especificação e a modelagem de sistemas complexos pode ainda ser uma tarefa difícil. A principal limitação remanescente é a falta de mecanismos que permitam especificar a modularidade dos sistemas. Assim, especificações e modelos de sistemas complexos elaborados através de redes de Petri de alto-nível têm aspecto essencialmente *plano* onde toda a estrutura faz parte de um único módulo, dado que a natureza dos paradigmas não permite identificar subestruturas do sistema.

Nos últimos anos, um novo paradigma emergiu como alternativa às atividades de desenvolvimento de sistemas de software: a orientação a objetos. O paradigma é baseado em conceitos simples como classes, objetos, atributos, encapsulamento, herança e agregação. A principal razão do seu sucesso e crescente uso é que ele pode prover um sólido esquema de estruturação e controle de complexidade para sistemas de software.

5.3 Introdução Informal a Redes de Petri Coloridas

Nesta seção apresentam-se os conceitos básicos das redes de Petri coloridas (*CPN - Coloured Petri Nets*). As CPNs são uma classe de redes de Petri de alto-nível aplicável à especificação, simulação, validação e implementação de sistemas complexos de software [Jen92]. Sua ampla utilização deve-se essencialmente à clareza do sólido embasamento matemático, aos métodos formais de análise e verificação e à existência de ferramen-

tas de software já plenamente desenvolvidas. Uma CPN é composta essencialmente por uma estrutura, um conjunto de inscrições e um conjunto de declarações. A estrutura de uma CPN é semelhante à estrutura das redes Lugar/Transição. Portanto, é também um grafo dirigido e bipartido. Entretanto, ao invés de pesos inteiros, os arcos são associados a inscrições que são expressões que determinam dinamicamente quantas e quais fichas podem ser removidas ou adicionadas aos lugares associados, na ocorrência de uma transição. O estado inicial de uma CPN também é determinado por inscrições associadas aos lugares. Cada inscrição é, em geral, uma expressão construída a partir de constantes, variáveis e operadores previamente definidos. O conjunto de declarações de uma CPN serve primordialmente para declarar a natureza dos elementos citados nas diversas inscrições e, por conseguinte, dos objetos manipulados. As inscrições e declarações de uma CPN podem, a priori, ser escritas em praticamente qualquer linguagem que tenha sintaxe e semântica bem definidas. Pode-se, por exemplo, usar a notação matemática padrão, embora isso seja de certa forma inconveniente, devido a problemas gerados pelo uso de símbolos. Em geral, CPN vêm sendo utilizadas em associação com uma linguagem denominada CPN-ML [Ja96], derivada da linguagem funcional Standard ML [Ull93] e mais recentemente em Standard ML'97 [Ull98; Ja99], cuja sintaxe é bastante semelhante à usada por linguagens de programação convencionais.

Em um modelo descrito com CPN pode ser ainda utilizado um outro tipo de inscrição denominada de *guarda*. As guardas são associadas às transições e têm a função de restringir as condições de sua ocorrência. Devido à sua função, as guardas devem ser expressões de natureza booleana, isto é, ao serem avaliadas devem resultar exclusivamente em valores do conjunto verdade. As guardas são representadas graficamente por inscrições entre colchetes ao lado das transições. As fichas presentes em uma CPN sempre transportam algum valor que deve pertencer ao domínio de algum tipo de dado bem definido nas declarações. Por razões históricas, em CPN usa-se a expressão conjunto de cores (*colour set*) em substituição a tipo de dados e, por conseqüência, cada valor é denominado de cor (*colour*). Desta forma, a cada lugar na estrutura é associado um conjunto de cores que indica o tipo de fichas que o lugar pode conter. A linguagem CPN-ML dispõe de mecanismos que permitem a definição de conjuntos de cores relativamente complexos, além de oferecer diversos conjuntos previamente definidos.

Um conceito fundamental para o estabelecimento das CPN é a idéia de multi-conjuntos (*multi-sets*). Multi-conjuntos são semelhantes a conjuntos, exceto pelo fato de que podem conter múltiplas aparições de um mesmo elemento. Todo multi-conjunto é definido sobre um conjunto a partir do qual os elementos são tomados. A título de exemplo, os multi-conjuntos a seguir foram criados sobre o conjunto dos números naturais: $\{0,2,3,7\}$, $\{0,1,1,3\}$, $\{0,1,1,2,3,3,7\}$, $\{\}$. No Capítulo 6 introduz-se formalmente os multi-conjuntos. Uma marcação para uma CPN é uma distribuição de fichas nos seus lugares. A marcação de cada lugar é definida como um multi-conjunto sobre o conjunto de cores associado ao lugar.

É comum usar a expressão *variáveis da transição* para nos referirmos ao conjunto de variáveis presentes nas inscrições dos arcos e na guarda da referida transição. Outro conceito de extrema importância nas redes de Petri coloridas é o de ligação (*binding*) de uma transição. Uma ligação de uma transição pode ser vista como a substituição de cada variável da transição por uma cor (valor). É requerido, entretanto, que as cores pertençam aos conjuntos de cores apropriados e que impliquem na avaliação da guarda como verdadeira. Estabelecidos os conceitos de marcação e ligação podemos definir como se comportam as redes de Petri coloridas. Em cada marcação, uma transição sob uma determinada ligação é dita habilitada se todos os seus lugares de entrada tiverem fichas suficientes para satisfazer às expressões dos arcos. Cada expressão deve ser devidamente avaliada segundo as substituições determinadas pela ligação, a fim de determinar quantas e quais fichas são requeridas nos lugares de entrada. Caso a transição ocorra, então são retiradas fichas dos lugares de entrada e depositadas novas fichas nos lugares de saída. A quantidade de fichas é determinada também pela avaliação das expressões dos arcos segundo as substituições implicadas pela ligação.

5.4 Introdução Informal aos Sistemas ECPN

ECPN são estruturas de redes de Petri baseadas em objetos para a modelagem de BDTR. Elas foram definidas partindo-se dos princípios que originaram a estrutura G-CPN, introduzida em [Gue97; GPdF97; GdFP97] e nas redes de Petri coloridas [Jen92].

ECPN permitem o desenvolvimento incremental da especificação de um sistema, favorecendo as condições de manutenção e de reusabilidade. Isto ocorre pela utilização

dos conceitos de módulos fracamente acoplados, interfaces bem definidas e informação escondida, que resultam no fato de que um módulo só pode ter acesso a informações e métodos de outros módulos no sistema através de um mecanismo de abstração. A formalização de ECPN foi construída de forma a preservar aspectos gerais da natureza sintática das redes CPN [Jen92; Jen87], por exemplo, a mesma linguagem funcional é utilizada para as inscrições das redes, CPN-ML [Uni96]. Isto significa que, em certo nível, a modelagem em ECPN se assemelha à modelagem em CPN. Muito embora o comportamento das redes ECPN seja diferente, implicando em definições semânticas diferentes. Do ponto de vista intuitivo, o comportamento associado a um módulo ECPN é perfeitamente dedutível para desenvolvedores habituados às redes CPN.

5.4.1 Sistemas ECPN

Sistemas ECPN são conjuntos de módulos ECPN (apresentados na próxima seção) concorrentes, cooperantes e fracamente acoplados cujo objetivo é a modelagem e especificação formal e executável de sistemas de banco de dados em tempo-real. É importante estabelecer a diferença entre um sistema ECPN e um módulo ECPN, ou simplesmente, uma ECPN. Estruturalmente um sistema ECPN é a integração de um conjunto de módulos ECPN e um sincronizador. Cada módulo ECPN define um objeto instanciável do sistema. O sincronizador é o elemento responsável pela semântica associada ao sistema. Em se tratando da modelagem de sistemas distribuídos, o sincronizador pode ser visto como a abstração da rede de comunicação, sendo o responsável pela semântica de transferência de mensagens entre os módulos. Quando um módulo requisita um serviço de outro módulo, o sincronizador é responsável pelo transporte das mensagens de requisição e de resposta. Desta forma, do ponto de vista do módulo, a transferência pode ser abstraída, bastando indicar qual o serviço desejado e o identificador do módulo que o oferece. Uma outra característica relevante de módulos ECPN, é o fato de poderem efetuar chamadas recursivas.

O modelo de interação entre módulos ECPN é tipicamente cliente-servidor [Sim96]. Um módulo cliente requisita um serviço (execução de um método) em um módulo servidor. Quando a requisição da execução de um serviço é atendida pelo sincronizador, diz-se que uma *invocação* ocorreu. A partir daí, o módulo cliente apenas espera que o

sincronizador faça efetivamente a ativação do serviço remoto, e passa a esperar a chegada da resposta. O sincronizador, de posse das informações necessárias, efetua uma *ativação* do método invocado no módulo servidor através da passagem dos dados necessários ao método. Observe que, ceso funções de compatibilidade e restrições tenham sido declaradas na interface do módulo, o método somente é invocado se estas elas forem satisfeitas. Após a ativação, o módulo servidor atende ao pedido. Quando for obtido algum resultado, o sincronizador detecta a disponibilidade desses resultados (fichas em um lugar de terminação do método) e os retoma a fim de retransmiti-los ao módulo cliente. Este evento é denominado *terminação*. Após a ocorrência da terminação no módulo servidor, o sincronizador detecta o módulo cliente correspondente e devolve corretamente os dados, forçando uma ocorrência do último evento associado à interação, denominado *retorno*.

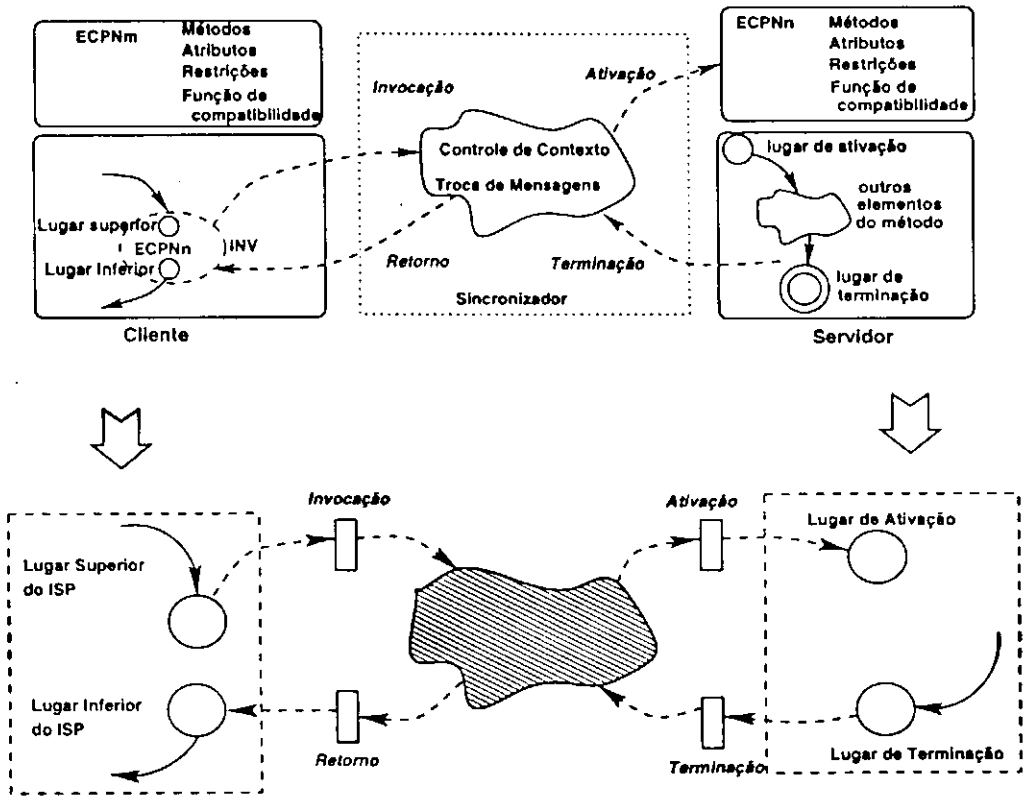

Figura 5.1: Sistema ECPN e eventos na interação

Os quatro eventos associados à interação entre dois módulos, mostrados na Figura 5.1, são: *invocação*, *ativação*, *terminação* e *retorno*. Observe que apenas dois deles ocorrem em cada um dos módulos envolvidos. Uma invocação e um retorno ocorrem

no módulo cliente, enquanto que uma ativação e uma terminação ocorrem no módulo servidor.

5.4.2 Módulos ECPN

Um módulo ECPN é estruturalmente representado por duas subestruturas: a primeira representa a interface de um módulo ou objeto e é denotada por IN; a segunda é a estrutura interna do módulo, denotada por EI. A interface determina a visão do módulo para o resto do sistema. Nela estão declarados os atributos (AT) e métodos encapsulados (MT). Para cada método declarado na interface existem dois lugares diferenciados na EI: o primeiro determina onde serão colocadas as fichas de ativação do método; e o segundo especifica qual o lugar de terminação. A estrutura interna é sintaticamente uma rede de Petri colorida. A semântica é modificada devido à introdução de um conjunto de lugares especiais denominados de *invocadores (INV)* na estrutura interna. Os invocadores são usados para invocar métodos remotos. Seu funcionamento é intuitivo: quando uma ficha é depositada nele, uma *invocação* pode ocorrer, implicando no desaparecimento da ficha; em seguida, dependendo do funcionamento do método remoto, deverá ocorrer um *retorno*, e uma outra ficha com novos dados será depositada no invocador, permitindo a habilitação das transições de saída.

Os lugares normais, bem como os atributos e lugares que indicam ativação de métodos, são representados graficamente por círculos (veja a Figura 5.2). Os lugares de terminação são representados por círculos de borda dupla. Invocadores são denotados por elipses e uma inscrição interna indicando o método e a ECPN a invocar. Os demais elementos (transições e inscrições) seguem a mesma notação usada em CPN. Uma exceção deve ser notada: dois conjuntos de cores são associados a cada invocador (INV), um para fichas de invocação e outro para fichas de terminação. Um único módulo ECPN pode atender a qualquer número de pedidos de serviços concorrentemente. Isso é possível porque ao ocorrer uma ativação, uma nova instância do módulo criada automaticamente, será encarregada de atender o pedido. Cada instância é tratada pelo que denomina-se de *contexto*. Um contexto pode ser visto como um novo objeto cuja estrutura funcional é uma cópia fiel da estrutura interna do módulo invocado. A única exceção é quanto aos lugares que representam atributos, que não serão copiados, mas

sim compartilhados. O contexto é automaticamente destruído quando não restarem mais fichas relativas àquela invocação na rede.

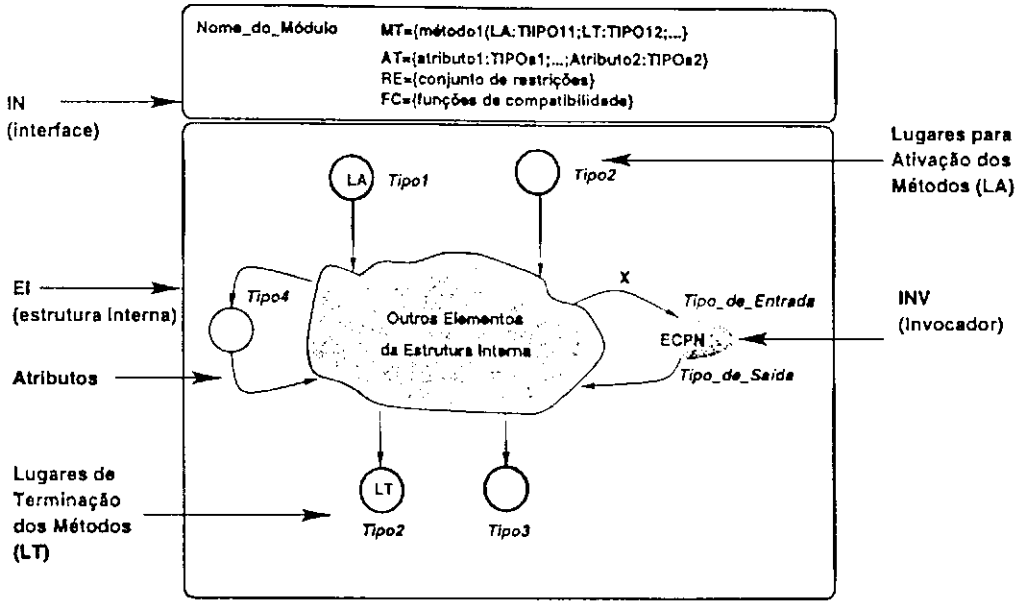

Figura 5.2: Notação para módulo ECPN

Como os atributos são considerados parte do estado global do objeto, o seu estado (marcação) é único e, portanto, compartilhado por todos os contextos do mesmo módulo ECPN. Conseqüentemente, os diversos contextos ativos concorrem pela utilização dos atributos (fichas nos lugares que os representam). Esta forma de definir ECPN permite que durante a modelagem um módulo ECPN represente ou um objeto propriamente dito, com características intrinsecamente concorrentes, ou uma classe, que a cada invocação constrói objetos idênticos. Neste caso, os atributos declarados devem representar atributos de classe.

5.5 Consideração Sobre Redes de Petri Temporizadas

Em muitas aplicações, as redes de Petri são utilizadas para investigar aspectos relacionados à satisfação de restrições lógicas de um dado sistema. Desta forma são consideradas as propriedades dinâmicas e as funções do sistema. Por outro lado, as redes de Petri podem também ser utilizadas para investigar aspectos relacionados ao desempenho dos sistemas modelados. Tais aspectos podem ser relacionados às medidas de natureza determinística, como por exemplo a investigação do tem-

po máximo utilizado para a execução de certas atividades, ou então aspectos não determinísticos como o tempo médio para atendimento de requisições. Para ser possível investigar estes aspectos ou propriedades relacionadas com o tempo, diversas técnicas foram definidas para as extensões temporais de redes de Petri. Estas extensões utilizam basicamente dois tipos de abordagem: determinística [BR90; GMMP89; MF76; Ram74; Sif80; vdA93; Zub91] e estocástica [AMBC84; HS86; Mol81; Nat80].

Na abordagem determinística, os requisitos de tempo são definidos como constantes e a análise, nesses casos, pode ser efetuada através de um método analítico ou por simulação. Em geral, nesse tipo de abordagem, a caracterização das restrições de tempo modifica o comportamento qualitativo do modelo em relação ao modelo correspondente sem as restrições de tempo. A abordagem determinística é adequada para a modelagem de sistemas em tempo-real em que a caracterização de limites de tempo é essencial. Essa abordagem é limitada para a avaliação de desempenho de sistemas em que, basicamente, determina-se o tempo de ciclo mínimo, isto é, o tempo necessário para atingir um determinado estado a partir de um estado inicial. As extensões determinísticas foram usadas em diversas áreas de aplicação como por exemplo, para modelar protocolos de comunicação [Mer79; MF76], e em sistemas flexíveis de manufatura [SJ91; SJ92], entre outras. Diversas metodologias de análise baseadas em grafos de alcançabilidade foram propostas [BD91; MJ92; Ram74; RH80; SP]. As extensões temporais determinísticas também foram aplicadas à classe de redes de Petri de alto-nível para reduzir a complexidade na modelagem de sistemas complexos [Fig94; GMMP91; Jen97; vdA92]. As redes de alto-nível temporais, notadamente as redes de Petri coloridas temporizadas [Jen97], são adequadas para modelagem de sistemas complexos. No Capítulo 6 introduz-se o conceito de ECPN temporizada com base no trabalho de Jensen [Jen97]. Esta opção é natural no contexto deste trabalho, uma vez que a base formal para o desenvolvimento das ECPNs são as redes de Petri coloridas.

5.6 Conclusão

Neste capítulo apresentou-se os conceitos básicos de redes de Petri clássicas e redes de Petri coloridas. Também introduziu-se informalmente os conceitos das redes de Petri

orientadas a objetos, ECPN, que são a base formal no desenvolvimento desta tese. No Capítulo 6 introduz-se formalmente os conceitos apresentados neste capítulo.

Capítulo 6

Formalização de ECPN

Neste capítulo são introduzidas as definições formais para ECPN. Primeiramente são introduzidos os conceitos básicos necessários ao entendimento das definições formais apresentadas. A seguir formalizam-se os conceitos de sintaxe e semântica para módulos e sistemas ECPN e a extensão temporizada adotada nesta tese. Primeiramente introduz-se a sintaxe e a semântica para um módulo e a seguir a sintaxe e a semântica para um sistema ECPN.

Com o objetivo de preservar a definição sintática da estrutura interna inalterada, será necessário incluir os novos elementos sintáticos na interface do módulo e associá-los aos lugares na estrutura interna. Embora, nenhum novo elemento seja adicionado à estrutura interna, algumas restrições são impostas aos elementos que os representam, a fim de que se possa inferir o comportamento adequado às invocações de serviços remotos. Além disso, as G-CPNs são estendidas considerando-se a inclusão de mecanismos que possibilitam descrever aspectos específicos para BDTR, tais como restrições temporais e funções de compatibilidade. Deve-se enfatizar que o objetivo foi definir mecanismos tais que para um módulo ECPN somente fossem introduzidas modificações sintáticas. De fato a semântica aqui apresentada para módulos ECPN na Seção 6.3 é essencialmente a introduzida por Guerrero em [Gue97; GPdF97; GdFP97; GdFP00]. Por outro lado, e a fim de preservar a semântica para os módulos e introduzir as restrições temporais e a função de compatibilidade, são introduzidas modificações na semântica dos sistemas ECPN. Nesta tese introduz-se o conceito de sincronizador, com base em [RAS96; RA96; NRA95], que proporciona a definição dos mecanismos para promover a comunicação

entre módulos ECPN e a garantia de comportamento de modo a satisfazer as restrições temporais e a função de compatibilidade.

No caso de modelos de transações periódicas com prazos estritos, como será detalhado no Capítulo 7, os modelos obtidos capturam a periodicidade e o tempo de liberação das transações.

6.1 Conceitos Básicos

Antes de introduzir a sintaxe para um módulo ECPN, é importante introduzir alguns conceitos básicos, tais como o de multi-conjunto (*multi-set*).

Definição 6.1 *Multi-conjunto*
Um multi-conjunto m, definido sobre um conjunto finito e não vazio S, é uma função $m \in S \rightarrow \mathbb{N}$, onde $\mathbb{N}$ é o conjunto dos inteiros não negativos, e $m(s) \in \mathbb{N}$ é o número de ocorrências do elemento $s \in S$ no multi-conjunto m [Jen92; Jen97].

Um multi-conjunto m pode ser representado pela soma formal:

$$\sum_{s \in S} m(s)`s$$

e S_{MS} denota o conjunto de todos os multi-conjuntos definidos sobre S. Considere por exemplo que $S = a, b, c, d, e$, então $m_1 = a + 2`c + e$ e $m_2 = a + 2`b + 3`c + e$, são membros de S_{MS}.

Para dar uma definição abstrata de ECPN, assim como no caso de CPN [Jen92], não é necessário fixar uma sintaxe concreta para escrever expressões nas redes, e então assumimos que tal sintaxe existe juntamente com uma semântica bem definida, tornando possível falar sem ambigüidades sobre:

- os elementos de um tipo, T. O conjunto de todos os elementos definidos em T é escrito simplesmente como T;
- o tipo de uma variável, v - denotado por $Tipo(v)$;
- o tipo de uma expressão, *Expressão* - denotado por *Tipo(Expressão)*;

- o conjunto de variáveis em uma expressão, *Expressão* - denotado por *Var(Expressão)*;
- uma ligação de um conjunto de variáveis, V - associando a cada variável $v \in V$ um elemento $b(v) \in Tipo(v)$;

Uma expressão *sem* variáveis é dita ser uma expressão fechada. Ela pode ser avaliada em todas as ligações, e todas as avaliações dão o mesmo valor.

Agora estamos prontos para definir ECPN. Observe que nas definições formais que seguem não se assume uma linguagem específica para escrever as expressões das redes.

6.2 Sintaxe de Módulos ECPN

No que segue introduz-se os conceitos formais básicos para um módulo ECPN[1].

Definição 6.2 *Módulo ECPN*

Um Módulo ECPN, denotado por E, é uma dupla $\langle \mathbf{EI}, \mathbf{IN} \rangle$ onde:

1. $\mathbf{IN}$ é a *interface* do módulo
2. $\mathbf{EI}$ é a *estrutura interna* do módulo

Definição 6.3 *Estrutura interna de um módulo* ($\mathbf{EI}$)

A tupla $\langle \Sigma, P, T, A, N, C, G, E, I \rangle$ é a estrutura interna de um módulo ECPN, onde:

1. Σ é um conjunto finito e não vazio de tipos denominado de *conjunto de cores.*
2. P é um conjunto finito de *lugares.*
3. T é um conjunto finito de *transições.*
4. A é um conjunto finito de *arcos* tal que $P \cap T = P \cap A = T \cap A = \varnothing$.
5. $N : A \rightarrow P \times T \cup T \times P$ é uma *função de nós.*
6. $C : P \rightarrow \Sigma$ é uma *função de cor.*

[1]Nas definições apresentadas neste capítulo, $\mathbb{B}$ denota o tipo *booleano* e contém os elementos $falso e verdadeiro$ que aplicam-se a operações padrão definidas para lógica proposicional.

7. $G : T \rightarrow \Sigma_{\text{Expressões}}$ é uma *função de guarda*, tal que:

$$\forall t \in T : [Tipo(G(t)) = \mathbb{B} \text{ e } Tipo(Var(G(t))) \subseteq \Sigma].$$

8. $E : A \rightarrow \Sigma_{\text{Expressões}}$ é uma *função de expressão de arcos*, tal que:

$$\forall a \in A : [Tipo(E(a)) = C(p(a))_{MS} \text{ e } Tipo(Var(E(a))) \subseteq \Sigma]$$

onde $p(a)$ é o lugar de $N(a)$.

9. $I : P \rightarrow \Sigma_{\text{Expressões}}$ é uma *função de iniciação de expressão*, tal que:

$$\forall p \in P : [Tipo(I(p)) = C(p)_{MS} \text{ e } I(p) \text{ é fechada}]$$

Deve-se observar que a definição da estrutura interna de um módulo ECPN é idêntica à definição sintática de redes coloridas não-hierárquicas como definidas por Jensen em [Jen92]).

Definição 6.4 *Interface de um módulo*(IN)

A tupla $\langle MT, AT, INV, FI, FT, LS, LI, OBJ, RE, FC \rangle$ é a interface de um módulo ECPN, onde:

1. MT é um conjunto finito e não vazio de *métodos*.

2. $AT \subseteq P$ é um conjunto finito de *atributos*.

3. INV é um conjunto finito de *invocadores*.

4. $FI : MT \rightarrow P$ e $FT : MT \rightarrow P$ são as funções de *iniciação* e *terminação*, respectivamente, onde $\forall m \in MT : [(FI(m) = p1 \Rightarrow I(p1) = 0) \wedge (FT(m) = p2 \Rightarrow I(p2) = 0)]$.

5. $LS : INV \rightarrow P$ e $LI : INV \rightarrow P$ as funções dos lugares *superiores* e *inferiores* para os *invocadores*, e
$\forall inv \in INV : [LS(inv) = p1 \wedge LI(inv) = p2 \Rightarrow p1^{\bullet} =^{\bullet} p2 = \varnothing \wedge I(p1) = I(p2) = 0]$[2].

[2] $p^{\bullet}$ e ${}^{\bullet}p$ denotam os conjuntos de saída e entrada para o lugar p, respectivamente.

6. $OBJ : INV \rightarrow S$ é a função *objetivo* para um *invocador*, onde S é o conjunto de todos os métodos invocáveis.

7. RE é um conjunto de restrições, tal que $\forall re \in RE, re : MT \rightarrow \Sigma_{\text{Expressões}}$ é uma *restrição*, e $\forall mt \in MT : [Tipo(re(mt)) = \mathbb{B}$ e $Tipo(Var(re(mt))) \subseteq \Sigma]$.

8. FC é um conjunto de funções de compatibilidade, tal que $\forall fc \in FC, fc = (fc_R, fc_{AI})$ e $fc : MT \times MT \rightarrow Im(fc_R) \times Im(fc_{AI})$, tal que:

   - $fc_R : MT \times MT \rightarrow \Sigma_{\text{Expressões}}$ é uma *função de restrição*, tal que: $\forall (mt_i, mt_j) \in MT \times MT : [Tipo(fc_R(mt_i, mt_j)) = \mathbb{B}$ e $Tipo(Var(fc_R(mt_i, mt_j))) \subseteq \Sigma]$.

   - $fc_{AI} : MT \times MT \rightarrow \Sigma_{\text{Expressões}}$ é uma *função de acumulação de imprecisão*, tal que: $\forall (mt_i, mt_j) \in MT \times MT : [Tipo(fc_{AI}(mt_i, mt_j)) = Tipo(FT(mt_j))$ e $Tipo(Var(fc_{AI}(mt_i, mt_j))) \subseteq \Sigma]$.

Com relação à Definição 6.4, algumas observações relevantes são enfatizadas a seguir:

1. O conjunto de atributos define os tipos de dados sobre os quais o módulo ECPN atua.

2. O conjunto de métodos declara e identifica os serviços permitidos através da interface.

3. O conjunto de invocadores contém um elemento identificador para cada ponto na estrutura interna que deva ser considerado sob a semântica de um invocador.

4. As funções de ativação e terminação de métodos associam a cada método dois lugares: o primeiro indica onde devem ser colocadas as fichas de ativação e, o segundo indica de onde devem ser retiradas as fichas de terminação. Além disso, os conjuntos de cores associados aos lugares de ativação e terminação definem indiretamente os tipos de dados recebidos e retornados pelos métodos. Exige-se que estes lugares não tenham marcação inicial nenhuma.

5. As funções de lugar superior e lugar inferior de invocadores associam dois lugares a cada invocador declarado na interface **IN**. A presença de fichas no lugar superior caracteriza as habilitações de invocações. O lugar inferior é onde devem ser colocadas as fichas relativas aos retornos. Lugares superiores não podem ter transições de saída, e lugares inferiores não podem ter transições de entrada. De forma análoga aos lugares de ativação e terminação também determinam as cores dos dados enviados e recebidos. Exige-se que tenham marcação inicial nula.

6. A função objetivo leva cada elemento de invocação - INV da estrutura interna no identificador do serviço que deve ser invocado. O conjunto S é descrito apenas informalmente a fim de que represente, no nível de módulo, um conjunto de serviços.

7. A função RE restringe a ativação dos métodos dependendo dos estados dos atributos para o módulo invocado.

8. Dado um par de métodos (m_i, m_j), qualquer $fc \in FC$ é definida por duas funções: fc_R restringe a ativação do método m_j dependendo do estado do m_i, e fc_{AI} determina a alteração do resultado de retorno para a invocação do método m_j ou m_i, caso fc_R seja avaliada como verdadeira.

Na definição de sistemas ECPN na Seção 6.4, o conjunto S, referenciado acima, será adequadamente formalizado.

Observe que a função OBJ mapeia cada *inv* para um único serviço. Desta forma o mesmo método é sempre invocado, a menos que as funções RE e FC não sejam satisfeitas.

6.3 Semântica de Módulos ECPN

As definições seguintes referem-se explicitamente a módulos ECPN introduzidos formalmente na Definição 6.2. No que segue, as seguintes convenções serão utilizadas:

1. $\mathbf{E} = \langle \mathbf{IN}, \mathbf{EI} \rangle$ é qualquer módulo ECPN.

2. $\mathbf{IN} = \langle MT, AT, INV, FI, FT, LS, LI, OBJ, RE, FC \rangle$ é a interface de **E**.

3. $\mathsf{EI} = \langle \Sigma, P, T, A, N, C, G, E, I \rangle$ é a estrutura interna do módulo.

4. $INV = \{inv_1, inv_2, \cdots, inv_i, \cdots, inv_n\}$ é o conjunto de serviços de E.

5. M_0, M_1 e M_2 são marcações alcançáveis de E

6. $Var(t)$ é o conjunto de variáveis associadas à transição t.

7. $Tipo(v)$ é o conjunto de cores da variável v

8. O é o conjunto de contextos.

O conjunto de contextos é utilizado para rotular fichas (mensagens) na invocação. Observe, que este rótulo identifica de forma única a mensagem, desta forma possibilitando que servidores possam devolver mensagens aos clientes corretos ou a suas instâncias corretas.

Definição 6.5 *Elemento de Ficha e Conjunto de Todos os Elementos de Ficha*
Um *elemento de ficha* é a tripla $\langle p, c, o \rangle$ onde: $p \in P$, $c \in C(p)$, e $o \in O$, onde O é um conjunto infinito de contextos. O conjunto de todos os elementos de ficha é denominado *EF*.

A Definição 6.5 estabelece que um elemento de ficha é determinado por um lugar na estrutura interna do módulo, uma cor pertencente ao conjunto de cores do lugar, e um contexto ao qual a ficha pertence. O contexto possibilita atividades concorrentes para um mesmo método em um módulo ECPN. Somente fichas de um mesmo contexto são utilizadas para habilitação e ocorrência de uma transição, exceção feita quando tratando com fichas de atributos. Como será discutido mais adiante neste caso defini-se um contexto especial.

Definição 6.6 *Marcação de um Contexto*
Uma marcação M para um módulo E é um multi-conjunto sobre EF. O multi-conjunto $\overset{o}{M}$ é a *marcação relativa ao contexto o* se e somente se $\overset{o}{M} \subseteq M$ e $\forall \langle p, c, o \rangle \in M$: $\langle p, c, o \rangle \in \overset{o}{M}$

De acordo com a Definição 6.6, uma marcação para um módulo ECPN é um multi-conjunto sobre o conjunto de elementos de ficha. A marcação de um contexto é uma

submarcação do conjunto de marcações restrita a um dado contexto. Ou seja, a submarcação para um dado contexto é dada pela marcação do módulo menos os elementos de ficha cujo contexto não é o referido.

Definição 6.7 *Ligação para uma Transição*

Uma ligação para uma transição t é uma função b definida sobre $Var(t)$, tal que $\forall v \in Var(t) : b(v) \in Tipo(v)$, e $G(t)\langle b \rangle$. O conjunto de todas as ligações para a transição t é denotado por $B(t)$.

Devido à sua independência da definição de elemento de ficha, a definição de ligação permanece inalterada em relação à definição de mesmo nome para redes CPN. A ligação para uma transição é a substituição de cada variável com uma cor válida e a avaliação da guarda da transição como verdadeira.

Definição 6.8 *Elemento de Ligação*

Seja O um conjunto infinito de contextos. Um elemento de ligação é uma tripla $\langle t, b, o \rangle$, tal que $t \in T$, $b \in B(t)$ é uma ligação pertencente ao conjunto de ligações da transição t; $o \in O$ é um contexto. O conjunto de todos os elementos de ligação é denotado por BE.

Um elemento de ligação é a associação de uma transição, uma ligação para a transição, e um contexto. Dado um elemento de ligação, pode-se identificar de forma única, quais substituições ocorrerão para uma determinada transição.

Definição 6.9 *Passo*

Um passo Y é um multi-conjunto sobre BE.

Definição 6.10 *Passo de Contexto*

Dado um passo $Y \in BE$ para um módulo E, um multi-conjunto $\overset{o}{Y}$ é o passo relativo ao contexto o ou simplesmente o passo do contexto o se e somente se $\overset{o}{Y} \subseteq Y$ e $\forall \langle t, b, o \rangle \in Y : \langle t, b, o \rangle \in \overset{o}{Y}$.

Observe que o objetivo é selecionar todos os elementos de ligação para um dado contexto. Um passo de contexto é dado pelo passo em questão menos os elementos de ligação de outros contextos.

Definição 6.11 *Passo de Contexto Habilitado*

Um passo de contexto $\overset{\circ}{Y}$ é dito habilitado em uma marcação $\overset{\circ}{M}$ se e somente se

$$\forall p \in P: \sum_{(t,b,o)\in\overset{\circ}{Y}} E(p,t)\langle b\rangle \leqslant \overset{\circ}{M}(p)$$

Um passo de contexto habilitado expressa quantas e quais fichas são necessárias em cada lugar na estrutura interna para permitir a ocorrência do passo. Caso o passo ocorra, as fichas são removidas dos lugares de entrada e novas fichas são adicionadas aos lugares de saída. Entretanto, deve-se notar que para um determinado módulo pode haver diferentes contextos ao mesmo tempo, devido a concorrência para um mesmo método, e que os atributos são recursos compartilhados independente do contexto. Logo, para que um passo esteja habilitado para um módulo ECPN, além da habilitação do passo de contexto, deve-se observar se existem fichas suficientes nos lugares de atributos.

Definição 6.12 *Passo Habilitado*

Um passo Y é dito habilitado na marcação M se e somente se:

1. $\forall o \in O: \forall \overset{\circ}{M} \subseteq M: \overset{\circ}{Y}$ está habilitado em $\overset{\circ}{M}$
2. $\forall p \in AT: \sum_{(t,b,o)\in Y} E(p,t)\langle b\rangle \leqslant M(p)$.

Esta definição permite detectar as pré-condições para uma ocorrência de passo num módulo ECPN. Um passo pode ocorrer apenas se todos os passos de contexto estão habilitados e os lugares que representam atributos têm fichas suficientes para satisfazer simultaneamente as expressões relativas a todos os contextos simultaneamente. Então, um passo está habilitado se todos os seus subpassos de contextos estiverem habilitados e se há fichas suficientes nos lugares de atributos para todos os elementos de ligação em Y. Portanto, nunca dois elementos de ligação compartilham ficha alguma em um passo habilitado.

As definições a seguir apresentam a formalização dos eventos em um módulo ECPN.

Definição 6.13 *Ocorrência de um Passo*

Um passo Y habilitado na marcação M_1 pode ocorrer. Se ocorre, então a marcação do módulo muda de M_1 para M_2 dada por:

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b,o)\in Y} E(p,t)\langle b\rangle) + \sum_{(t,b,o)\in Y} E(t,p)\langle b\rangle$$

Em um módulo ECPN, a ocorrência de um passo não é o único evento capaz de modificar as marcações. Cinco tipos de eventos são definidos para um sistema ECPN: ocorrências de passos ou simplesmente passos, invocações, ativações, terminações e retornos. Invocações, são eventos que removem fichas dos lugares superiores dos invocadores. Ativações, adicionam fichas com novos contextos nos lugares iniciais dos métodos invocados. Terminações, são eventos que removem fichas dos lugares alvo. Retornos são eventos que adicionam fichas aos lugares inferiores dos invocadores. A seguir define-se formalmente estes eventos do ponto de vista de um módulo.

Definição 6.14 *Invocação*
A invocação de $inv_i \in INV$ é habilitada na marcação M_1 pelo elemento de ficha $\langle p, c, o\rangle$ se e somente se $p = LS(inv_i)$. Neste caso, a invocação pode ocorrer. Se ocorrer a marcação do módulo muda de M_1 para M_2 definida por:

$$M_2 = M_1 - \langle p, c, o\rangle.$$

Onde $p \in P$, $c \in C(p)$, e $o \in O$

Uma invocação pode ocorrer se e somente se uma ficha é depositada no lugar superior de um *inv*. Se ela ocorre, então a ficha será removida.

Definição 6.15 *Ativação*
Dado um método $m \in MT$, a ativação do método m pode ocorrer num módulo cuja marcação é M_1. Se ocorrer então a marcação do módulo muda de M_1 para M_2 dada por:

$$M_2 = M_1 + \langle p_1, c_1, o_1\rangle.$$

Onde $p_1 \in FI(m)$, $c_1 \in C(p_1)$, $o_1 \in O$, e $\forall\langle p, c, o\rangle \in M_1 : o_1 \neq o$.

Observe que uma ativação está sempre habilitada. Ou seja, não há restrição alguma para a ativação da invocação de um método para um dado módulo ECPN. Apesar de ser verdadeiro para um módulo isto não é verdade para um sistema ECPN. Além disto, a ocorrência da ativação de um método provoca uma única alteração no estado do objeto:

acrescenta uma ficha, denominada ficha de inicialização, no lugar de ativação do método ativado. Ainda, o lugar ao qual se acrescenta a ficha, p_1 (definido por $FI(m)$), deve ter a mesma cor definida para o elemento de ficha, c_1.

Definição 6.16 *Terminação*
Dado um método $m \in MT$, a terminação do método m é habilitada na marcação M_1 pelo elemento de ficha $\langle p, c, o \rangle$ se e somente se $p = LT(m)$. Neste caso, a terminação pode ocorrer. Se ocorrer, a marcação do módulo muda de M_1 para M_2 definida por:

$$M_2 = M_1 - \langle p, c, o \rangle.$$

A única conseqüência da ocorrência de uma terminação ou uma invocação num módulo ECPN, é a eliminação da ficha que habilita o evento.

Definição 6.17 *Retorno*
Um retorno de $inv_i \in INV$ pode ocorrer num módulo cuja marcação é M_1. Se ocorre, então a marcação do módulo muda de M_1 para M_2 dada por:

$$M_2 = M_1 + \langle p_1, c_1, o_1 \rangle$$

Onde $p_1 \in LI(inv_i)$, $c_1 \in C(p_1)$, e $o_1 \in O$

Apresentou-se nesta Seção as definições formais relativas à estrutura e ao comportamento de um Módulo ECPN. O módulo assim definido permite a descrição executável de objetos de software em termos de atributos e métodos encapsulados. O acesso às informações se dá exclusivamente através de uma interface denominada IN que permite a invocação dos métodos. Dentro da estrutura interna, denominada EI, o formalismo permite descrever métodos invocáveis concorrentemente sem que o projetista precise dedicar esforços especiais para o controle. Além disso, a própria natureza das redes de Petri permite a descrição de métodos que desempenhem suas funções através de procedimentos intrinsecamente concorrentes. A fim de oferecer maior facilidade em termos de descrição de controle de concorrência, os atributos são elementos formalizados como lugares de acesso comum na estrutura interna, garantindo que apenas uma linha de execução terá acesso instantâneo. Através de um elemento de invocação é possível que descrições internas dos métodos invoquem serviços de elementos externos ao módulo.

Tudo isto é possível sem fazer nenhuma suposição sobre o sistema usuário do módulo, exceto que ele manterá a seqüência apropriada de eventos de comunicação. Na Seção seguinte define-se a sintaxe para sistemas ECPN.

6.4 Sintaxe de Sistemas ECPN

De forma intuitiva, um sistema ECPN poderia ser formalizado simplesmente como um conjunto de módulos ECPN. Entretanto, isso pode levar a diversos problemas de integração entre os módulos e conseqüentemente dificuldades bem maiores no estabelecimento das definições semânticas. Por exemplo, considere o fato de que cada módulo precisa *visualizar* os serviços disponibilizados pelos outros módulos. Isto é conseguido através da externalização dos elementos de acesso aos métodos de cada módulo a fim de formar um novo conjunto de serviços, igualmente visível para todos os módulos.

Ainda assim, um sistema ECPN será definido como uma estrutura que integra um conjunto de Módulos ECPN num sistema com um objetivo em comum. Entretanto, é preciso esclarecer precisamente como se forma essa estrutura que integra além de módulos ECPN, alguns componentes adicionais. Além disso, não é qualquer conjunto de módulos ECPN que pode formar um sistema ECPN. É preciso estabelecer critérios que permitam restringir a natureza dos módulos e suas relações para que determinem um sistema ECPN. Assim, por exemplo, dois módulos que não usem nenhum de seus serviços mutuamente não apresentarão nenhum problema de integração. Entretanto, se quisermos integrar os módulos A e B nos quais A invoca algum método do módulo B, então é preciso garantir que a passagem de parâmetros seja possível. Além disso, embora tenha ficado implícito, é necessário que cada método seja identificável de forma única em todo o sistema.

A definição sintática dada a seguir foi elaborada considerando as observações anteriores. O sistema deve ser, portanto, um sincronizador comum aos módulos, onde cada objeto (módulo ECPN) pode *ver* os demais objetos e seus serviços. A estrutura formal de um sistema ECPN, portanto, deve prover além do conjunto de módulos, um conjunto de identificadores de serviços que cada módulo oferece. Além disso a estrutura inclui o conjunto total de cores sobre o qual o sistema opera.

Nas definições que se seguem, utiliza-se o operador ponto . já consolidado pelas

linguagens orientadas a objetos para fazer referências a elementos pertencentes aos módulos ECPN do sistema. Assim, o conjunto de conjuntos de serviços dos módulos E_j e E_i, por exemplo, serão indicados por $\mathsf{E}_j.MT$ e $\mathsf{E}_i.MT$ ao invés de MT_j e MT_i.

Definição 6.18 *Sintaxe de Sistemas ECPN*
Um sistema ECPN é a tripla$(\Sigma_{\mathsf{SE}}, S, \mathsf{SE})$, onde:

1. $\mathsf{SE} = \{\mathsf{E}_1, \mathsf{E}_2, \cdots \mathsf{E}_n\}$ é um conjunto finito e não vazio de *módulos* tal que:

   $\mathsf{E}_1.P \cap \mathsf{E}_2.P \cap ... \cap \mathsf{E}_n.P = \varnothing$, ou seja, os conjuntos de lugares são disjuntos.

   $\mathsf{E}_1.T \cap \mathsf{E}_2.T \cap ... \cap \mathsf{E}_n.T = \varnothing$, ou seja, os conjuntos de transições são disjuntos.

   $\mathsf{E}_1.A \cap \mathsf{E}_2.A \cap ... \cap \mathsf{E}_n.A = \varnothing$, ou seja, os conjuntos de arcos são disjuntos.

2. $\forall \mathsf{E}_i \in \mathsf{SE} : \mathsf{E}_i.\Sigma \subseteq \Sigma_{\mathsf{SE}}$, onde $\Sigma_{\mathsf{SE}} = \mathsf{E}_1.\Sigma \cup \mathsf{E}_2.\Sigma \cup ... \cup \mathsf{E}_n.\Sigma$ é a união dos conjuntos de cores de todos os módulos do sistema ECPN.

3. $S = \mathsf{E}_1.MT \cup \mathsf{E}_2.MT \cup \cdots \cup \mathsf{E}_i.MT \cup \cdots \cup \mathsf{E}_n.MT$ é o conjunto de *serviços* do sistema. Seja $inv \in \mathsf{E}_i$ um invocador e $s \in \mathsf{E}_j.MT$ um serviço, se $\mathsf{E}_i.OBJ(inv) = \mathsf{E}_j.s$ então a restrição para o conjunto de cores deve ser satisfeita:

$$\mathsf{E}_i.C(LS(inv)) = \mathsf{E}_j.C(FI(s)) \wedge \mathsf{E}_i.C(LI(inv)) = \mathsf{E}_jC(FT(s))$$

A definição acima estabelece que um sistema ECPN é formado por um conjunto de módulos ECPN cujos métodos são unidos para formar o conjunto de serviços do sistema. É imposto, ainda, que cada módulo defina um subconjunto do conjunto de cores do sistema como seu próprio conjunto de conjuntos de cores. Todo par invocador-serviço deve ter em comum os conjuntos de cores de seus lugares correspondentes.

6.5 Semântica de Sistemas ECPN

No contexto de um sistema ECPN, denotado por SE, três conceitos novos centrais são introduzidos: sincronizador, estado e eventos.

O conceito de sincronizador foi introduzido na teoria de Atores (*Actors*) [NRA95; RAS96; RA96] e aplicado no contexto de CPN por Nigro em [NP98]. A idéia básica é disponibilizar um mecanismo de alto nível para a troca de mensagens em um sistema

de objetos. Em [Gue97] este conceito foi introduzido como ambiente. Entretanto, para evitar ambigüidades com o conceito de ambiente com o qual um sistema de software interage e devido a propriedade do termo, adotaremos o termo sincronizador no contexto desta tese.

Informalmente um sincronizador é uma parte do estado de um sistema ECPN que não está representada nas marcações dos módulos ECPN. Como em [Gue97] um sincronizador é formalizado como uma estrutura de conjuntos de relacionamentos que mantém a informação sobre pendências de seqüências de eventos não terminados, que são caracterizados pelos eventos de interação entre os módulos, quais sejam: invocações, ativações, terminações e retornos.

Desta forma espera-se que para toda invocação ocorrida em um módulo ocorra uma ativação no módulo apropriado do sistema ECPN. Observe que no contexto deste trabalho uma ativação somente está habilitada caso as restrições e funções de compatibilidade, caso definidas para o módulo invocado, sejam todas satisfeitas. É importante enfatizar que neste sentido tais restrições e funções desempenham um papel similar às restrições de interação definidas em [NRA95] para Atores. Ainda, uma terminação de um método somente ocorre após uma ativação do método, e uma tal terminação deve levar a ocorrência de um retorno no módulo invocador, o qual somente deve ocorrer após uma invocação.

Deve-se observar ainda que o sincronizador deve implementar os mecanismos que possibilitem uma associação correta para contextos entre módulos interagindo.

No que segue apresentamos as definições formais para sistemas ECPN. Primeiramente introduzimos a definição de *ativações pendentes*, que identifica pedidos de ativações prontos para serem efetivados. Uma ativação pendente contém informações sobre o serviço a ser ativado, o contexto e os dados de cor da ficha de iniciação a ser utilizada.

Definição 6.19 *Ativação Pendente*
Uma ativação pendente é uma tripla $\langle s, c_{SE}, o \rangle$, onde:

1. $s \in S$ é um serviço para um sistema ECPN.
2. $c_{SE} \in \Sigma_{SE}$ é uma cor do conjunto união de todas as cores de todos os domínios.
3. $o \in O$ é um elemento do conjunto de contextos.

Um segundo conjunto a ser definido é o conjunto dos retornos pendentes com o mesmo significado do conjunto de ativações pendentes, acima comentado e definido.

Definição 6.20 *Retorno Pendente*
Um retorno pendente é uma dupla $\langle c_{\mathsf{SE}}, o \rangle$, onde:

1. $c_{\mathsf{SE}} \in \Sigma_{\mathsf{SE}}$ é uma cor do conjunto união de todas as cores de todos os domínios.
2. $o \in O$ é um elemento do conjunto de contextos.

Observe que um retorno não contém informação suficiente para promover o retorno da ficha ao invocador. É necessário que sejam identificados o contexto e o invocador para o método invocado. Para definir a relação entre o contexto do módulo invocador e do invocado introduzimos o conceito de associação de contextos.

Definição 6.21 *Associação de Contexto*
Uma associação de contexto é uma tripla $\langle o_{invc}, o_{invs}, inv \rangle$, onde:

1. $\langle o_{invc}, o_{invs} \rangle \in O$ são elementos do conjunto de contextos.
2. $inv \in INV$, é um invocador do módulo que invoca o serviço.

Por fim estamos prontos para definir o conceito de sincronizador para um sistema ECPN. Um sincronizador é uma estrutura integrando um conjunto de ativações pendentes, um conjunto de retornos pendentes e um conjunto de associações de contexto.

Definição 6.22 *Sincronizador*
Um sincronizador é uma tripla $\mathbf{SI} = \langle \mathbf{AP}, \mathbf{RP}, \mathbf{AC} \rangle$, onde $\mathbf{AP}$ é um conjunto de ativações pendentes, $\mathbf{RP}$ um conjunto de retornos pendentes e $\mathbf{AC}$ um conjunto de associações de contexto.

Definição 6.23 *Estado de um Sistema ECPN*
Um estado E de um sistema ECPN SE, é dado por uma tripla $\langle \mathbf{SI}, \mathbf{FM}, \mathbf{NC} \rangle$, onde:

1. $\mathbf{SI}$ é um sincronizador.
2. $\mathbf{FM}$ é uma função de marcação de módulos definida por:
   $\mathbf{FM} : \mathsf{SE} \rightarrow \mathsf{E}_1.M \cup \mathsf{E}_2.M \cup \cdots \cup \mathsf{E}_n.M$
   $\forall j, 1 \leqslant j \leqslant n : \mathbf{FM}(\mathsf{E}_j) = \mathsf{E}_j.M$

3. NC é um conjunto de contextos denominados novos contextos tal que $\mathsf{NC} \subseteq O$.

A seguir são definidos os eventos que causam as mudanças de estado em um sistema ECPN. Tais eventos são classificados em: eventos que alteram somente as marcações dos módulos, e são denominados passos, e eventos que alteram a marcação dos módulos e do sincronizador são denominados invocações, ativações, terminações e retornos.

Definição 6.24 *Passo para um Sistema ECPN*
Um passo, Y_{SE}, para um sistema ECPN SE, é um conjunto formado por um passo habilitado de cada módulo do sistema que, quando ocorre, altera o estado do sistema de E_1 para E_2, dado que:

1. seja $E_1 = \{\mathsf{SI}, \{\mathsf{E}_1.M_1, \mathsf{E}_2.M_1, \cdots, \mathsf{E}_i.M, \cdots, \mathsf{E}_n.M_1\}, \mathsf{NC}_1\}$.

2. seja o passo habilitado $Y_{\mathsf{SE}} = \{\mathsf{E}_1.Y, \mathsf{E}_2.Y, \cdots, \mathsf{E}_i.Y, \cdots, \mathsf{E}_n.Y\}$

3. se Y_{SE} ocorre então o novo estado do sistema será:
   $E_2 = \{\mathsf{SI}, \{\mathsf{E}_1.M_2, \mathsf{E}_2.M_2, \cdots\cdots, \mathsf{E}_i.M_2, \cdots, \mathsf{E}_n.M_2\}, \mathsf{NC}_1\}$
   onde $\forall j, 1 \leqslant j \leqslant n : \mathsf{E}_j.M_2$, é a marcação do módulo E_j após a ocorrência de $\mathsf{E}_j.Y$

As definições dos eventos que ocorrem em uma comunicação entre módulos ECPN tem por base os eventos correspondentes nos módulos, e suas ocorrências devem determinar como são modificadas as marcações dos módulos, bem como o estado do sincronizador.

Para os módulos os eventos ocorrendo no sincronizador alteram suas marcações da mesma forma que os eventos correspondentes nas definições para os módulos. Basicamente, a diferença nas definições que seguem são a inclusão das condições para a ocorrência dos eventos no sincronizador e como o estado do sistema é alterado.

Definição 6.25 *Invocação para um Sistema ECPN*
Uma invocação $inv_j \in \mathsf{E}_i.INV$ está habilitada no estado $E = \{\mathsf{SI}, \{\mathsf{E}_1.M, \mathsf{E}_2.M, \cdots, \mathsf{E}_i.M, \cdots, \mathsf{E}_1.M\}, \mathsf{NC}\}$, se e somente se existe um elemento de ficha $\langle p, c, o\rangle$ habilitando uma invocação de inv_j em E_i no estado $\mathsf{E}_i.M$.

Observe que então o elemento de ficha $\langle p, c, o\rangle$ habilita a invocação inv_j no sincronizador.

Definição 6.26 *Ocorrência de uma Invocação para um Sistema ECPN*
Uma invocação $inv_j \in \mathrm{E}_i.INV$ habilitada para o estado E_1 pode ocorrer. Se ocorrer, então o estado do sistema é alterado de E_1 para E_2, dado por:

1. seja $E_1 = \{\langle \mathrm{AP}_1, \mathrm{RP}_1, \mathrm{AC}_1\rangle, \{\mathrm{E}_1.M_1, \mathrm{E}_2.M_1, \cdots, \mathrm{E}_i.M_1, \cdots \mathrm{E}_n.M_1\}, \mathrm{NC}_1\}$
2. $\langle p, c, o\rangle$ o elemento de ficha que habilita a invocação.
3. $E_2 = \{\langle \mathrm{AP}_2, \mathrm{RP}_1, \mathrm{AC}_2\rangle, \{\mathrm{E}_1.M_1, \mathrm{E}_2.M_1, \cdots, \mathrm{E}_i.M_2, \cdots \mathrm{E}_n.M_1\}, \mathrm{NC}_2\}$.
   onde: $\mathrm{AP}_2 = \mathrm{AP}_1 \cup \langle \mathrm{E}.OBJ(inv_j), c_{\mathrm{SE}}, nc\rangle$, onde $nc \in \mathrm{NC}$, $\mathrm{AC}_2 = AC_1 \cup \langle o, nc\rangle$, $\mathrm{E}_i.M_2 = \mathrm{E}_i.M_1 - \langle p, c, o\rangle$, e $\mathrm{NC}_2 = \mathrm{NC}_1 - nc$.

Definição 6.27 *Ativação para um Sistema ECPN*
A ativação do método $\mathrm{E}_i.m_k \in \mathrm{E}_i.MT$ é dita habilitada para o sistema ECPN no estado $E = \{\langle \mathrm{AP}_1, \mathrm{RP}_1, \mathrm{AC}_1\rangle, \mathrm{FM}, \mathrm{NC}\}$, se e somente se existe uma ativação pendente $\langle m, c_{\mathrm{SE}}, o\rangle \in AP$, onde $c_{\mathrm{SE}} \in C(FI(m))$, e $\mathrm{E}_i.RE(m_k) \rightarrow$ VERDADE, e $\forall FC(m_k, m_l) \in \mathrm{E}_i.MT \times \mathrm{E}_i.MT$, $\mathrm{E}_i.FC_R(m_k, m_l) \rightarrow$ VERDADE.

Uma ativação está habilitada se e somente se existe uma ativação pendente para o método e tanto as restrições quanto às funções de compatibilidade são verdadeiras.

Definição 6.28 *Ocorrência de uma Ativação para um Sistema ECPN*
Se a ativação do método $\mathrm{E}_i.m \in \mathrm{E}_i.MT$ está habilitada para o sistema ECPN no estado E_1, a ativação pode ocorrer. Se ocorrer, então o estado do sistema é alterado de E_1 para E_2, dado por:

1. seja $E_1 = \{\langle \mathrm{AP}_1, \mathrm{RP}_1, \mathrm{AC}_1\rangle, \{\mathrm{E}_1.M_1, \mathrm{E}_2.M_1, \cdots, \mathrm{E}_i.M_1, \cdots \mathrm{E}_n.M_1\}, \mathrm{NC}_1\}$
2. $\langle s, c_{\mathrm{SE}}, o\rangle$ a ativação pendente que habilita a ativação do método $\mathrm{E}_i.m \in \mathrm{E}_i.MT$
3. $E_2 = \{\langle \mathrm{AP}_2, \mathrm{RP}_1, \mathrm{AC}_1\rangle, \{\mathrm{E}_1.M_1, \mathrm{E}_2.M_1, \cdots, \mathrm{E}_i.M_2, \cdots \mathrm{E}_n.M_1\}, \mathrm{NC}_1\}$.
   onde: $\mathrm{AP}_2 = \mathrm{AP}_1 - \langle s, c_{\mathrm{SE}}, o\rangle$ e $\mathrm{E}_i.M_2 = \mathrm{E}_i.M_1 + \langle FI(\mathrm{E}_i.m), c_{\mathrm{SE}}, o\rangle$

Definição 6.29 *Terminação para um Sistema ECPN*
Uma terminação do método $\mathrm{E}_i.m \in \mathrm{E}_i.MT$ é dita habilitada para o sistema ECPN $E = \{\mathrm{SI}, \{\mathrm{E}_1.M, \mathrm{E}_2.M, \cdots, \mathrm{E}_i.M, \cdots, \mathrm{E}_1.M\}, \mathrm{NC}\}$, se e somente se existe uma ficha $\langle p, c, o\rangle$ habilitando a terminação de $\mathrm{E}_i.m \in \mathrm{E}_i.MT$ no estado $\mathrm{E}_i.M$.

Definição 6.30 *Ocorrência de uma Terminação para um Sistema ECPN*
Se a terminação do método $\mathsf{E}_i.m \in \mathsf{E}_i.MT$ está habilitada para o sistema ECPN no estado E_1, a terminação pode ocorrer. Se ocorrer, então o estado do sistema é alterado de E_1 para E_2, dado por:

1. Seja $E_1 = \{\langle \mathsf{AP}_1, \mathsf{RP}_1, \mathsf{AC}_1\rangle, \{\mathsf{E}_1.M_1, \mathsf{E}_2.M_1, \cdots, \mathsf{E}_i.M_1, \cdots \mathsf{E}_n.M_1\}, \mathsf{NC}_1\}$
2. $\langle p, c, o\rangle$ é a terminação pendente que habilita a terminação do método $\mathsf{E}_i.m \in \mathsf{E}_i.MT$
3. $E_2 = \{\langle \mathsf{AP}_1, \mathsf{RP}_2, \mathsf{AC}_1\rangle, \{\mathsf{E}_1.M_1, \mathsf{E}_2.M_1, \cdots, \mathsf{E}_i.M_2, \cdots \mathsf{E}_n.M_1\}, \mathsf{NC}_1\}$
   onde: $\mathsf{RP}_2 = \mathsf{RP}_1 + \langle c_{\mathsf{SE}}, o\rangle$ e $\mathsf{E}_i.M_2 = \mathsf{E}_i.M_1 - \langle p, c, o\rangle$

Definição 6.31 *Retorno para um Sistema ECPN*
Um retorno de um invocador $\mathsf{E}_i.inv \in \mathsf{E}_i.INV$ está habilitado em um sistema ECPN no estado $E = \{\langle \mathsf{AP}, \mathsf{RP}, \mathsf{AC}\rangle, \mathsf{FM}, \mathsf{NC}\}$ se e somente se existe um retorno pendente $\langle c_{SE}, o\rangle \in \mathsf{RP}$ e uma associação de contextos $\langle o_1, o_2, inv\rangle \in \mathsf{AC}$, tais que $c_{SE} \in C(LI(\mathsf{E}_i.inv))$ e $o = o_2$.

Definição 6.32 *Ocorrência de um Retorno para um Sistema ECPN*
Se o retorno do invocador $\mathsf{E}_i.inv$ está habilitado no sistema ECPN no estado E_1 então o retorno pode ocorrer. Se ocorrer, o então sistema muda do estado E_1 para o estado E_2, dado por:

1. seja $E_1 = \{\langle \mathsf{AP}_1, \mathsf{RP}_1, \mathsf{AC}_1\rangle, \{\mathsf{E}_1.M_1, \mathsf{E}_2.M_1, \cdots \mathsf{E}_i.M_1 \cdots \mathsf{E}_n.M_1\}, \mathsf{NC}_1\}$.
2. sejam $\langle c_{SE}, o_2\rangle$ e $\langle o_1, o_2, inv\rangle$ o retorno pendente e a associação de contextos que habilitam o retorno do invocador $\mathsf{E}_i.inv$
3. $E_2 = \{\langle \mathsf{AP}_1, \mathsf{RP}_2, \mathsf{AC}_2\rangle, \{\mathsf{E}_1.M_1, \mathsf{E}_2.M_1, \cdots, \mathsf{E}_i.M_2, \cdots \mathsf{E}_n.M_1\}, \mathsf{NC}_1\}$.
   onde: $\mathsf{RP}_2 = \mathsf{RP}_1 - \langle c_{SE}, o_2\rangle$ e, $\mathsf{AC}_2 = \mathsf{AC}_1 - \langle o_1, o_2, inv\rangle$, e $\mathsf{E}_i.M_2 = \mathsf{E}_i.M_1 + \langle LI(\mathsf{E}_i.inv), c_{SE}, o_1\rangle$.

Nos Capítulos 7 e 8 apresenta-se um exemplo de modelagem de sistema em tempo-real e BDTR integrados e a transformação de um sistema ECPN para uma CPN Hierárquica (CPNH) com o objetivo de validar os resultados teóricos e metodológicos no contexto de uma aplicação. No Capítulo 8 apresentam-se ainda resultados relevantes, relativos à análise e verificação de modelos ECPN transformados para CPNH.

6.6 ECPN Temporizada

Como discutido no Capítulo 5, estamos interessados em uma extensão temporal determinística para ECPN. Assim como Jensen [Jen97], o principal elemento introduzido é um relógio global representando o tempo do modelo. Intuitivamente, uma ficha, além de uma cor (tipo), também carrega um valor de tempo, denominado de rótulo de tempo, o qual descreve o tempo de modelo mais cedo que a ficha pode ser utilizada, ou seja removida do elemento de ligação (vide Definição 6.8).

Assim como para uma CPN temporizada, um elemento de ligação é dito ser habilitado para a cor quando os requisitos da regra de habilitação usual são satisfeitos (Definição 6.12). Entretanto, para estar habilitado, o elemento de ligação deve estar pronto. Significando que todos os rótulos de tempo das fichas removidas dos lugares de entrada de uma transição devem ser menores ou iguais ao tempo atual do modelo. A remoção de fichas para um módulo ECPN é apresentada na Definição 6.13. Na seção seguinte introduzimos alguns conceitos básicos, para então definir o conceito de módulo ECPN temporizado.

6.6.1 Conceitos Básicos

O objetivo desta Seção é introduzir algumas definições básicas necessárias ao entendimento da extensão temporizada. Uma expressão temporizada pode ser por exemplo:

$$2`(y,3)@[123,456] + 3`(k,3)@[678]$$

denotando que existem dois elementos (ou fichas) com a cor $(y,3)$, e rótulos de tempo 123 e 456, e três elementos de cor $(k,3)$ com rótulo de tempo 678. A expressão denota um multi-conjunto temporizado, definido a seguir.

Definição 6.33 *Multi-conjunto Temporizado*
Um multi-conjunto temporizado tm, definido sobre um conjunto finito e não vazio S, é uma função $tm \in [S \times \mathbb{Z}] \rightarrow \mathbb{N}$, onde $\mathbb{Z}$ é o conjunto dos números inteiros, tal que a soma:

$$tm(s) = \sum_{z \in \mathbb{Z}} ms(s,z)$$

é finita para todo $s \in S$ [Jen97].

O inteiro não negativo $ms(s)$ é o número de ocorrências do elemento s no multi-conjunto temporizado tm. A lista $tm[s] = [z_1, z_2, \cdots, z_{tm(s)}]$ contém os valores de tempo $z \in Z$ tais que $tm(s,z) \neq 0$. Cada z aparece $tm(s,z)$ vezes na lista, que é ordenada de tal forma que $z_i \leqslant z_{i+1}$ para todos $i \in 1..tm(s)-1$.

Um multi-conjunto temporizado tm é denotado pela soma formal:

$$\sum_{s \in S} tm(s)`s@tm[s]$$

Ainda, S_{TMS} denota o conjunto de todos os multi-conjuntos temporizados sobre S. Os inteiros não negativos $tm(s) : s \in S$ são denominados de coeficientes do conjunto temporizado tm, e $tm(s)$ é denominado de coeficiente de s. Um elemento $s \in S$ é dito pertencer ao multi-conjunto temporizado tm se e somente se $tm(s) \neq 0$ denotado por $s \in tm$.

É importante ainda observar que é possível definir um multi-conjunto ordinário determinado a partir do multi-conjunto temporizado $tm \in S_{TMS}$ denotado por $tm_U \in S_{TMS}$ e definido por:

$$tm_U = \sum_{s \in S} tm(s)`s$$

As mesmas operações definidas para multi-conjuntos ordinários podem ser aplicadas a multi-conjuntos temporizados, exceção feita à comparação e à subtração. Ao leitor interessado em detalhes relativos a estas duas operações sobre multi-conjuntos temporizados recomenda-se consultar [Jen97].

6.6.2 Módulo ECPN Temporizado

Definição 6.34 *Módulo ECPN temporizado (ECPNT)*
Um módulo ECPN temporizado é definido pela tripla $\langle \mathbf{E}, \mathbb{Z}, z_0 \rangle$, tal que:

1. $\mathbf{E}$ satisfaz o requisitos para um módulo ECPN (Definição 6.2, sendo que $E(a)$ e $I(p)$ são tanto multi-conjuntos temporizados ou multi-conjuntos ordinários definidos sobre $C(p(a))$ e $C(p)$ respectivamente.

2. $\mathbb{Z}$ é o conjunto de valores de tempo, denominado de rótulos de tempo.

3. $z_0 \in \mathbb{Z}$ é o valor inicial de tempo.

É importante observar que para um dado modelo ECPN podem haver cores não temporizadas, significando que fichas do tipo S devem estar *sempre* disponíveis, independentemente de restrições de tempo. Logo para um conjunto de cores não temporizado as expressões de iniciação para os arcos são do tipo S_{MS}.

O conjunto de ligações $B(t)$, os elementos de ficha EF, e os elementos de ligação BE e os passos Y são definidos da mesma forma que nas Definições 6.7–6.9.

Definição 6.35 *Marcação Temporizada*
Uma marcação temporizada é o multi-conjunto temporizado definido sobre EF. A marcação inicial M_0 é a marcação obtida pela avaliação da expressão de iniciação, $\forall p \in P : M_0(p) = I(p)_{z_0}$

Logo, um estado é um par $\langle M, z \rangle$, onde M é a marcação e z é o valor de tempo. Então, o estado inicial é o par $\langle M_0, z_0 \rangle$. Para um estado $S = (M, r)$, utiliza-se S_U para denotar o multi-conjunto ordinário $M_U \in EF_{MS}$. S_U é a marcação não temporizada determinada por S. Desta forma possibilitando descartar toda informação temporizada relativa ao estado S.

Definição 6.36 *Passo Temporizado Habilitado*
Um passo Y é dito habilitado na marcação $\langle M_1, z_1 \rangle$ no tempo z_2 se e somente se:

1. $\forall o \in O : \forall \overset{o}{M}_1 \subseteq M_1 : \overset{o}{Y}$ está habilitado em $\overset{o}{M}_1$.
2. $\forall p \in AT : \sum_{(t,b,o) \in Y} E(p,t)\langle b \rangle_2 \leqslant M_1(p)$.
3. $z_1 \leqslant z_2$.
4. z_2 é o menor elemento em $\mathbb{Z}$ para o qual existe um passo habilitado satisfazendo 1, 2 e 3.

A segunda condição na Definição 6.36 indica que os lugares além de possuirem fichas suficientes para satisfazer as expressões (habilitados), os rótulos de tempo de tais fichas têm valores suficientemente pequenos, ou seja estão prontos. A terceira condição garante que não há como recuar no tempo. Por fim, a última condição garante que os passos habilitados são executados na seqüência em que ficam prontos.

Definição 6.37 *Ocorrência de um Passo Temporizado*

Um passo Y habilitado na marcação $\langle M_1, r_1 \rangle$ no tempo r_2 pode ocorrer. Se ocorre, então a marcação do módulo muda de $\langle M_1, r_1 \rangle$ para $\langle M_2, r_2 \rangle$, e é dada por:

1. $\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b,o) \in Y} E(p,t)\langle b \rangle_{r_2}) + \sum_{(t,b,o) \in Y} E(t,p)\langle b \rangle_{r_2}$

A primeira soma representa as fichas removidas e a segunda as fichas adicionadas.

As demais definições para ECPN temporizada são as mesmas introduzidas anteriormente.

6.7 Conclusão

Neste Capítulo foram introduzidas as definições formais para módulos ECPN e sistemas ECPN. A seguir foram introduzidos conceitos estendendo ECPN com a noção de tempo. Nos capítulos que seguem os conceitos formalmente introduzidos são aplicados no contexto de BDTR.

Capítulo 7

Modelagem de Bancos de Dados em Tempo-real

A informação é o componente principal de qualquer sistema, uma vez que ela é a base para a comunicação entre os subsistemas existentes em uma empresa. Daí é importante que a informação seja corretamente modelada e seu uso cuidadosamente planejado para otimizar as operações e conseqüentemente evitar sobrecarregar o sistema desnecessariamente.

A modelagem e análise de sistemas de informação têm sido foco de muita pesquisa e desenvolvimento, tanto pela comunidade de banco de dados, com vários avanços na representação e manipulação dos dados, quanto pela comunidade de engenharia de software, com metodologias de análise e projeto e métodos de especificação formal. Atualmente, a maior parte dessas pesquisas tem se concentrado na abordagem orientada a objetos [Boo94; CY90; Dou98; RJB99; Rum91]. Vários métodos orientados a objetos para modelagem de sistemas foram propostos. Esses métodos podem ser usados para a modelagem dos aspectos estáticos e dinâmicos dos sistemas. No entanto, esses métodos não oferecem suporte ou disponibilizam mecanismos e técnicas muito limitadas para a análise formal. Esse suporte é extremamente importante no projeto de sistemas grandes e complexos, para reduzir o tempo de desenvolvimento e manutenção desses sistemas [Mar95], bem como aumentar a segurança em seu funcionamento [Jal94]. As características das redes de Petri e, em particular, das ECPNs introduzidas no Capítulo 5 e formalizadas no Capítulo 6, indicam que elas são mais apropriadas para a modela-

gem das propriedades dinâmicas de BDTR e fornecem suporte para a análise formal dos requisitos de tais sistemas.

Neste capítulo apresenta-se um método orientado a objetos para modelagem de BDTR. A aplicação do método resulta na obtenção de um *modelo de objetos* que captura os aspectos estáticos do sistema e um *modelo de processos* que captura os aspectos dinâmicos do sistema. O método suporta análise formal, como será introduzido no Capítulo 8. Além disso, o método suporta a modelagem integrada tanto dos aspectos puramente de controle de um STR, como aqueles relativos ao BDTR.

7.1 Método de Modelagem de Bancos de Dados em Tempo-real

Nesta seção apresenta-se um método para modelagem de BDTR. O método é baseado em OMT, (*Object Modeling Technique*), introduzida no Capítulo 3, no modelo de dados orientado a objetos em tempo-real RTSORAC, (*Real Time Semantic Objects Relationships and Constraints*) introduzido por DiPippo [DiP95b], e em ECPN.

A representação diagramática para o modelo de objetos da OMT é uma forma de diagrama de classes que considera todos os aspectos de orientação a objetos, tais como classes, agregação, associação, generalização e é bastante difundida entre os desenvolvedores de sistemas orientados a objetos. O RTSORAC, por sua vez, considera a definição de restrições temporais dos dados e uma função de compatibilidade que indica em que condições dois métodos de um objeto podem ser executados concorrentemente, mas não considera todos os aspectos de orientação a objetos considerados pelo diagrama de classes da OMT, como, por exemplo, agregação. Desta forma, juntamos as características positivas da OMT e do RTSORAC em um único modelo, como será visto adiante.

O método introduzido pode ser usado para modelar os três aspectos básicos de um sistema: dados, funções e dinâmica. A aplicação do método resulta na obtenção de um *Modelo de Objetos* e de um *Modelo de Processos*:

Modelo de Objetos: é usado para modelar as propriedades estáticas dos objetos, tais como: atributos, operações, e restrições lógicas e temporais,

Modelo de Processos: é usado para modelar as propriedades funcionais e dinâmicas dos objetos, isto é, modela as operações identificadas no modelo de objetos (métodos).

Nas seções seguintes detalhamos cada um destes modelos.

7.1.1 Modelo de Objetos

O *modelo de objetos* descreve as estruturas estáticas dos dados do sistema. Ele considera as estruturas em termos de grupos de objetos similares (classes), suas diferenças e similaridades (generalização), e seus relacionamentos (associações) [BP98]. Então, o modelo de objetos define o universo de discurso para o modelo de processos.

Cada *objeto* é uma entidade única; uma *classe* é um conjunto de objetos similares que tem a mesma estrutura (atributos) e o mesmo comportamento (operações); os *atributos* são propriedades estruturais das classes, que podem ter restrições lógicas e/ou temporais; os *relacionamentos* são as ligações entre as classes, e; as *operações* são funções ou procedimentos aplicáveis aos atributos das classes. Um *método* é uma implementação de uma operação.

Para modelar as propriedades estáticas do sistema usaremos como base o modelo de objetos da técnica OMT, que é adequado para representar objetos do mundo real de forma natural, tem uma representação gráfica simples e intuitiva, fácil de ser entendida tanto por usuários quanto projetistas do sistema. No entanto, a notação da técnica OMT não é suficiente para representar todas as características dos bancos de dados em tempo-real, como por exemplo, as restrições temporais dos dados e a compatibilidade de execução concorrente das operações, entre outras, introduzidas no Capítulo 2. Portanto, baseado no modelo RTSORAC, algumas extensões foram incorporadas ao modelo de objetos OMT, para que o mesmo possa ser utilizado para modelar as características dos bancos de dados em tempo-real.

Um modelo de objetos consiste de um conjunto de *diagramas de classes*, de um conjunto de *diagramas de objetos*, e de um *dicionário de dados*. Os diagramas de classes mostram a estrutura do sistema, enquanto os diagramas de objetos mostram instâncias dos objetos. O dicionário de dados define todas as entidades modeladas (classes, associações, atributos, operações, etc.), e explicita as razões para as decisões chave de modelagem [Rum91]. Freqüentemente, o modelo de objetos tem apenas os diagramas

de classes [AKZ96].

Obtendo o Modelo de Objetos

A modelagem dos objetos começa com uma análise da declaração do problema e é resumido nos seguintes passos:

1. *Identificação dos objetos e classes*;
2. *Identificação das associações entre classes*;
3. *Adição dos atributos das classes*;
4. *Uso de generalização para observar similaridades e diferenças*;
5. *Identificação das operações*;
6. *Identificação das operações concorrentes*;
7. *Identificação das restrições lógicas e temporais*;
8. *Refinamento do modelo*;

Em [Rum91], Rumbaugh detalha os passos 1-5 e 8. Os passos 6 e 7 foram adicionados à OMT, e serão detalhados aqui.

Identificação das Operações Concorrentes

Tendo identificado as principais operações que serão executadas pelo objeto, o projetista do sistema neste estágio analisa o sistema para descobrir quais as operações que precisam ser executadas concorrentemente e em que condições elas podem ser executadas. Em seguida ele define as funções de compatibilidade que descrevem detalhadamente as situações nas quais as operações podem ser executadas concorrentemente. Portanto é importante que o método usado para modelar o sistema seja fundamentado em métodos formais para se verificar se essas funções então sendo definidas adequadamente, ou seja, se os limites definidos pelo projetista estão dentro de padrões aceitáveis.

Identificação das Restrições Lógicas e Temporais

Neste estágio, o projetista deve declarar as restrições lógicas e temporais para os objetos. As restrições definem os estados corretos para os objetos. Assim, as restrições são definidas como predicados que incluem os atributos *valor*, *tempo* e *imprecisão* dos objetos. Por exemplo, na Figura 7.5 o predicado *agora-rótulodetempo*$\leq$ *avi* expressa uma restrição temporal no atributo *BD* que não deve ser maior que seu tempo de validade absoluta *avi*. O predicado *Tipo com T1,T2,Trj,Tni* expressa uma restrição lógica que indica que os tipos produzidos só podem ser T1,T2,Trj e Tni.

7.1.2 Modelo de Processos

O modelo de processos é a combinação de um modelo funcional e um modelo dinâmico em um único modelo.

- *Modelo Funcional*: define as operações dos objetos.
- *Modelo Dinâmico*: caracteriza as interações temporais dos objetos e suas respostas a eventos. Ele mostra sobre quais condições as operações são executadas e por quanto tempo elas devem continuar executando.

O método apresentado não separa o modelo funcional do modelo dinâmico. Os aspectos funcionais, temporais e de controle, bem como o acesso aos dados, são integrados em um único modelo, denominado *modelo de processos*.

Obtendo o Modelo de Processos

O modelo de processos é um sistema ECPN. Como apresentado no Capítulo 5, uma ECPN é definida por uma *interface* (IN) e sua *estrutura interna* (EI).

O modelo de processos descrito em ECPN é definido a partir do modelo de objetos. Para isso, os seguintes passos devem ser seguidos:

1. *Identificação dos objetos (ECPNs)*;
2. *Identificação das funções de cada objeto*;
3. *Definição da interface (IN)* de cada objeto;
4. *Definição da estrutura interna (EI)* de cada objeto.

Identificação dos objetos (ECPNs)

Neste passo identificam-se todos os objetos no modelo de objetos, e para cada objeto, para os quais foram definidos métodos, constrói-se um módulo ECPN.

Identificação das funções de cada objeto

Neste passo identificam-se quais são os métodos que devem ser executados por cada objeto e em seguida analisa-se e descreve-se detalhadamente o que cada objeto deve implementar, sem se preocupar com detalhes de realização.

Definição da interface de cada objeto (IN)

Neste passo declara-se a interface de cada objetos ECPN, indicando os métodos com seus respectivos argumentos de entrada e saída, as restrições definidas para a classe, além das funções de compatibilidade.

Definição da estrutura interna de cada objeto (EI)

Neste passo constrói-se as ECPN que modelam os métodos declarados na interface dos objetos satisfazendo os requisitos declarados na fase de identificação das funções dos objetos, ou seja, no passo 2.

Observe que as associações entre classes, indicações de generalização e agregação, e restrições entre associações, não são consideradas para o modelo de processos.

O modelo de processos pode ser usado nas fases de análise, projeto e implementação do sistema. Estas fases podem ser tratadas simultaneamente, uma vez que um sistema ECPN pode ser analisado tanto por simulação como pela enumeração e análise do espaço de estados, veja Capítulo 8. Então, o projetista de uma aplicação pode usar as ECPNs para simular a aplicação e discutir os detalhes do comportamento dinâmico da aplicação com os usuários em todos os estágios da especificação.

O modelo de processos especifica, na fase de modelagem conceitual, o comportamento interno das atividades do sistema. O modelo também serve como uma ferramenta de validação do esquema obtido, através da simulação da execução dos métodos e geração de informações que o usuário possa entender de forma mais simples. Nesta tese simulamos um sistema ECPN para diferentes cenários e geramos diagramas de seqüências de

mensagem, (veja Capítulo 8), que são utilizados para validação.

7.1.3 Modelando o Acesso aos Dados

As operações básicas de acesso aos dados, tais como consulta, inclusão, exclusão, alteração, e outras, podem ser modeladas com ECPN. Nas Figuras 7.1.a, 7.1.b, 7.1.c, e 7.1.d, mostram-se as quatro primeiras estruturas.

Os arcos podem conter inscrições que modelam as restrições de acesso aos dados e os dados que são passados entre lugares e transições.

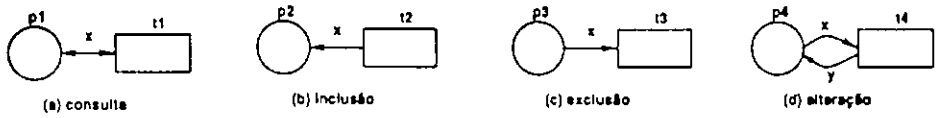

Figura 7.1: Operações básicas de bancos de dados

A Figura 7.1.a ilustra que a transição T1 apenas lê do lugar P1 (relação P1). No entanto, ela não muda esta relação, pois os arcos entre o lugar P1 e a transição T1 são rotulados da mesma maneira em ambas as direções. A Figura 7.1.b mostra que a transição T2 insere dados no lugar P2. O arco é direcionado de T2 para P2. A Figura 7.1.c mostra que a transição T3 exclui dados do lugar P3. O arco é direcionado de P3 para T3. A Figura 7.1.d mostra que a transição T4 altera os dados do lugar P4. Observe que os arcos entre T4 e P4 contém inscrições. O arco de P4 para T4 contém o valor a ser removido, e o arco de T4 para P4 contém o novo valor para ser colocado em P4.

7.1.4 Especificação de Transações com ECPN

Muitas transações são processos concorrentes que influenciam o acesso aos dados, ou outros recursos compartilhados no sistema por outras transações. Claramente, ECPN são bastante apropriadas para descrever os modelos de diversos tipos de transações, permitindo modelar adequadamente seqüências, concorrência e conflito.

A estrutura de um sistema ECPN permite modelar transações compostas de subtransações através de seus invocadores (INV), que são usados para prover a intercomunicação entre os objetos. Na Figura 7.2 mostra-se como modelar transações *seqüenciais* e *paralelas*. Por exemplo, para modelar um conjunto de subtransações seqüenciais, os

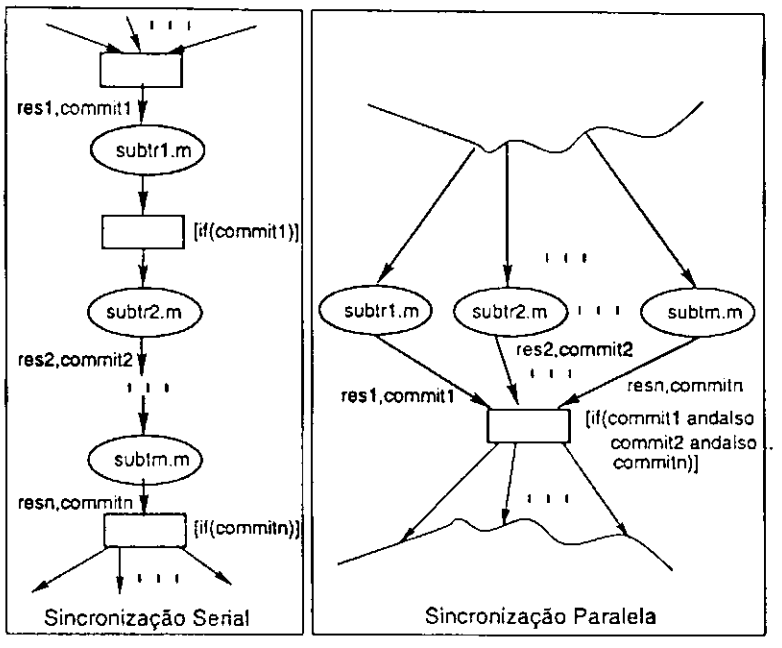

Figura 7.2: Transações seqüenciais e paralelas

limites de uma subtransação não interferem com outra. Cada subtransação, dentro da seqüência, só pode iniciar quando a subtransação anterior terminar. Na estrutura paralela de uma transação, modelando concorrência, as subtransações podem começar em tempos diferentes, mas possuem uma transição comum sincronizando suas operações. Observe que uma transição modelando sincronização, só pode ter como lugares de entrada as subtransações que ela esta sincronizando. A estrutura de transações aninhadas é o modelo normal de transações ECPN, uma vez que ele usa o modelo de interação cliente-servidor.

7.1.5 Transações de uma Aplicação de Tempo-real

No contexto desta tese, consideramos o escalonamento de transações periódicas com prazos estritos. A abordagem que introduzimos nesta seção baseia-se em [CGGC96]. Tal abordagem foi adotada, pois se adequou a ECPN de forma natural visto que foi proposta para redes de Petri Coloridas. Com o objetivo de conceitualizar a abordagem, considere um conjunto finito de transações em tempo-real modeladas em ECPN, onde para cada transação τ_i define-se os parâmetros temporais pela quatupla (tl_i, tc_i, pr_i, pe_i), onde

1. tl_i: é o tempo de liberação da transação, isto é, o momento no qual todos os

recursos necessários à execução da transação τ_i estão disponíveis. A partir deste momento a transação estará pronta para ser executada.

2. tc_i: representa o tempo computacional da transação, isto é, o tempo de processamento necessário para a executá-la.

3. pr_i: especifica o prazo máximo para a execução da transação τ_i.

4. pe_i: indica a periodicidade da transação.

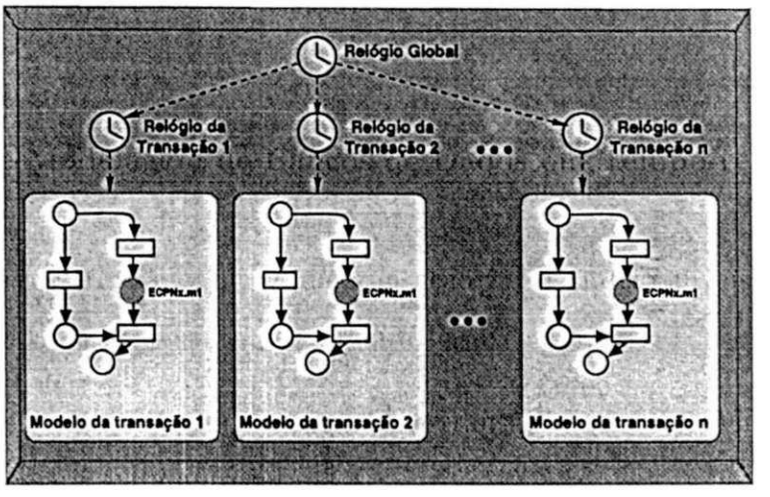

Figura 7.3: Transações em tempo-real com prazo estrito

Na Figura 7.3 ilustramos um conjunto de transações modeladas por um único módulo ECPN. Cada transação tem seu relógio local que é sincronizado por um relógio global. As características temporais são modeladas da seguinte forma:

- No modelo existem relógios locais para cada transação e um relógio global que representa o avanço do tempo no sistema. Estes relógios são representados por lugares e transições adicionais ao modelo de cada transação. Para detalhar o procedimento considere a ECPN apresentada na Figura 7.11 (página 130), então:
  - os pares (AtualizaS Subprodutos,AcessaSP SubProduto), (AtualizaP Produtos,AcessaP Produtos), e (LeOBDCellT, AcessaBD Produtos), para os relógios locais de cada transação.
  - o par (Atualiza Banco de Dados, Relógios Locais das Transações), para o relógio global.

- Os tempos computacionais de cada transação são definidos pelos tempos de execução dos objetos invocados.

- Os prazos de execução são identificados através de marcações terminais, indicando se uma tarefa cumpriu ou não seu prazo para execução.

- O tempo de liberação é representado pela marcação inicial do relógio local da transação, e são definidas da seguinte maneira:

$$M_0(\text{Relógio\_Local}) = \begin{cases} 1 & \text{se } tl = 0 \\ pe - tl + 1 & \text{se } tl > 0 \end{cases}$$

$$M_0(\text{Ativação}) = pronta + terminada = \begin{cases} pronta + terminada & \text{se } tl = 0 \\ terminada & \text{se } tl > 0 \end{cases}$$

Na Figura 7.11, para a transação Atualizacao de Subprodutos Produzidos (à esquerda da figura), o período é de 12 unidades de tempo ($pe = 12$) e o tempo de liberação é de 3 unidades de tempo ($tl = 3$). Assim, a marcação inicial para o *Relógio_Local*, no lugar AtualizaS Subproduto temos 10 fichas (10'e) e a marcação inicial para o *lugar de Ativação* (IniciaAS) é 1'terminada).

- Os períodos são representados pela quantidade de fichas removidas dos lugares modelando os relógios locais. No caso da Figura 7.11, a inscrição do arco entre o lugar AtualizaS SubProdutos e a transição AcessaSP SubProduto é 12'e, desta forma determinando o período de 12 unidades de tempo.

Uma transação pode ser iniciada quando num lugar de ativação, por exemplo, no lugar IniciaAS há duas fichas, ou seja: 1'pronta++1'terminada. A execução da transação é iniciada pela ocorrência da transição de iniciação associada a este lugar, que no caso do exemplo é a transição LESP SubProduto, a qual retira as duas fichas do lugar de ativação. Ao fim da execução da transação, no exemplo modelado pela transição FAtualizouOBDSP Subprodutos, uma ficha 1'terminada é depositada no lugar de ativação. Observe que caso o tempo de execução da transação tenha expirado, uma ficha 1'pronta é depositada no lugar de ativação, caso não exista uma ficha 1'terminada neste lugar, isto indica que a transação não cumpriu seu prazo de execução. Observe que a marcação 1'pronta++1'terminada indica uma marcação terminal para o exemplo considerado. Outros detalhes de modelagem são descritos na Seção 7.3.

7.2 Modelo do Sistema de Controle de uma Célula de Manufatura

As idéias básicas do método apresentado são ilustradas através de um exemplo que mostra como obter o modelo conceitual da célula de manufatura da Figura 7.4.

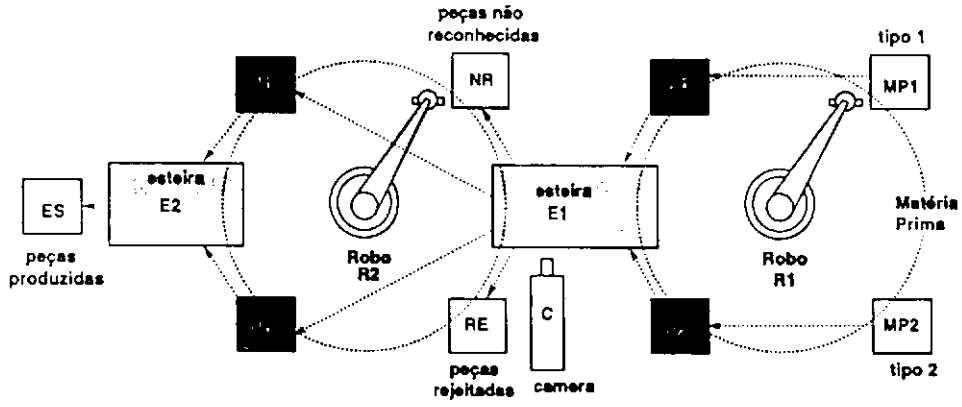

Figura 7.4: Uma célula de manufatura

A célula de manufatura é composta de dois robôs R1 e R2, quatro máquinas P1, P2, M1 e M2 que produzem peças do *tipo*1 e *tipo*2, cinco depósitos MP1, MP2, NR, RE e ES, duas esteiras E1 e E2, e uma câmera C. As máquinas P1 e P2 são utilizadas no pré-processamento da matéria-prima. As máquinas M1 e M2 são utilizadas para processamento adicional da matéria-prima. As esteiras E1 e E2 transportam as peças dentro da célula. O robô R1 é responsável por movimentar a matéria-prima dos depósitos MP1 e MP2 para as máquinas P1 e P2, respectivamente, e deles para a esteira de transporte E1. A câmera C obtém imagens das peças, e a partir destas imagens e de um banco de dados contendo tipos de peças, estas são reconhecidas e classificadas como *tipo*1 ou *tipo*2. O robô R2 é responsável por remover as peças da esteira E1 e coloca-las nos depósitos NR ou RE ou nas máquinas M1 e M2. Peças não reconhecidas, por falta de tempo, são depositadas no depósito NR para inspeção futura. Peças não reconhecidas, por não constarem no banco de dados, são depositadas no depósito de rejeitados RE. Peças reconhecidas são depositadas nos centros de processamento adicional M1 ou M2, de acordo com seu tipo, *tipo*1 ou *tipo*2, respectivamente. Após o processamento, as peças são depositadas na esteira E2, que as conduz para o depósito de peças produzidas ES.

Neste exemplo pode-se observar que alguns dados e ações têm restrições temporais. Por exemplo, as características das peças são capturadas pela câmera com o objetivo de reconhecer seu tipo e verificar se existe alguma anomalia. A peça deve ser reconhecida através da comparação de suas características com modelos de diferentes peças armazenadas no banco de dados. Este reconhecimento deve ser concluído em tempo, de forma que o comando que direciona o robô possa ser dado antes que a peça alcance o ponto onde ela deve ser enviada para outro local. Dependendo das características observadas, a peça é enviada para o local apropriado e o sistema atualiza o banco de dados com informações sobre a peça. As características da peça devem ser coletadas enquanto ela ainda está em frente à câmera, e elas aplicam-se apenas à peça em frente à câmera, ou seja, elas perdem sua validade no momento que outra peça é apresentada ao sistema. Se as ações com restrições de tempo não forem executadas dentro dos limites estabelecidos, outras alternativas devem ser possíveis.

Inicialmente cria-se o modelo de objetos para a célula. Em seguida, esse modelo é utilizado como base para o projeto do modelo de processos em ECPN. O modelo de objetos para a célula de manufatura é apresentado na Figura 7.5.

Para o objeto PEÇA definimos o atributo SPD: TIPO*QUANTIDADE para armazenar os tipos de peças pré-processadas e suas respectivas quantidades. Definimos o método RP() para ler a quantidade de peças pré-processadas e, o método MP() para pré-processar as peças. Também definimos uma restrição TIPO com SP1,SP2 que determina que só podemos pré-processar peças do *tipo1* ou *tipo2*.

Para o objeto CELLDB definimos o atributo DB: TIPO*QUANTIDADE*AVI*ROTULODETEMPO*IMPRECISÃO para armazenar os tipos de peças processadas, incluindo as peças rejeitadas e as não identificadas, e suas respectivas quantidades. Definimos o método RP() para ler a quantidade de peças processadas, e o método AT() para atualizar a quantidade de peças processadas. Definimos uma função de compatibilidade que indica quando os métodos RP e AT podem ser executados concorrentemente. Também definimos uma restrição TIPO com T1,T2,Trj,Tni que determina que as peças armazenadas no banco de dados podem ser dos tipos *tipo1*, *tipo2*, *rejeitadas*, ou *não identificadas*.

Para o objeto ROBÔ definimos o método MM() para realizar o movimento de um robô dentro da célula. Também definimos uma restrição TIPO com R1,R2 que determina

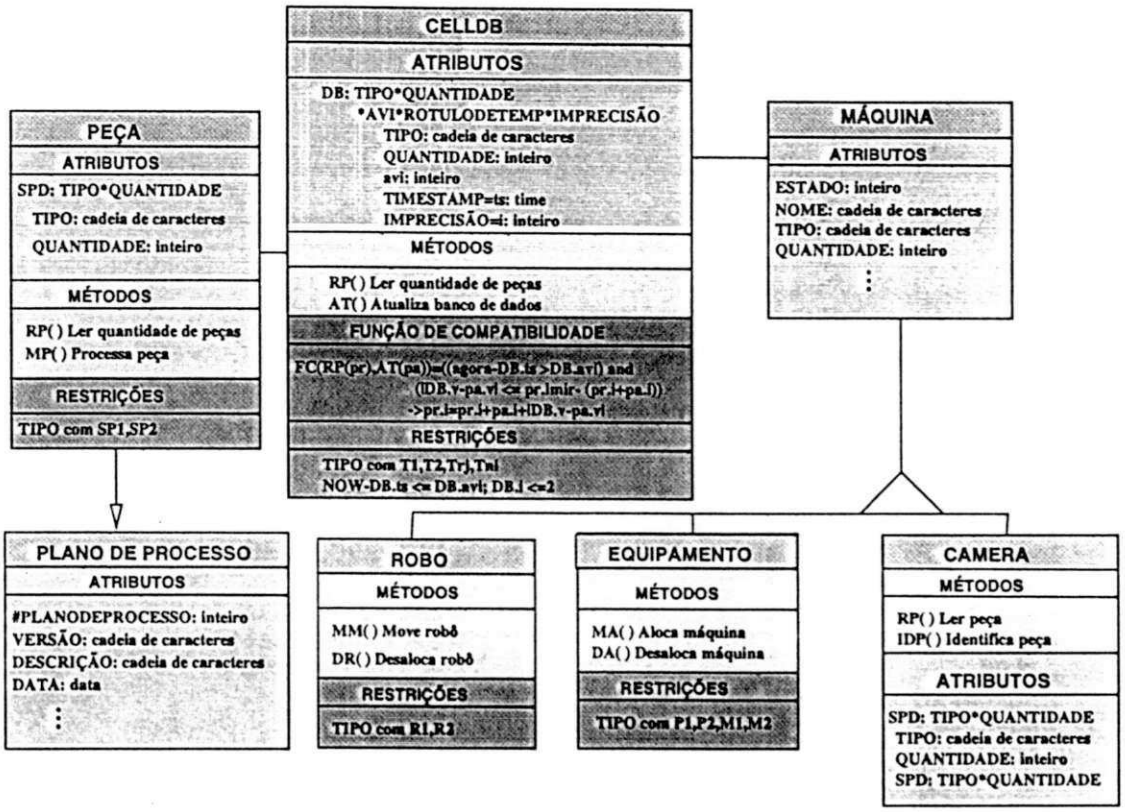

Figura 7.5: Modelo de objetos para a célula de manufatura da Figura 7.4

os tipos de robôs.

Para o objeto EQUIPAMENTO definimos o método MA() para alocar uma máquina para processar uma peça e, o método DA() para desalocar uma máquina. Também definimos uma restrição TIPO com P1,P2,M1,M2 que determina os tipos de máquinas.

Para o objeto CÂMERA definimos o atributo SPD: TIPO*QUANTIDADE para armazenar os tipos de peças processadas e suas respectivas quantidades. Também definimos o método RP() para ler a quantidade de peças processadas e, o método IDP() para identificar o tipo de peça processada e dar continuidade ao processamento adicional da peça.

7.2.1 Modelo de Processos com ECPN

Nesta seção exemplifica-se a aplicação dos passos introduzidos na Seção 7.1.2 para obter o modelo ECPN para o exemplo da célula de manufatura. A definição das cores, variáveis, constantes e funções usadas nas ECPN estão no Apêndice B.

Passo 1: Identificação dos objetos

Neste primeiro passo, os objetos (com métodos definidos) são identificados a partir do modelo de objetos. Assim, os objetos identificados são: EQUIPAMENTO, CÂMERA, ROBÔ, PEÇA e CELLDB.

Na Seção 7.2.2 detalha-se o comportamento dos modelos ECPN Produtos, Camera[1], e OBDCel, que modelam os objetos PEÇA, CÂMERA e CELLDB, respectivamente.

Passo 2: Identificação das funções de cada objeto

Neste passo identificam-se as funções de cada objeto. O objeto PEÇA, modelado pela ECPN Produtos, apresentada na Figura 7.6, com base no tipo de peça que deve ser pré-processada, aloca o equipamento para iniciar o pré-processamento da peça (método ProduzPeca), e quando terminar a operação, atualizar o banco de dados (lugar SubPecas Produzidas). Também possibilita a leitura da quantidade de peças pré-processadas (método LeSubPeca).

O objeto EQUIPAMENTO, modelado pela ECPN Equipamento, apresentada na Figura 7.10, aloca (método Aloca) e desaloca (método Desaloca) as máquinas M1 e M2 e os pré-processadores P1 e P2, e indica quando uma peça é concluída.

O objeto ROBÔ, modelado pela ECPN Robo, apresentada na Figura 7.9, implementa os mecanismos para comandar o movimento de matéria-prima dos depósitos MP1 e MP2 para os pré-processadores P1 e P2, respectivamente, mover as peças dos pré-processadores P1 e P2 para a esteira E1 e desta para as máquinas M1 ou M2, ou depósitos NR ou RE, dependendo do tipo indicado (método RoboMove).

O objeto CÂMERA, modelado pela ECPN Camera, apresentada na Figura 7.7, implementa os mecanismos para decidir a rota que uma peça deve seguir, de acordo com o seu tipo (*tipo1, tipo2, não reconhecido, ou rejeitado*). Quando a peça é identificada, ela deve ser enviada para processamento final, nas máquinas M1 ou M2, ou para o depósito de rejeitados. Se ela não for identificada, deve ser enviada para o depósito de objetos não identificados. Em seguida a quantidade de peças é atualizada de acordo com seu tipo (método Identifica). Também permite a leitura de peças já produzidas, incluindo

[1] A ausência de acentos nos nomes utilizados no modelo se deve ao fato de que a ferramenta utilizada, o Design/CPN (veja Capítulo 8), não suportar o conjunto de caracteres ISO-LATIN.

peças rejeitadas e não identificadas (método LeTipo).

O objeto CELLDB, modelado pela ECPN OBDCel, apresentada na Figura 7.8, acessam as informações sobre as quantidades de peças armazenadas no banco de dados. Os métodos AtualizaOBDC e LeOBDC implementam a atualização e a leitura do banco de dados, respectivamente.

Passo 3: Definição da interface de cada objeto (IN)

No terceiro passo, definimos a interface de cada objeto.

Para a ECPN Produtos, a interface é definida pelos métodos ProduzPeca e LeSubPeca, e pelo atributo SubPecas Produzidas que corresponde a um registro, onde são armazenadas as quantidades de peças pré-processadas.

1. Método ProduzPeca: Implementa o pré-processamento de peças. Ele recebe como argumento, uma ficha da cor TPROD, que indica o tipo da peça que deve ser produzida. Para este método, o lugar inicial, isto é, o lugar onde a ficha é inicialmente depositada quando o objeto Produtos é invocado, é o lugar ProduzPeca da estrutura interna.

2. Método LeSubPeca: Implementa a leitura da quantidade de peças pré-processadas por P1 e P2. Ele recebe como argumento, uma ficha da cor SPROD, que indica o tipo da peça que deve ser lida. Para este método, o lugar inicial, é o lugar LeSProd da estrutura interna.

Para a ECPN Camera, a interface é definida pelos métodos Identifica e LeTipo, e pelo atributo ProdutosId que corresponde a um registro onde serão armazenadas as quantidades de peças processadas.

1. Método Identifica: Este método implementa o mecanismo de decisão de roteamento da peça na esteira. Ele recebe como argumento, uma ficha da cor PROD, que indica o tipo de peça pré-processada. Para este método, o lugar inicial é o lugar Identifica da estrutura interna.

2. Método LeTipo: Este método implementa a leitura do total de peças produzidas por tipo, peças não-identificadas ou rejeitadas. Ele recebe como argumento de

entrada uma ficha da cor PROD, que indica o tipo de peça que deve ser lido. Para este método, o lugar inicial é o lugar LeTipoPROD da estrutura interna.

Para a ECPN Robo, a interface é definida pelo método RoboMove.

1. Método RoboMove: Este método realiza a funcionalidade para comandar os movimentos necessários dos robôs dentro da célula, transportar as peças dos equipamentos para as esteiras, depósitos, etc. O lugar inicial para a realização deste método na estrutura interna é o lugar MoveItem. Ele recebe como argumento de entrada, uma ficha da cor MOVE_ROBO, que indica o tipo de peça, o robô a ser usado, e a ação a ser realizada pelo robô.

Para a ECPN Equipamento, a interface é definida pelos métodos Aloca e Desaloca.

1. Método Aloca: Este método define a alocação do equipamento necessário para processar a peça, de acordo com seu tipo e fase de processamento. Recebe como argumento de entrada uma ficha da cor PROD_ROBO, que indica o tipo de peça a ser processado e o robô a ser usado. O lugar inicial deste método é o lugar AlocaEquip.

2. Método Desaloca: Este método é utilizado para indicar que uma máquina ou pré-processador completou sua atividade e que a peça sendo processada pode então ser removida. Este método recebe como argumento de entrada uma ficha da cor EQUIP, que indica qual dos recursos deve ser desalocado. O lugar inicial deste método é o lugar Desaloca.

Para a ECPN OBDCel, a interface é definida pelos métodos AtualizaOBDC, LeOBDC e pelo atributo OBDCel que corresponde ao banco de dados onde são armazenados os dados de peças.

1. Método LeOBDC: Este método implementa a leitura da quantidade de peças produzidas. Ele recebe como argumento de entrada uma ficha da cor PRODSIMPLMI, que indica o tipo da peça que deve ser lido, a imprecisão acumulada, e o limite máximo de imprecisão aceita. Para este método o lugar inicial é o lugar LeOBD da estrutura interna.

2. Método AtualizaOBDC: Este método implementa a atualização da quantidade de peças produzidas. Ele recebe como argumento de entrada uma ficha da cor OBDCEL que define um registro indicando o tipo e quantidade da peça que deve ser atualizada. Para este método o lugar inicial é o lugar AtualizaOBD da estrutura interna.

Passo 4: Definição da estrutura interna de cada objeto(EI)

Finalmente, no quarto passo, definimos a estrutura interna de cada objeto. A estrutura interna é uma rede de Petri colorida que modela os métodos declarados na interface do objeto.

7.2.2 Comportamento dos Objetos Modelados em ECPN

Nas seções seguintes detalha-se o comportamento dos objetos de controle Produtos e Câmera, e do objeto de banco de dados OBDCel. Os outros objetos são descritos em [GFPdF97; PTP98; PPC96].

Comportamento para o Objeto Produtos

Na Figura 7.6 apresenta-se o modelo ECPN para o objeto Produtos. Este objeto ativa o pré-processamento de peças, do *tipo*1 e do *tipo*2. O objeto mantém também informação relacionada ao número de sub-produtos de cada tipo que foram pré-processadas. Como dito na Seção 7.2.1, dois métodos são definidos para este objeto, quais sejam: método ProduzPeca e método LeSubPeca.

Quando a ECPN Produtos é invocada para pré-processar uma peça, com o método ProduzPeca, uma ficha é depositada no lugar ProduzPeca. De modo a iniciar o pré-processamento de uma peça, tanto o equipamento associado à produção da peça quanto o robô devem estar disponíveis. Caso a transição iniciaproducao dispare, o pré-processamento de uma peça (seja do *tipo*1 ou do *tipo*2) é iniciado. Após o disparo da transição iniciaproducao a ECPN Equipamento é invocada com o método Aloca, de modo a alocar o recurso P1 ou P2 para o pré-processamento de uma peça e para alocar o robô para mover a peça para a esteira assim que o pré-processamento termine. Observe que a ECPN Equipamento é invocada apenas se o sistema não está alocado para o processamen-

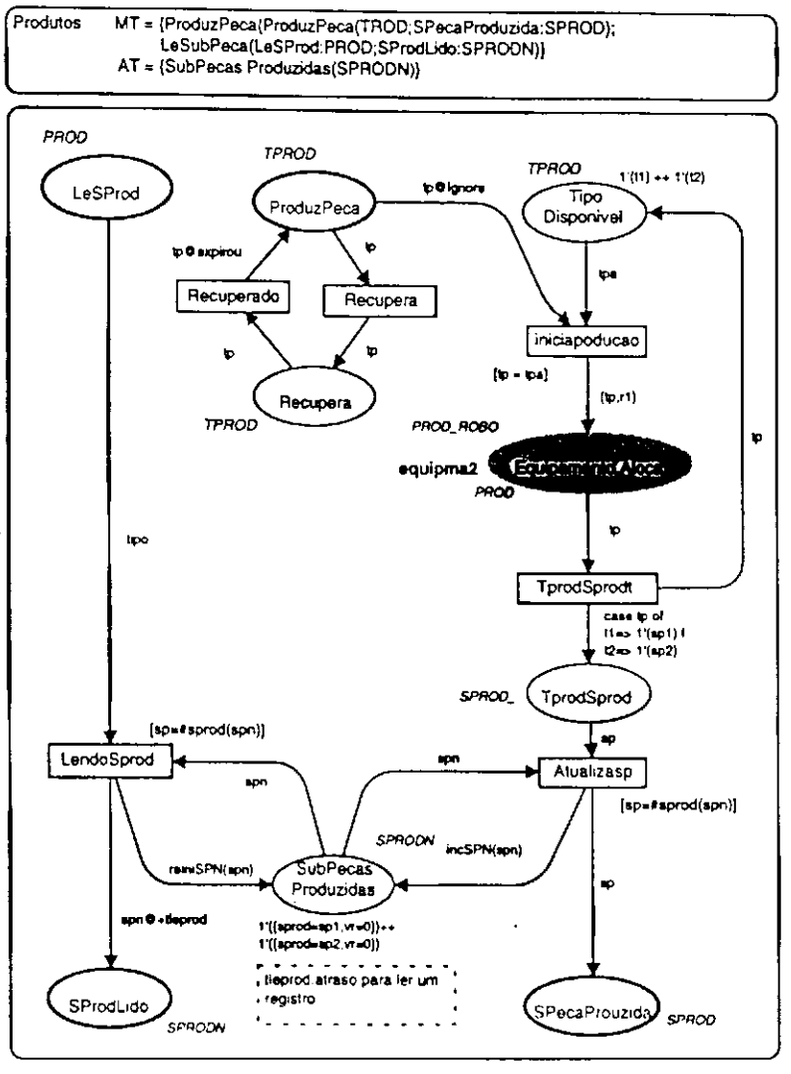

Figura 7.6: ECPN para o objeto Produtos

to do tipo específico indicado na invocação. Quando o resultado da ECPN Equipamento é retornado, a transição TprodSprodt pode ocorrer e disponibiliza o tipo novamente para processamento. Em seguida a transição Atualizasp também pode ocorrer. No caso de ocorrer, a ficha no lugar SubPecas Produzidas é atualizada e o objeto Produtos termina sua invocação.

Observe que quando a transição Atualizasp ocorre, uma ficha é removida do lugar SubPecas Produzidas, a qual mantém a informação relacionada com a peça pré-processada, uma função atualiza esta ficha e em seguida uma ficha atualizada é depositada no lugar SubPecas Produzidas. Uma ficha no lugar SPecaProduzida indica o fim da execução do método ProduzPeca.

De modo a considerar uma falha no sistema e disponibilizar o reinício da ECPN

Produtos, utiliza-se um *time-out*, modelado pela transição Recupera, lugar Recupera, e transição Recuperado. Caso o tempo máximo para que certo tipo de produto fique disponível ultrapasse o máximo definido a transição Recupera ocorre, removendo a ficha indicando a solicitação para produzir um determinado ítem do lugar ProduzPeca. Esta informação é então depositada no lugar Recupera e a transição Recuperado ocorre. Observe que a declaração @ignore associada ao arco entre o lugar ProduzPeca e a transição iniciaproducao indica que o rótulo de tempo associado às fichas no lugar ProduzPeca é ignorado para a ligação da variável tp.

Como dito anteriormente, o método LeSubPeca possibilita que o número de peças pré-processadas possa ser recuperado. Quando este método é invocado, uma ficha é depositada no lugar LeSProd, indicando o tipo a ser consultado. Quando a transição LendoSProd ocorre uma ficha com um registro indicando o tipo e a quantidade de sub produtos é removida do lugar SubPecas Produzidas e outra é colocada para aquele tipo com o valor zero. A função reiniSPN(spn) implementa tal reinicialização. Este registro é então retornado. Observe que modelamos um atraso para a disponibilização da ficha no lugar de terminação SProdLido. Este atraso é indicado no arco entre a transição LendoSprod e o lugar SProdLido e é spn@+tleprod.

Comportamento para o Objeto Câmera

O objeto Câmera modela a decisão a ser tomada, dependendo do tipo processado. O objeto Câmera, baseado no tipo de peça, decide a rota para o objeto. A ECPN para este objeto é apresentada na Figura 7.7. Como introduzido na Seção 7.2.1, dois métodos são definidos para este objeto: o método Identifica, que recebe o tipo da peça como argumento de entrada, e de acordo com o tipo, dispara a transição MoveTipo ou MoveRejNi. Se a peça for do *tipo*1 ou do *tipo*2 a transição MoveTipo dispara e o robô R2 é alocado para mover a peça da esteira para as máquinas M1 ou M2, respectivamente. Se a peça for do tipo *rejeitado* ou *não-reconhecida* a transição MoveRejNi dispara e o robô R2 é alocado para mover a peça para os depósitos de peças rejeitadas ou não-identificadas, respectivamente. O método LeTipo provê os meios pelos quais o número total de peças produzidas por tipo, peças não-identificadas ou rejeitadas possam ser acessados pelo objeto de leitura. Quando o método LeTipo é invocado, uma ficha é depositada no lugar LeTipoPROD, a transição LeTiposProduzidos dispara removendo uma ficha do lugar

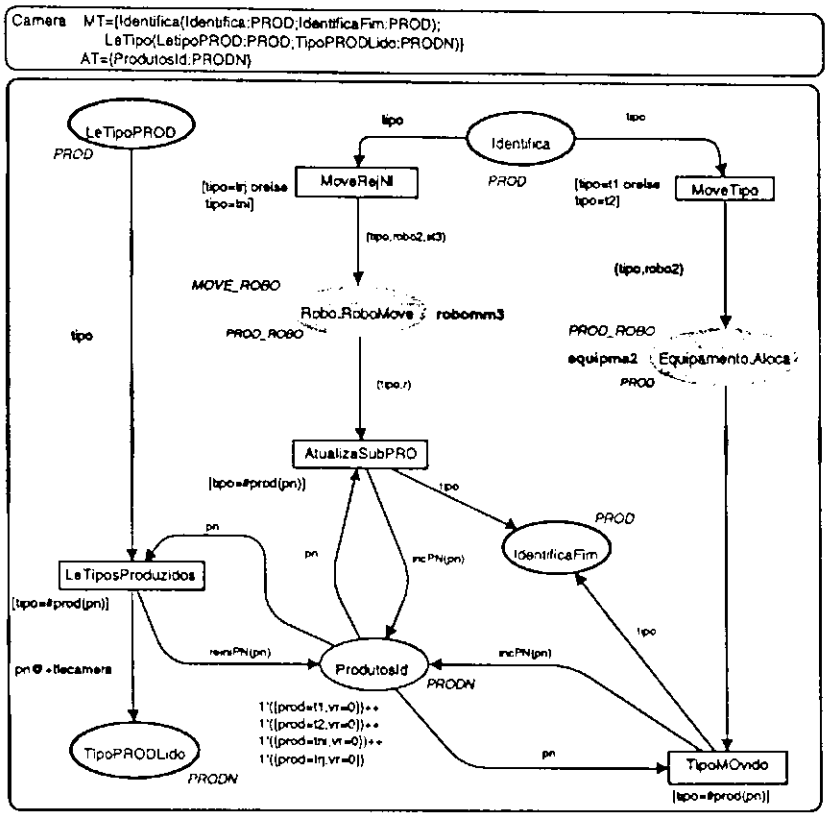

Figura 7.7: ECPN para o objeto Câmera

ProdutosId, correspondendo a um registro do tipo tipo definido. O registro no lugar ProdutosId contendo o tipo especificado é reinicializado (função reiniPN), e uma ficha contendo o registro lido é depositada no lugar de terminação TipoPRODLido. Observe que modelamos um atraso representando o tempo para recuperar o registro identificado pela constante tlecamera associada à inscrição do arco entre a transição LeTiposProduzidos e o lugar TipoPRODLido.

Comportamento para o Objeto de Banco de Dados OBDCel

Na Figura 7.8 mostra-se o objeto de banco de dados considerado para a célula modelada. Dois métodos são modelados para este objeto, o método AtualizaOBDC e o método LeOBDC. O atributo é definido pelo lugar OBDCel e seu tipo associado. Este lugar armazena os dados obtidos dos objetos de controle Produtos e Câmera.

Quando o método AtualizaOBDC é invocado, uma ficha definindo um registro para um dado tipo de peça é depositada no lugar AtualizaOBD. Em seguida, a transição Atualizando ocorre, o banco de dados OBDCelP é atualizado, e o método termina sua invocação

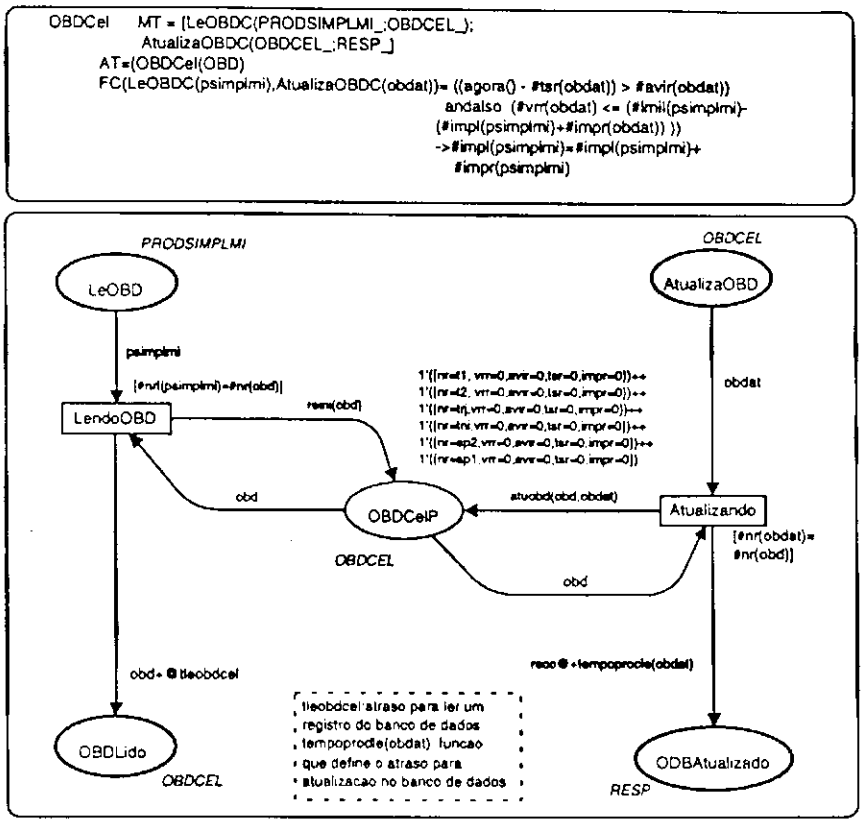

Figura 7.8: ECPN para o objeto OBDCel

depositando uma ficha no lugar ODBAtualizado. Para a atualização do banco de dados definimos um atraso associado à inscrição do arco entre a transição Atualizando e o lugar de terminação ODBAtualizado. Este atraso é determinado pela função tempoprocle(obdat). A necessidade de utilizar uma função foi devido ao fato de especificarmos atrasos diferentes para o caso de sub-produtos do *tipo1* e *tipo2*. Para os outros casos veja a função no Apêndice B.

Quando o método LeOBD é invocado, uma ficha contendo a peça a ser lida é colocada no lugar LeOBD, a transição LendoOBD dispara, a leitura no banco de dados é efetivada e o valor é reiniciado. Em seguida, o método termina sua invocação depositando uma ficha no lugar OBDLido. Para a leitura do banco de dados definimos um atraso associado à inscrição do arco entre a transição LendoOBD e o lugar de terminação OBDLido, este atraso é determinado pela constante tleobdcel.

Para este objeto define-se uma função de compatibilidade como:

```
FC(LeOBDC(psimplmi),AtualizaOBDC(obdat))=((agora()-#tsr(obdat)) > #avir(obdat))
   andalso  (#vrr(obdat) <= (#lmil(psimplmi)-(#impl(psimplmi)+#impr(obdat))))
                                    ->#impl(psimplmi)=#impl(psimplmi)+#vrr(obdat)
```

Esta função expressa uma negociação de consistência lógica por consistência temporal quando o método LeOBDC está sendo executado e o método AtualizaOBDC é invocado. Se a consistência temporal do item no banco de dados (lugar OBDCelP) foi violada, é importante permitir que o método AtualizaOBDC recupere a consistência temporal do item. No entanto, estes dois métodos só podem ser executados concorrentemente se o valor que está sendo atualizado em OBDCelP pelo método AtualizaOBDC (#vrr(obdat)), é menor ou igual ao limite máximo de imprecisão aceito pelo método LeOBDC. Esta determinação é baseada no limite máximo de imprecisão permitido para o argumento de retorno obdle do método LeOBDC (#lmil(psimplmi)), na imprecisão acumulada (#impl(psimplmi)), e na quantidade de imprecisão que o método AtualizaOBDC escreverá em OBDCelP através de obdat (#impr(obdat)). Ela também calcula a imprecisão resultante da execução concorrente dos métodos. Neste caso, o argumento de retorno do método LeOBDC pode ter um aumento na imprecisão igual a anterior mais a quantidade de imprecisão que é escrita em OBDCelP através de AtualizaOBDC (#impr(obdat)).

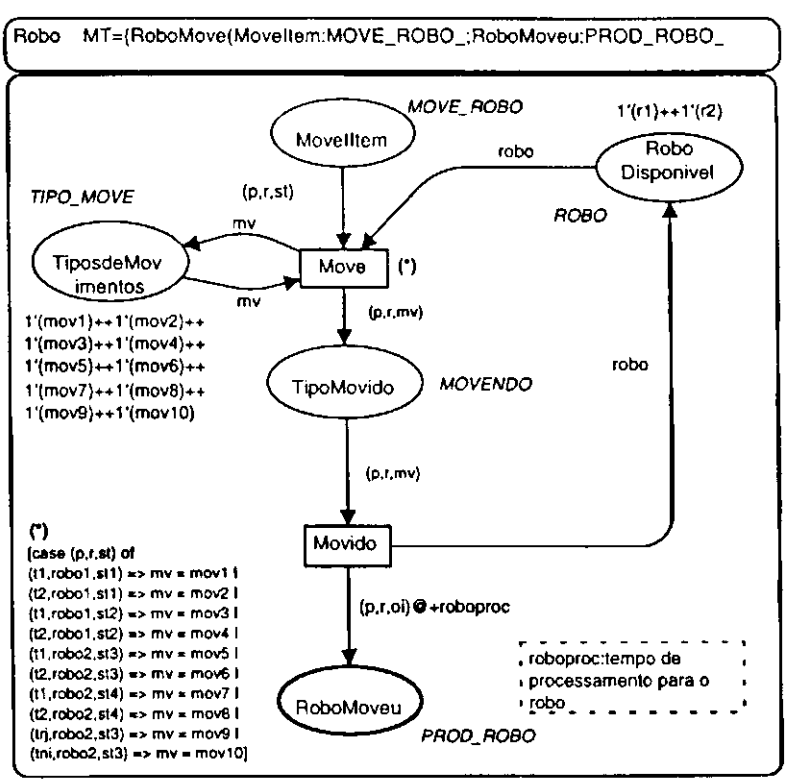

Figura 7.9: ECPN para o objeto Rôbo

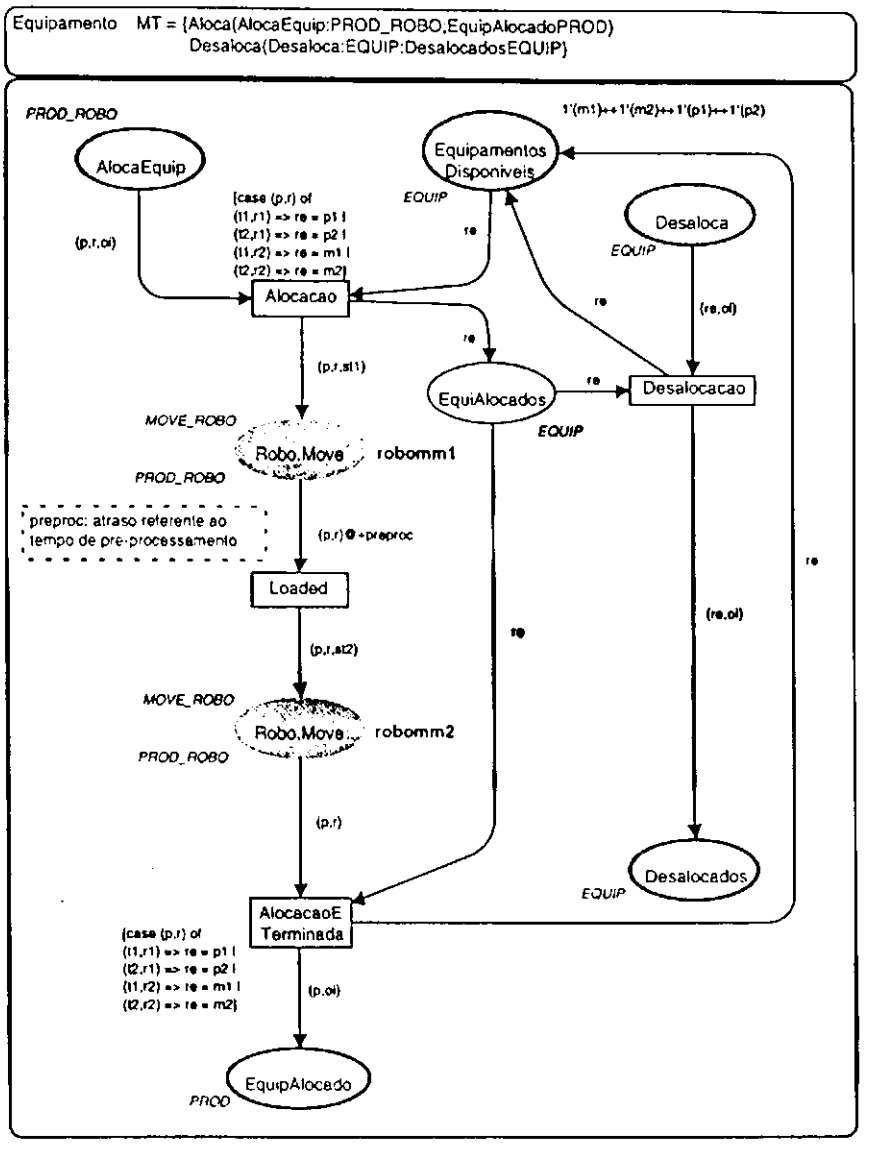

Figura 7.10: ECPN para o objeto Equipamento

Inicialmente o atributo OBDCelP possui uma marcação inicial definindo uma ficha para cada tipo de sub-produto (peça pré-processada) e produto (peça finalizada), ou seja, sp1 para o sub-produto do *tipo*1 e sp2 para sub-produto do *tipo*2, e t1 e t2 para produtos do *tipo*1 ou *tipo*2, respectivamente. Além disso, são disponibilizadas fichas para os itens rejeitados, trj, e não identificados, tni. Cada vez que os dados são lidos, tanto do objeto Produtos, quanto do objeto Camera, eles são adicionados aos valores prévios no campo correspondente para a ficha apropriada armazenada no lugar OBDCel.

Nas Figuras 7.9 e 7.10 são apresentados os modelos para os objetos Rôbo e Equipamento. O comportamento destes objetos encontra-se detalhado em [PTP98; PPC96].

7.3 Modelo para Transações do Banco de Dados

Como introduzido na Seção 7.1.5 as transações são modeladas por módulos ECPN. Para o exemplo da célula de manufatura modelamos três transações, como apresentado no modelo ECPN na Figura 7.11. Para este objeto, TransacoesOBDCel, somente um método é definido, InicializaTransacoes. Quando invocado o relógio global para as transações é inicializado, uma ficha 1'e é depositada no lugar Atualiza Banco de Dados.

As transações e seus parâmetros são:

1. *Atualização de quantidades de sub-produtos produzidos*: para esta transação temos: tempo de liberação $tl = 3$ unidades de tempo e periodicidade $pe = 12$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar AtualizaS Subprodutos temos 10 fichas (10'e), a marcação inicial para o *lugar de Ativação* (IniciaAS) é 1'terminada, e o peso do arco de entrada para a transição de ativação (AcessaSP SubProduto) é 12'e.

2. *Atualização de quantidades de produtos produzidos*: para esta transação temos: tempo de liberação $tl = 9$ unidades de tempo e periodicidade $pe = 18$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar AtualizaP Produtos temos 10 fichas (10'e), a marcação inicial para o *lugar de Ativação* (IniciaAP) é 1'terminada, e o peso do arco de entrada para a transição de ativação (AcessaP Produtos) é 18'e.

3. *Leitura do banco de dados*: para esta transação temos: tempo de liberação $tl = 36$ unidades de tempo, e; periodicidade $pe = 36$ unidades de tempo. A marcação inicial para o *Relogio_Local*, no lugar LeOBDCellT temos 1 ficha (1'e), a marcação inicial para o *lugar de Ativação* (IniciaLeOBD) é 1'terminada, e o peso do arco de entrada para a transição de ativação (AcessaOBD Produtos) é 36'e.

Observe que as transações podem executar concorrentemente cada operação de leitura, tanto dos objetos de controle Camera e Produtos, como leitura do banco de dados, OBDCel. Desta forma, as transações somente podem ser terminadas quando todas as instâncias terminam, por exemplo, para a transação de atualização de sub-produtos, a transição FAtualizouOBDSP Subprodutos, somente ocorre quando duas fichas 1'terminada estão disponíveis, veja a inscrição para o arco de entrada da transição, indicando que

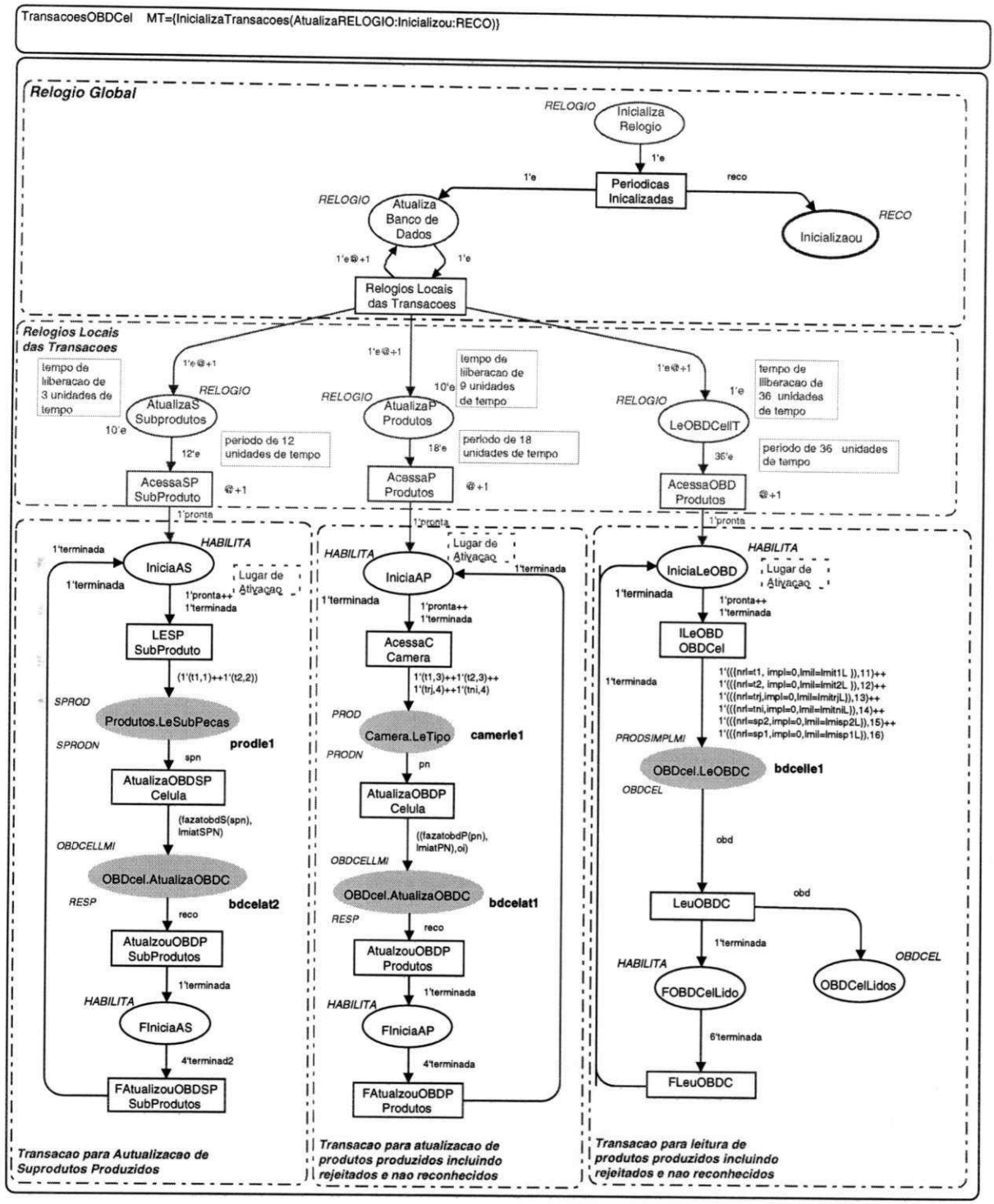

Figura 7.11: ECPN para o objeto AtualizaOBD

tanto o sub-produto sp1 como sp2 foram atualizados. O raciocínio para as outras duas transações é semelhante, observe a Figura 7.11. Os tempos de execução para estas tarefas serão obtidos a partir do procedimento de análise que introduziremos no Capítulo 8.

7.4 Conclusão

Neste capítulo apresentou-se um método orientado a objetos para modelagem de bancos de dados em tempo-real. O método é baseado na técnica OMT, no modelo RTSORAC, e em uma linguagem de redes de Petri baseada em objetos, denominada ECPN. A aplicação do método resulta na obtenção de um *modelo de objetos*, usado para modelar os aspectos estáticos do sistema, e um *modelo de processos*, usado para modelar os aspectos dinâmicos do sistema. O método suporta o tratamento de consistência lógica e temporal, e controle de concorrência semântico, com base em funções de compatibilidade definidas pelo projetista do sistema e declaradas na interface dos modelos para os objetos. Definirão-se também os passos necessários para a obtenção dos dois modelos. Finalmente, exemplificou-se a aplicação desse método ao domínio de aplicação considerado nesta tese, que são os sistemas flexíveis de manufatura.

Capítulo 8

Análise de Modelos ECPN

Neste capítulo apresentamos um procedimento para a análise de modelos ECPN. O procedimento é exemplificado com base no modelo para a célula de manufatura apresentado no Capítulo 7. Para a análise de um modelo ECPN, este é primeiramente transformado em uma rede de Petri Colorida Hierárquica (HCPN). Este modelo HCPN é então analisado utilizando cenários e diagramas de seqüências de mensagens, e através da geração e análise do espaço de estados para os cenários. O pacote de ferramentas Design/CPN [Ja96; Ja99] é utilizado em todo o processo de construção e análise do modelo.

8.1 Considerações para Análise de Modelos ECPN

A análise de um modelo ECPN tem o objetivo de verificar o comportamento lógico e temporal bem como auxiliar na correção da especificação obtida. Para analisar um modelo ECPN é fundamental dispor de um conjunto de ferramentas computacionais para edição, simulação e análise. Uma possível abordagem seria o desenvolvimento de uma ferramenta dedicada a tais propósitos. Entretanto, no contexto desta tese optou-se por utilizar o Design/CPN [Ja96; Ja99], uma vez que, devido a diversas ferramentas agregadas, como será introduzido na Seção 8.2, este satisfaz as necessidades para a construção e análise de modelos ECPN.

8.2 O Design/CPN

O Design/CPN é um pacote de ferramentas para o desenvolvimento de modelos de Redes de Petri Coloridas. O Design/CPN é constituído de quatro ferramentas computacionais integradas em um único pacote:

1. editor gráfico para construir, modificar e executar análise sintática de modelos CPN.

2. um simulador podendo operar simulação interativa ou automática, possibilitando ainda definir diferentes critérios para parada e observação da evolução da rede.

3. uma ferramenta para gerar e analisar grafos de ocorrência de modelos CPN.

4. uma ferramenta para análise de desempenho que possibilita a simulação e observação de dados de desempenho de modelos CPN.

O Design/CPN é um dos pacotes de ferramentas mais elaborados para construção, modificação, e análise de redes de Petri, e tem sido extensivamente utilizado com sucesso em diferentes domínios de aplicação. O leitor interessado em exemplos de diferentes aplicações pode consultar as seguintes referências [Jen98b; Jen98a; Jen99].

No contexto deste trabalho utilizou-se também uma biblioteca para a geração de diagramas de seqüência de mensagens, introduzidos no Capítulo 3 e contextualizados para a verificação de cenários (Seção 8.6.1). Por estar em fase experimental de desenvolvimento esta ferramenta ainda não está integrada ao Design/CPN [CdF99]. O leitor interessado em detalhes sobre o Design/CPN pode encontrar mais informações em http://www.daimi.au.dk/designCPN.

8.3 Introdução Informal a Redes de Petri Hierárquicas

Nesta seção introduzimos informalmente as redes de Petri Coloridas Hierárquicas (HCPN) [Jen92]. O objetivo é familiarizar o leitor com a nomenclatura utilizada em modelos HCPN. A motivação para definir as HCPNs é disponibilizar mecanismos para construir modelos através da combinação de um conjunto de CPNs denominadas *páginas*. Esta abordagem pode ser comparada à construção de um programa a partir de um conjunto de módulos e funções.

Do ponto de vista teórico uma HCPN possibilita descrever os mesmos tipos de sistemas que uma CPN não hierárquica e, portanto, são modelos equivalentes. É sempre possível transformar uma HCPN em uma CPN não hierárquica. Contudo, as HCPNs disponibilizam ao projetista, mecanismos de abstração que permitem descrever de forma mais organizada modelos de sistemas complexos. Considerando que no caso de sistemas complexos, um dos grandes problemas enfrentados por desenvolvedores de sistemas atualmente é lidar com muitos detalhes ao mesmo tempo, as HCPNs disponibilizam mecanismos de abstração que permitem ao projetista lidar e manter o foco em uma determinada parte de um modelo de cada vez.

Modelos HCPN são implementados utilizando-se os conceitos de *lugares de fusão* e *transições de substituição*. Lugares de fusão são estruturas que permitem especificar um conjunto de lugares como funcionalmente um único lugar, isto é, se uma ficha é removida ou adicionada em um dos lugares, uma ficha idêntica é adicionada ou removida dos outros lugares pertencentes ao conjunto. Um conjunto de lugares de fusão é denominado de conjunto de fusão (*fusion set*).

Uma transição de substituição pode ser vista como uma transição de mais alto nível que se relaciona a uma CPN mais complexa que descreve com mais detalhes as atividades modeladas pela transição de substituição. A página que contém a transição de substituição é denominada de *superpágina* da CPN mais detalhada correspondente, que por sua vez é denominada de *subpágina*. Cada transição de substituição é denominada de *supernó* da subpágina correspondente.

Uma transição de substituição se relaciona com sua subpágina através da utilização de um tipo de conjunto de fusão de dois membros denominados portas e *sockets*. Estas estruturas descrevem a interface entre a transição de substituição e a subpágina. *Sockets* são atribuídos aos lugares conectados à transição de substituição e portas são associadas a determinados lugares na subpágina tal que um par *socket*/porta formam um conjunto de fusão. Dessa forma, quando uma ficha é depositada num *socket*, ela aparece também na porta associada àquele *socket*, permitindo assim a conexão entre a superpágina e a subpágina. Cada *socket* pode ser associado a uma ou mais portas, e uma porta pode ser associada somente a um *socket*.

Como dito anteriormente, é sempre possível traduzir uma rede de Petri hierárquica para sua correspondente não hierárquica. Para isso, basta substituir cada transição de

substituição e arcos conectados, por sua respectiva subpágina, "colando" cada *socket* com sua respectiva porta. Detalhes teóricos relacionados a HCPNs podem ser encontrados em [Jen92].

8.4 Transformação de um Sistema ECPN em uma HCPN

Nesta Seção introduzimos um procedimento para a transformação de um sistema ECPN em uma HCPN. O procedimento de transformação é ilustrado na Figura 8.1. No que segue definimos os algoritmos para transformar um sistema ECPN em uma HCPN. Pode-se observar que primeiramente constrói-se o modelo para o sincronizador. Para cada invocador definido nos módulos ECPN define-se os lugares superiores e inferiores. Por fim, compõem-se os modelos CPN para os módulos com o modelo CPN para o sincronizador. No segue apresentamos os algoritmos para este processo de transformação.

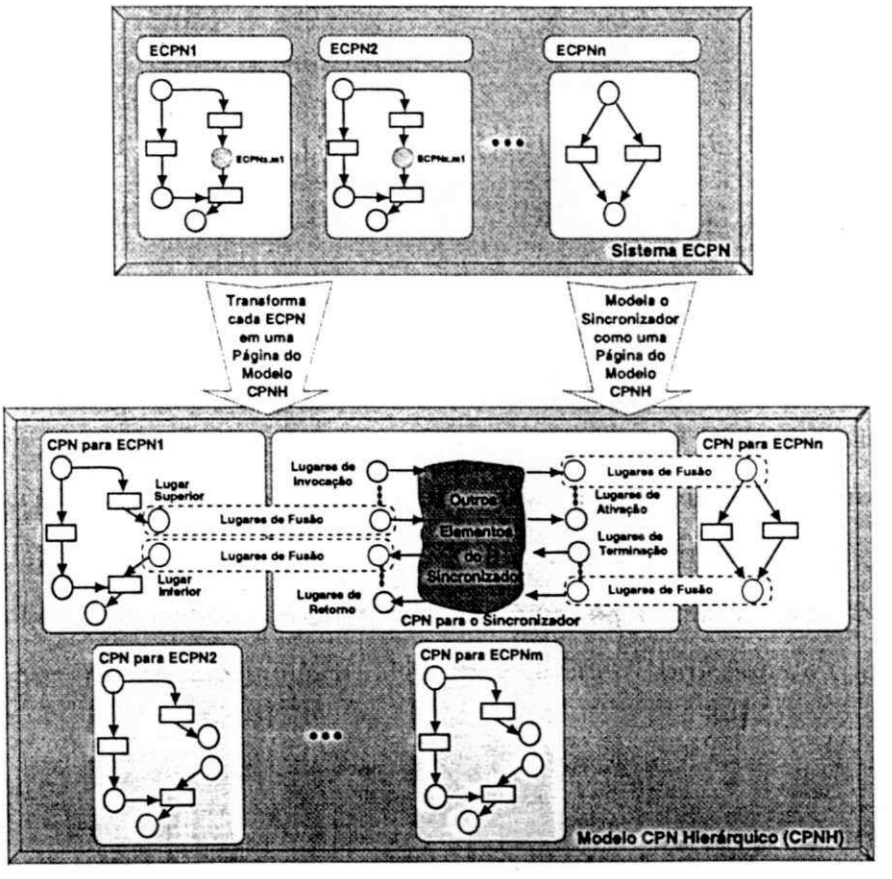

Figura 8.1: Transformação de um modelo ECPN em HCPN

No detalhamento para os algoritmos apresentados o leitor deve observar a Figura 8.3 para visualizar os resultados de alguns dos passos do algoritmo. Alguns detalhes foram omitidos a fim de evitar que a figura ficasse ilegível. Deve-se ainda observar que a notação utilizada nos algoritmos segue a definida no Capítulo 6.

Algoritmo 8.1 Transforma um sistema ECPN (SE) em uma HCPN ($\mathsf{HCPN_{SE}}$)

1: *Sincronizador* ← *Cria_Página*
2: *Declarações*←*Cria_Página*
{Novas páginas para o sincronizador e nó de declaração global}
3: *Declarações*← $\bigcup_{i=0}^{\#SE} E_i.\Sigma$
{Define o conjunto de cores para a HCPN no nó de declarações}
4: *Sincronizador* ← *CriaLugar*(*Ativações*)
5: *Sincronizador* ← *CriaLugar*(*Retornos*)
6: *Sincronizador* ← *CriaLugar*(*Gerador*)
7: *Sincronizador* ← *CriaLugar*(*Associações*)
{Cria lugares para ativações, retornos, gerador de contexto e associações de contexto}
8: $C(Ativações) \leftarrow \bigcup_{i=0}^{\#SE} \bigcup_{j=0}^{\#(E_i.INV_j)} C(LS(E_i.INV_j))$
{A cor do lugar *Ativações* é a união das cores dos lugares superiores dos módulos}
9: $C(Retornos) \leftarrow \bigcup_{i=0}^{\#SE} \bigcup_{j=0}^{\#(E_i.INV_j)} C(LI(E_i.INV_j))$
{A cor do lugar *Retornos* é a união das cores dos lugares inferiores dos módulos}
10: **para todo** $E_i \in SE$ **faça**
11: *CriaInvocador*(E_i,*Sincronizador*)
12: *CriaAtivador*(E_i,*Sincronizador*)
13: *CriaTerminador*(E_i,*Sincronizador*)
14: *CriaRetornos*(E_i,*Sincronizador*)
15: **fim para**

O Algoritmo 8.1 define a transformação de um sistema ECPN, denotado por SE, em uma HCPN, denotada por $\mathsf{HCPN_{SE}}$.

No Passo 1 cria-se uma nova página para construir a página correspondente ao sincronizador do modelo HCPN, denotado por *Sincronizador*.

No Passo 2 cria-se uma nova página para o nó de declaração global do modelo $\mathsf{HCPN_{SE}}$.

No Passo 3 associa-se a esta página a união dos conjuntos de cores definidos para cada um dos módulos ECPN pertencentes ao sistema ECPN SE. Observe que nos algoritmos #SE denota a cardinalidade do conjunto SE, isto é, o número de módulos ECPN no sistema SE.

Nos Passos 4, 5, 6 e 7 são criados no *Sincronizador* os lugares referentes às ativações, retornos, gerador de contexto e associação de contextos. Na Figura 8.3 correspondem

aos lugares Ativacoes, Retornos, Gerador de Contexto, e Associacoes de Contexto.

Nos Passos 8 e 9 definem-se os conjuntos de cores para os lugares *Ativações* e *Retornos*. Observe que pela Definição 6.18 (restrição 3, página 96, Capítulo 6) os conjuntos de cores para estes lugares são consistentemente definidos. No caso da Figura 8.3 estes dois conjuntos de cores correspondem as cores ATIVACAOS_ e RETORNOS_. A cor ATIVACAOS_, (veja o Apêndice B), é definida como um produto das cores OBJ, ATIVA e OCC. A cor OBJ, relacionada com a função objetivo (OBJ, Definição 6.4, página 87, Capítulo 6), define um conjunto enumerado de nomes de métodos para todos os objetos do sistema ECPN. Neste caso somente são definidos aqueles que são invocados no sistema, ou seja, robomm correspondendo a invocação Robo.RoboMove; equipma correspondendo a invocação Equipamento.Aloca; bdcelle correspondendo a invocação OBDCel.LeOBDC; bdcelat correspondendo a invocação OBDCel.AtualizaOBDC; camerle correspondendo a invocação Camera.LeTipo; prodle correspondendo a invocação Produtos.LeSubPecas. Observe que as invocações Camera.Identifica e Produtos.ProduzPeca não estão neste enumerados pois correspondem a invocações por agentes externos que não foram modelados. A cor ATIVA é a união das cores definidas para os lugares superiores dos invocadores. Veja por exemplo o lugar EquipmaLS2S, cuja cor é PROD_ROBO_. Para o modelo HCPN o _ sufixado ao nome da cor definido para a ECPN original indica que à cor original acrescentou-se o contexto. A cor OCC define o contexto, cujo domínio são os números inteiros.

Neste algoritmo omitiu-se a definição dos conjuntos de cores para os lugares *Gerador de Contexto* e *Associações de Contexto*. Na Figura 8.3, a cor do lugar Gerador de Contexto é OCC e a cor do lugar Associacoes de Contextos é INVOCC2 a qual é definida como um produto das cores INV, OCC, e OCC. A cor INV define um conjunto enumerado de nomes que identifica de forma única os invocadores, neste caso são: robomm1, robomm2, robomm3, equipma1, equipma2, bdcelle1 bdcelat1, bdcelat2, camerle1, e prodle1. A primeira cor OCC é definida para o contexto da ECPN invocadora, e a segunda cor OCC para o contexto gerado para a ECPN invocada.

Entre os Passos 10 e 15 são criados os invocadores, ativadores, terminadores e retornos. No que segue detalhamos cada um destes algoritmos.

8.4.1 Cria Invocadores no Sincronizador

Algoritmo 8.2 CriaInvocador(E_i,*Sincronizador*)

1: para todo $LS(inv_j) \in E_i.INV$ faça
2: $Sincronizador \leftarrow CriaLugar(LSSIN_{jE_i})$
{Lugar correspondente ao lugar superior definido para o invocador}
3: $C(LSSIN_{jE_i}) \leftarrow C(LS(inv_j))$
{Define a cor do lugar criado como a cor do lugar superior do invocador}
4: Cria um conjunto de fusão $CFLS_{jE_i}$
5: $CFLS_{jE_i} \leftarrow LS(inv_j)$
6: $CFLS_{jE_i} \leftarrow LSSIN_{jE_i}$
{Associa $LS(inv_j)$ e $LSSIN_{jE_i}$ ao conjunto de fusão $CFLS_{jE_i}$}
7: $Sincronizador \leftarrow CriaTransição(Invocação_{jE_i})$
{Cria uma transição de invocação}
8: Estabelece a relação $(LSSIN_{jE_i}, Invocação_{jE_i})$
9: Estabelece a relação $(Invocação_{jE_i}, Ativação)$
10: Estabelece a relação $(Invocação_{jE_i}, Associações)$
11: Estabelece a relação $(Gerador, Invocação_{jE_i})$
12: Estabelece a relação $(Invocação_{jE_i}, Gerador)$
13: fim para

O Algoritmo 8.2 estabelece como criar no *Sincronizador* a parte do modelo para cada uma das invocações definidas pelos invocadores para todos os módulos ECPN do sistema.

No Passo 2 é criado um lugar ($LSSIN_{jE_i}$) que corresponde ao lugar superior definido para cada um dos invocadores $LS(inv_j)$ de um módulo E_i[1]. Na Figura 8.3, um exemplo para ($LSSIN_{jE_i}$) é o lugar EquipmaLS2S[2], que corresponde ao lugar EquipmaLS2 para o modelo mostrado na Figura 8.4.

No Passo 3 define-se a cor do lugar $LSSIN_{jE_i}$ como a cor do lugar superior do invocador j, ou seja, $C(LS(inv_j))$. Para os mesmos lugares citados anteriormente, nas Figuras 8.3 e 8.4, a cor é PROD_ROBO_ (veja o Apêndice B).

Nos passos 4, 5, e 6 define-se um conjunto de fusão e associam-se os lugares $LSSIN_{jE_i}$ e $LS(inv_j)$ ao mesmo. Nas Figuras 8.3 e 8.4 está indicado por um FG ao lado dos lugares exemplificados.

No Passo 7 criam-se as transições modelando as invocações do método. Na Figura 8.3 correspondem as transições rotuladas por $invocacao_i$.

Nos passos 8, 9, 10, 11 e 12 são estabelecidas as relações entre os lugares e as

[1]A função *LS* define o lugar superior para um invocador em um dado módulo (veja Definição 6.4, página 87, Capítulo 6).

[2]O quinto lugar, de cima para baixo, no canto superior esquerdo da Figura 8.3.

transições para as invocações. Na Figura 8.3 estão indicados cada um destes passos. Neste algoritmo não estão explicitadas as inscrições associadas às relações estabelecidas.

8.4.2 Cria Retornos no Sincronizador

O Algoritmo 8.3 estabelece como criar no *Sincronizador* a parte do modelo para cada um dos retornos pendentes definidos para todos os módulos ECPN do sistema. Seu funcionamento é similar ao Algoritmo 8.2. Na Figura 8.3 o leitor pode observar indicações sobre os resultados de cada um dos passos.

Algoritmo 8.3 CriaRetornos(E_i,*Sincronizador*)

1: para todo $LI(inv_j) \in E_i.INV$ faça
2: $Sincronizador \leftarrow CriaLugar(LISIN_{jE_i})$
{Lugar correspondente ao lugar inferior definido para o invocador}
3: $C(LISIN_{jE_i}) \leftarrow C(LI(inv_j))$
{Define a cor do lugar criado como a cor do lugar inferior do invocador}
4: Cria um conjunto de fusão $CFLI_{jE_i}$
5: $CFLI_{jE_i} \leftarrow LI(inv_j)$
6: $CFLI_{jE_i} \leftarrow LISIN_{jE_i}$
{Associa $LI(inv_j)$ e $LISIN_{jE_i}$ ao conjunto de fusão $CFLI_{jE_i}$}
7: $Sincronizador \leftarrow CriaTransição(Retorno_{jE_i})$
{Cria uma transição de retorno}
8: Estabelece a relação $(Retorno_{jE_i}, LISIN_{jE_i})$
9: Estabelece a relação $(Retornos, Retorno_{jE_i})$
10: Estabelece a relação $(Associações, Retorno_{jE_i})$
11: fim para

8.4.3 Cria Ativações no Sincronizador

O Algoritmo 8.4 estabelece como criar no *Sincronizador* a parte do modelo para cada uma das ativações pendentes. Seu funcionamento é similar ao Algoritmo 8.2. Na Figura 8.3 o leitor pode observar indicações sobre os resultados de cada um dos passos.

8.4.4 Cria Terminações no Sincronizador

O Algoritmo 8.5 estabelece como criar no *Sincronizador* a parte do modelo para cada uma das terminações definidas para os métodos para cada módulo ECPN. Seu funcionamento é similar ao Algoritmo 8.2. Na Figura 8.3 o leitor pode observar indicações sobre os resultados de cada um dos passos.

Algoritmo 8.4 CriaAtivações(E_i,*Sincronizador*)

1: **para todo** $FI(mt_j) \in E_i.MT$ **faça**
2: $Sincronizador \leftarrow CriaLugar(LASIN_{jE_i})$
{Lugar correspondente ao lugar de iniciação para o método}
3: $C(LASIN_{jE_i}) \leftarrow C(FI(mt_j))$
{Define a cor do lugar criado como a cor do lugar inicialização}
4: Cria um conjunto de fusão $CFLI_{jE_i}$
5: $CFLI_{jE_i} \leftarrow FI(mt_j)$
6: $CFLI_{jE_i} \leftarrow LASIN_{jE_i}$
{Associa $FI(inv_j)$ e $LASIN_{jE_i}$ ao conjunto de fusão $CFLI_{jE_i}$}
7: $Sincronizador \leftarrow CriaTransição(Ativação_{jE_i})$
{Cria uma transição de ativação}
8: Estabelece a relação $(Ativações, Ativação_{jE_i})$
9: Estabelece a relação $(Ativação_{jE_i}, LASIN_{jE_i})$
10: **fim para**

Algoritmo 8.5 CriaTerminações(E_i,*Sincronizador*)

1: **para todo** $FT(mt_j) \in E_i.MT$ **faça**
2: $Sincronizador \leftarrow CriaLugar(LTSIN_{jE_i})$
{Lugar correspondente ao lugar de terminação para o método}
3: $C(LTSIN_{jE_i}) \leftarrow C(FT(mt_j))$
{Define a cor do lugar criado como a cor do lugar de terminação}
4: Cria um conjunto de fusão $CFLT_{jE_i}$
5: $CFLT_{jE_i} \leftarrow FT(mt_j)$
6: $CFLT_{jE_i} \leftarrow LTSIN_{jE_i}$
{Associa $FT(inv_j)$ e $LTSIN_{jE_i}$ ao conjunto de fusão $CFLT_{jE_i}$}
7: $Sincronizador \leftarrow CriaTransição(Terminação_{jE_i})$
{Cria uma transição de terminação}
8: Estabelece a relação $(LTSIN_{jE_i}, Terminação_{jE_i})$
9: Estabelece a relação $(Terminação_{jE_i}, Retornos)$
10: **fim para**

8.5 Modelo HCPN para uma Célula de Manufatura

Nesta seção exemplificamos a transformação do sistema ECPN para a célula de manufatura apresentada no Capítulo 7. Na Figura 8.2 apresenta-se a página de hierarquia para o modelo HCPN. Nesta página estão identificadas as páginas referentes aos módulos ECPN, quais sejam: Produtos, Equipamento, Robo, e Camera, definindo os objetos de controle; Transacoes e OBDCel, referentes aos objetos de banco de dados, e; Sincronizador e modelo para a função de compatibilidade definida para o objeto OBDCel. A página para o Sincronizador resulta da aplicação dos passos definidos pelos algoritmos introduzidos na Seção 8.4. Além destas, há mais quatro páginas: Declaracoes, Resultados, Controle, e Performance. A página Declaracoes contém as declarações de cores (tipos de dados), variáveis, e funções utilizadas no modelo (vide Apêndice B). As outras três páginas são auxiliares para visualizar resultados de análise do espaço de estados (grafo de ocorrência), Resultados. Na página Controle há funções utilizadas para simulação e definição de critérios de parada para a construção do grafo de ocorrência, e a página Performance contém as definições e funções utilizadas pela ferramenta para análise de desempenho do Design/CPN. Detalhes sobre estas páginas são omitidos desta tese.

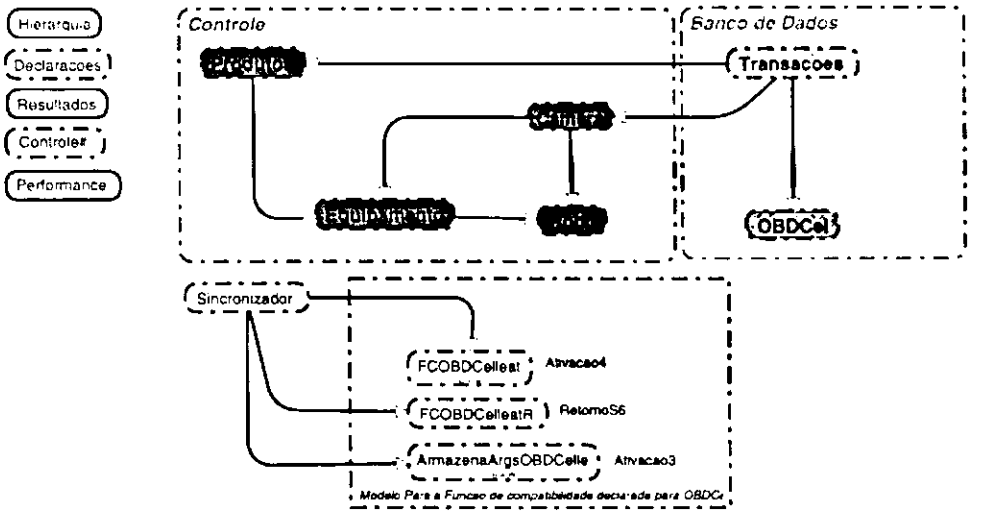

Figura 8.2: Hierarquia para o modelo HCPN

Na Figura 8.4 apresenta-se a página referente ao objeto Produtos. Observe que, foram introduzidos, além da representação explícita dos lugares superiores e inferiores para o invocador (EquipmaLS2 e EquipmaLI2), uma transição (Define Temporização) e um lugar (Temporiza ProduzPeca). Uma vez que o objeto Produtos é invocado por um agente

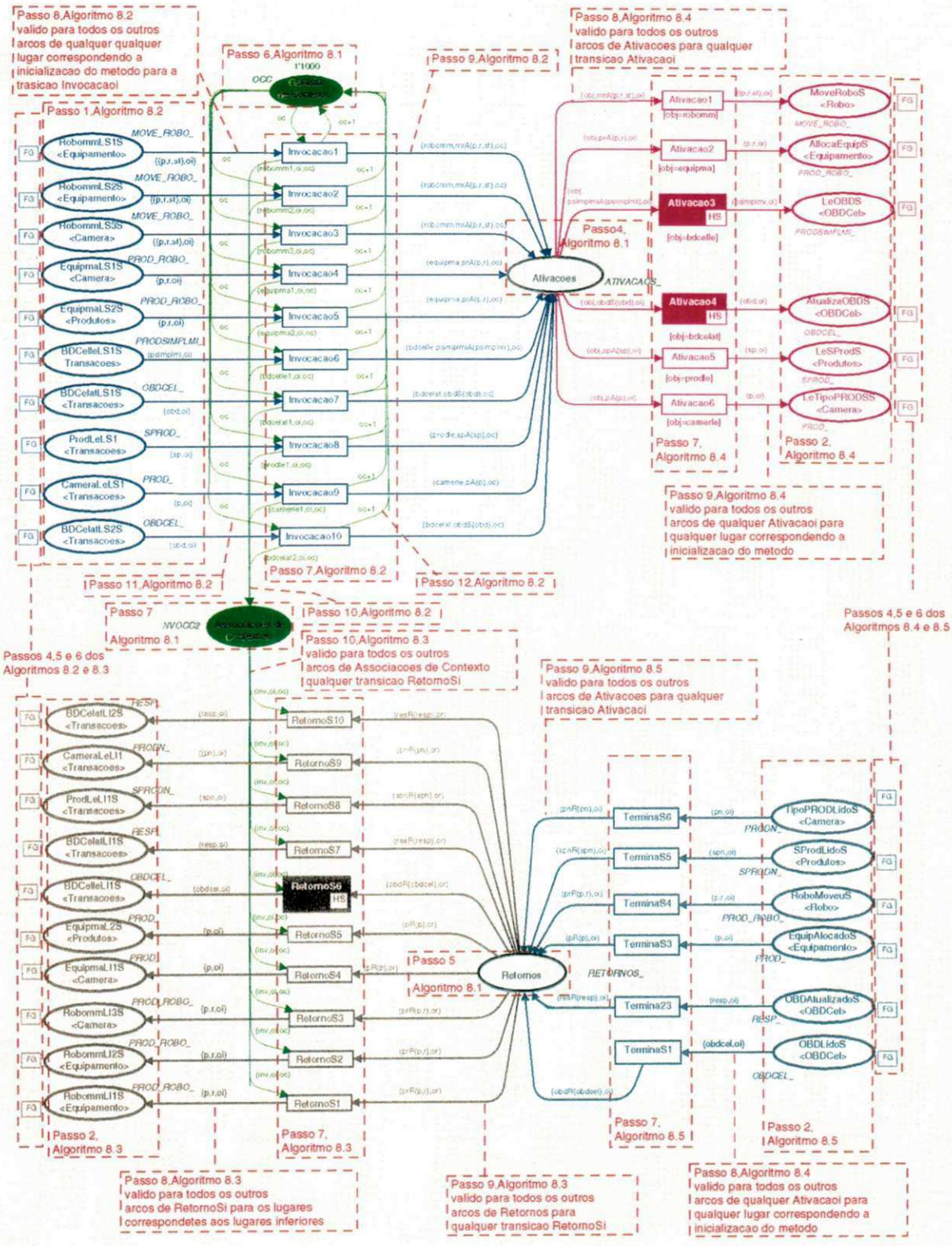

Figura 8.3: Modelo CPN para o sincronizador

externo para inicializar a produção de peças, e este agente não foi modelado, este lugar e esta transição modelam tal invocação, através de fichas no lugar Temporiza ProduzPeca. O comportamento deste modelo pode ser compreendido a partir da descrição incluída no Capítulo 7. Observe também que para alguns dos arcos o contexto foi declarado explicitamente, veja por exemplo a variável oi na inscrição do arco entre a transição Iniciaproducao e o lugar EquipmaLS2.

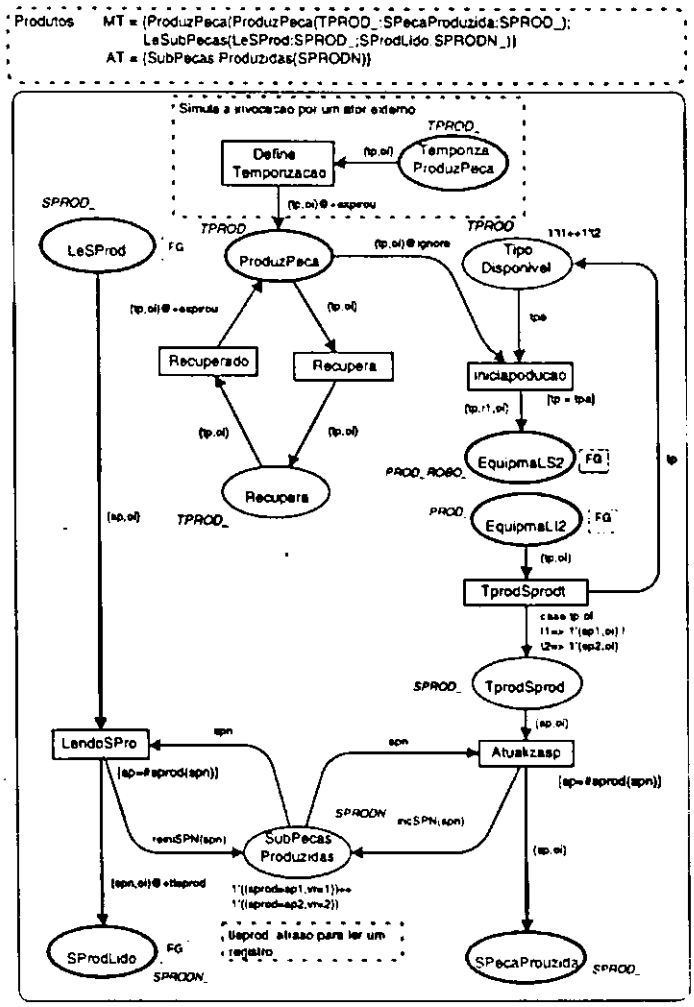

Figura 8.4: Modelo CPN para a ECPN Produtos

No Apêndice A são apresentadas às páginas restantes para o modelo HCPN resultante da transformação de um modelo ECPN apresentado no Capítulo 7. No Apêndice B apresenta-se os conjuntos de cores, variáveis e funções declaradas para o modelo HCPN.

8.6 Análise do Modelo HCPN

Como ilustrado na Figura 8.5, a análise do modelo HCPN é feita em dois passos: análise de cenários e análise de escalonamento.

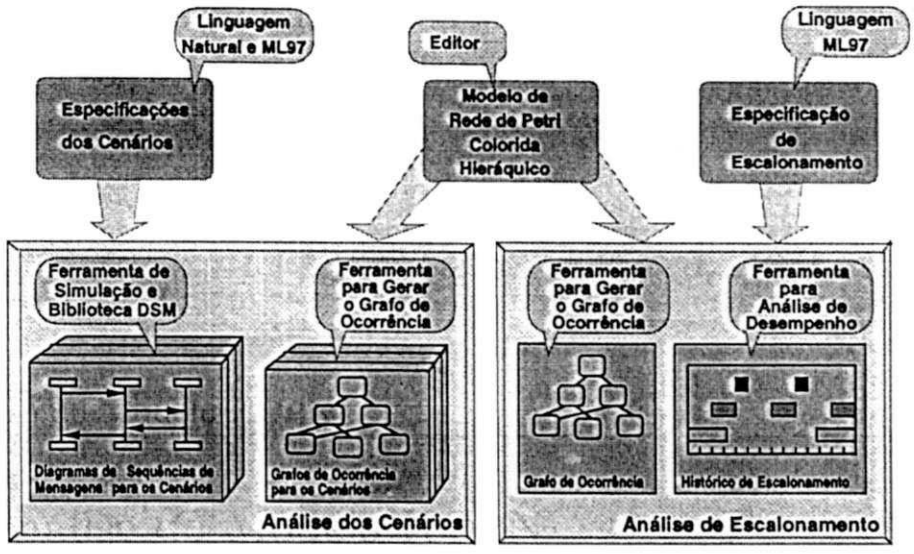

Figura 8.5: Análise do modelo HCPN

No primeiro passo são definidos os cenários. Os cenários serão representados através de *diagramas de seqüências de mensagens.* Para a análise do modelo foram considerados diversos cenários. Para cada um obteve-se, através de simulação do modelo HCPN, os diagramas de seqüência de mensagens entre os objetos, introduzidos na Seção 8.6.1. Uma vez que estes diagramas são obtidos considerando apenas uma possível seqüência de execução, para cada um dos cenários foi construído o espaço de estados, no caso o grafo de ocorrência, e verificado se os resultados particulares para uma das seqüências de execução eram válidos para todas a seqüências de execução. Na seção 8.6.1 detalham-se os cenários e a análise.

Para os diagramas de seqüências de mensagens apresentados nas seções seguintes a notação utilizada é apresentada na Figura 8.6.

No segundo passo executa-se a análise de escalonamento para as transações com prazos estritos.

8.6.1 Cenários para Análise

Os cenários para análise utilizados foram:

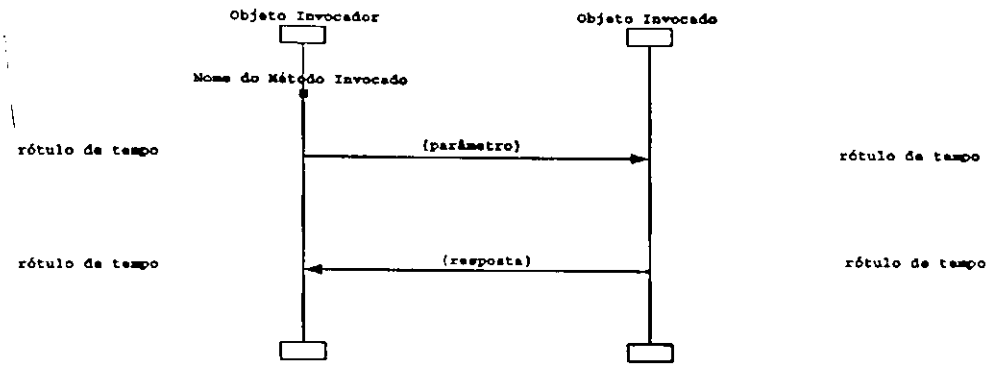

Figura 8.6: Notação para diagramas de seqüência de mensagens entre os objetos

1. *Produzir uma Peça*: produzir uma peça do tipo 1 e uma peça do tipo 2.

2. *Atualização de Subprodutos*: atualizar o número de subprodutos produzidos no objeto de banco de dados OBDCel executando a transação AtualizaS Subprodutos.

3. *Atualização de Produtos*: atualizar o número de produtos produzidos, incluindo os rejeitados e não identificados, no objeto de banco de dados OBDCel executando a transação AtualizaP Produtos.

4. *Leitura do Banco de Dados*: executar a leitura do objeto de banco de dados OBDCel executando a transação LeOBDCellT.

Cenário: Produzir uma Peça

Para analisar a produção de peças o seguinte cenário foi utilizado. Uma ordem de produção é definida inicialmente, uma ficha contendo o tipo de peça a produzir e um contexto inicial. Observe que para fins de análise o gerador de contexto para o sincronizador é iniciado com o valor 1000, deste modo o valor Gerador de Contexto deve ser maior ou igual a 1000. Consideramos duas situações para este cenário: produzir uma peça do tipo 1 e uma do tipo 2. Então a marcação inicial do lugar Temporiza Produz Peca para o objeto Produtos é 1'(t1,1) e 1'(t2,2), respectivamente. Para o objeto Camera definimos um temporizador com um atraso de 6 unidades de tempo para a identificação do objeto. Desta forma, após seis unidades de tempo o tipo especificado é então processado. Para este cenário definimos a marcação do lugar IdentificaT para o objeto Camera como 1'(t1,3) e 1'(t2,4), para o caso de identificação de uma peça do tipo 1 ou tipo 2, respectivamente.

Os diagramas de seqüências de mensagens para este cenário estão apresentados nas Figuras 8.7 e 8.8, respectivamente.

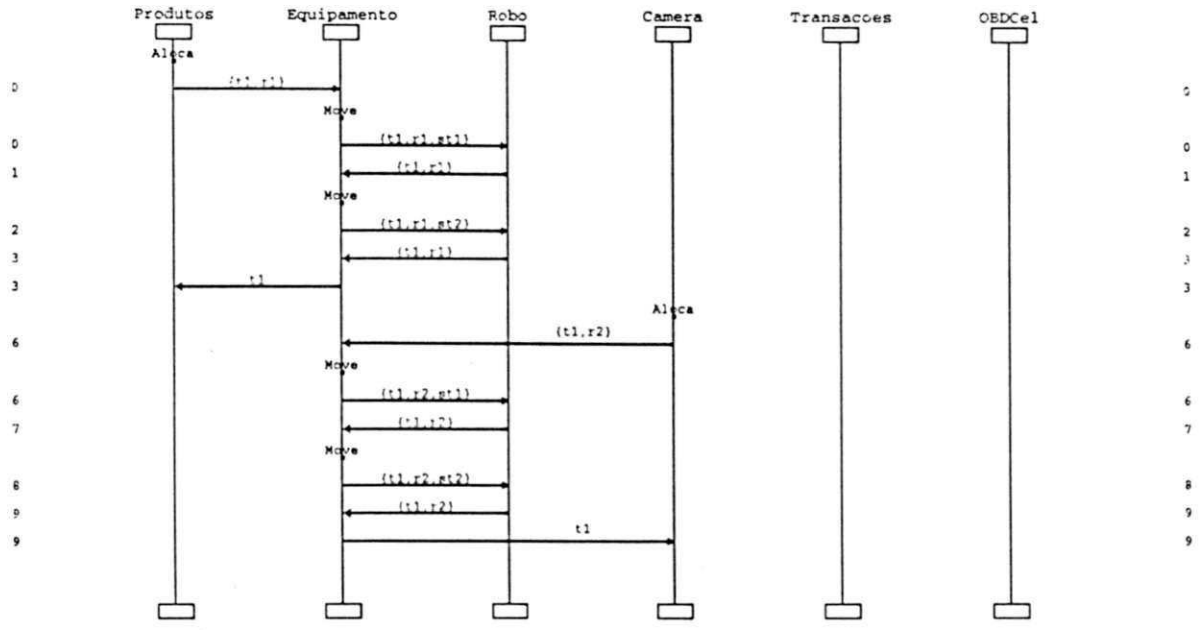

Figura 8.7: Diagrama de seqüência de mensagens para produzir um item do tipo 1

O grafo de ocorrência para ambos os cenários possui 128 nós, e a marcação final indica a produção correta de uma peça do tipo 1 e uma do tipo 2. Na marcação final temos:

1. a marcação do lugar SubPecas Produzidas para o objeto Produtos alcançada foi:

   - 1'{sprod=sp1,vr=1})++(1'{sprod=sp2,vr=0}), para o cenário de produzir uma peça do tipo 1 e,
   - 1'{sprod=sp1,vr=0})++(1'{sprod=sp2,vr=1}), para o cenário produção de uma peça do tipo 2.

2. a marcação do lugar ProdutosId para o objeto Camera alcançada foi:

   - 1'{prod=t1,vr=1})++(1'{prod=t2,vr=0})++ (1'{prod=tni,vr=0})++(1'{prod=trj,vr=0}), para o cenário de produzir uma peça do tipo 1 e,
   - 1'{prod=t1,vr=0})++(1'{prod=t2,vr=1})++ (1'{prod=tni,vr=0})++(1'{prod=trj,vr=0}), para o cenário de produzir uma peça do tipo 2.

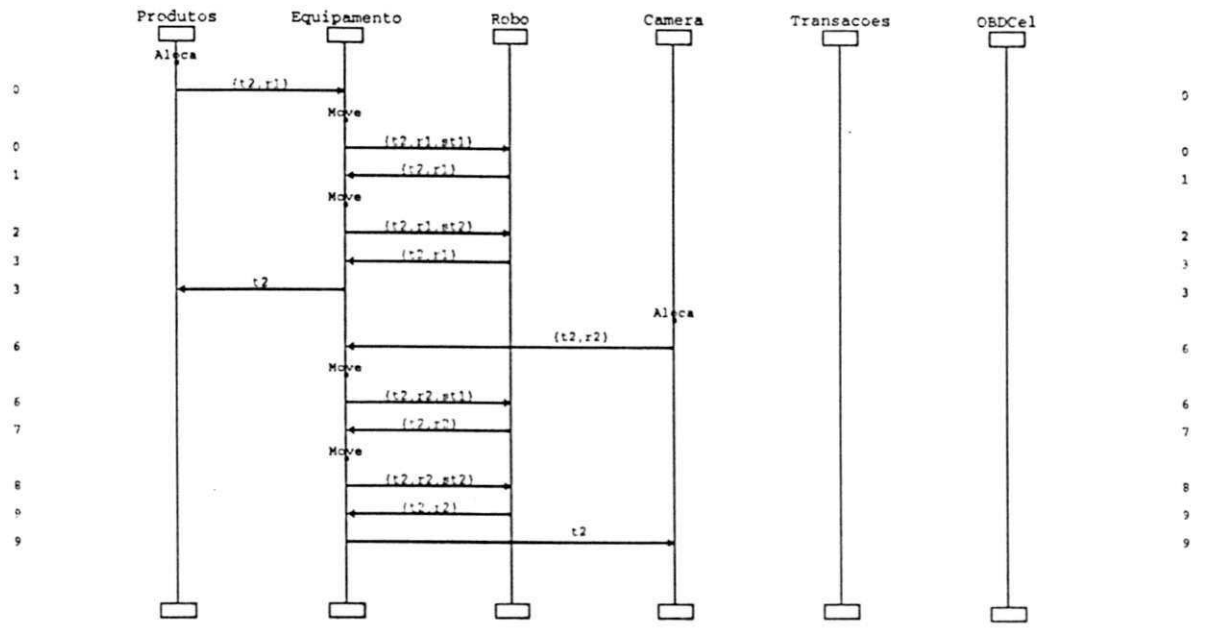

Figura 8.8: Diagrama de seqüência de mensagens para produzir um item do tipo 2

Observe nos diagramas de seqüências de mensagens que os tempos totais para produzir peças foram ambos de 9 unidades de tempo. A partir das marcações alcançadas em (1) e (2) conclui-se que o comportamento para a produção de peças está correto.

Cenário: Atualização de Subprodutos

Para analisar a transação AtualizaS Subprodutos o seguinte cenário foi considerado:

- marcação inicial do lugar SubPecas Produzidas para o objeto Produtos indicando 1 subpeça do tipo 1 e 2 subpeças do tipo 2, ou seja: 1'{sprod=sp1,vr=1})++(1'{sprod=sp2,vr=2}).

- marcação inicial do lugar IniciaAS (atualização de subprodutos) para o objeto Transação indicando que a transação para atualização da quantidade de subprodutos no objeto de banco de dados OBDCel está pronta para ser executada, ou seja: 1'terminada++1'pronta.

Ainda, a marcação para o objeto de banco de dados OBDCel, para o lugar OBDCelP no estado inicial:

1'{nr=t1,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=t2,vrr=0,avir=0,tsr=0,impr=0}++
1'{nr=trj,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=tni,vrr=0,avir=0,tsr=0,impr=0}++
1'{nr=sp1,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=sp2,vrr=0,avir=0,tsr=0,impr=0}

Para este cenário consideramos a execução da transação para atualização no banco de dados da quantidade de subprodutos produzidos. Na Figura 8.9 apresentamos o diagrama de seqüência de mensagens para a execução da transação AtualizaS Subprodutos. No instante de tempo 0 o objeto Transações invocou duas vezes o método LeSprod do objeto Produtos. As invocações estão indicadas pela marca e pelo rótulo na linha de execução para o objeto Transações, e duas mensagens foram enviadas uma indicando a leitura da quantidade de subprodutos do tipo 1, sp1, e outra para subprodutos do tipo 2, sp2. No instante de tempo 1, indicando um tempo de processamento de 1 unidade de tempo, duas mensagens foram enviadas pelo objeto Produtos para o objeto Transações, indicando o fim das execuções das duas invocações do método LeSprod, e retornando os valores lidos, no caso 1 unidade para subprodutos do tipo 1, sp1[1], e 2 unidades para subprodutos do tipo 2, sp2[2]. Ainda no instante de tempo 1, o objeto Transações invocou duas vezes o objeto OBDCel para atualizar o banco de dados com os valores lidos, indicado por dois marcadores AtualizaOBD. Por fim o objeto OBDCel indica o fim da atualização, representado por duas mensagens enviadas para o objeto Transações, com o rótulo Atualizado.

O grafo de ocorrência para este cenário possui 132 nós. A marcação final alcançada para o lugar OBDCelP do objeto de banco de dados OBDCel foi:

1'{nr=t1,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=t2,vrr=0,avir=0,tsr=0,impr=0}++
1'{nr=trj,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=tni,vrr=0,avir=0,tsr=0,impr=0}++
1'{nr=sp1,vrr=1,avir=0,tsr=0,impr=0}++1'{nr=sp2,vrr=2,avir=10,tsr=1,impr=0}

Observe no diagrama de seqüência de mensagens para esta transação que o tempo de liberação é de 3 unidades de tempo e que o tempo total para executar a transação foi de 3 unidades de tempo, o último rótulo é 6 unidades de tempo (6-3=3). Observe ainda que a segunda execução da transação indicada no diagrama, ocorreu no instante de tempo

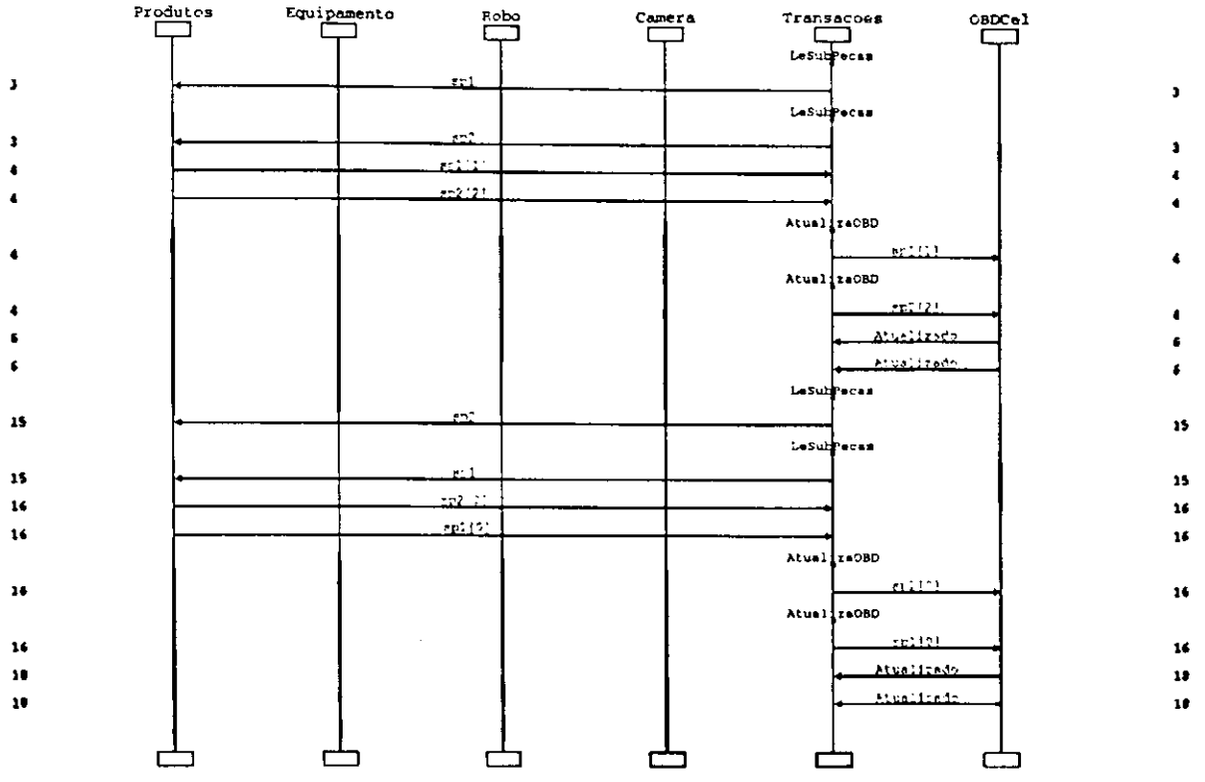

Figura 8.9: Diagrama de seqüência de mensagens para a execução da transação AtualizaS Subprodutos

15, ou seja, a periodicidade de 12 unidades de tempo como especificado. A análise do grafo de ocorrência dos diagramas de seqüências de mensagens para este cenário possibilitam concluir que transação executou como especificado (lógica e temporalmente) a atualização do objeto de banco de dados OBDCel.

Cenário: Atualização de Produtos

Para analisar a transação AtualizaP Produtos o seguinte cenário foi considerado:

- marcação inicial do lugar ProdutosId para o objeto Camera indicando 21 peças do tipo 1, 22 peças do tipo 2, 23 peças não reconhecidas, e 24 peças rejeitadas, ou seja é: 1'{prod=t1,vr=21})++(1'{prod=t2,vr=22})++ (1'{prod=tni,vr=23})++(1'{prod=trj,vr=24}).

- marcação inicial do lugar IniciaAP para o objeto Transação indicando que a transação para atualização da quantidade de produtos no objeto de banco de dados OBDCel está pronta para ser executada, ou seja é : 1'terminada++1'pronta.

Ainda, a marcação para o objeto de banco de dados OBDCel, para o lugar OBDCelP no estado inicial:

1'{nr=t1,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=t2,vrr=0,avir=0,tsr=0,impr=0}++
1'{nr=trj,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=tni,vrr=0,avir=0,tsr=0,impr=0}++
1'{nr=sp1,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=sp2,vrr=0,avir=0,tsr=0,impr=0}

Para este cenário consideramos a execução da transação para atualização no banco de dados da quantidade de produtos produzidos. Na Figura 8.10 apresentamos o diagrama de seqüência de mensagens para a execução da transação AtualizaP Produtos. No instante de tempo 0 o objeto Transações invocou quatro vezes o método LeTipo (lê itens) do objeto Camera, indicados pela marca e pelo rótulo na linha de execução para o objeto Transações, e quatro mensagens foram enviadas, uma indicando a leitura da quantidade de produtos do tipo 1, t1, uma para produtos do tipo 2, t2, uma para peças não reconhecidas, tni, e uma para peças rejeitadas, trj. No instante de tempo 1, indicando um tempo de processamento de 1 unidade de tempo, quatro mensagens foram enviadas pelo objeto Camera para o objeto Transações, indicando o fim das execuções das invocações do método LeTipo, e retornando os valores lidos, no caso, 21 unidades para produtos do tipo 1, t1[21], 22 para produtos do tipo 2, t2[22], 23 para peças não reconhecidas, tni[23], e 24 para peças rejeitadas, trj[24]. Ainda no instante de tempo 1, o objeto Transações invocou quatro vezes o objeto OBDCel para atualizar o banco de dados com os valores lidos, indicado por quatro marcadores e quatro rótulos AtualizaOBDC. Por fim, o objeto OBDCel indica o fim das atualizações, através das quatro mensagens enviadas para o objeto Transações, com o rótulo Atualizado.

O grafo de ocorrência para este cenário possui 33858 nós. A marcação final alcançada para o lugar OBDCelP do objeto de banco de dados OBDCel foi:

1'{nr=t1,vrr=21,avir=10,tsr=1,impr=0}++1'{nr=t2,vrr=22,avir=10,tsr=1,impr=0}
1'{nr=trj,vrr=24,avir=10,tsr=1,impr=0}++1'{nr=tni,vrr=23,avir=10,tsr=1,impr=0}++
1'{nr=sp1,vrr=0,avir=0,tsr=0,impr=0}++1'{nr=sp2,vrr=0,avir=0,tsr=0,impr=0}

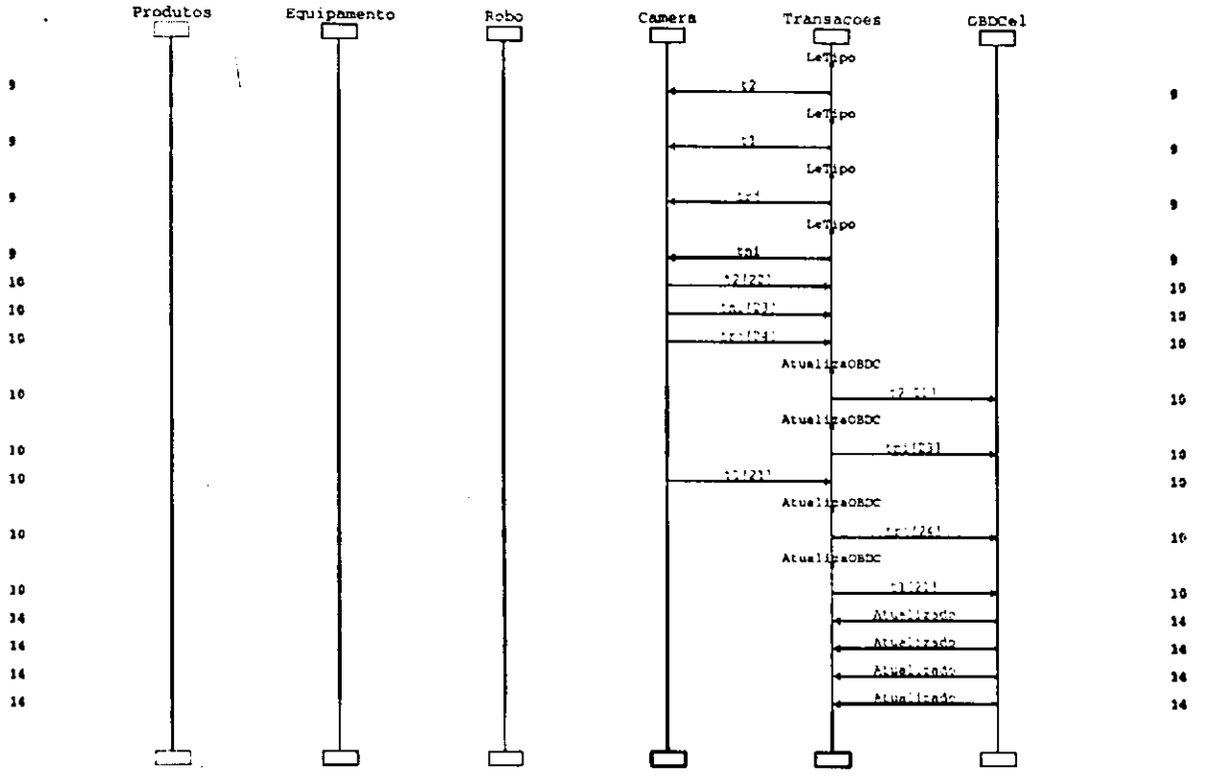

Figura 8.10: Diagrama de seqüência de mensagens para a execução da transação AtualizaP Produtos

Observe no diagrama de seqüência de mensagens para esta transação que o tempo de liberação é de 9 unidades de tempo e que o tempo total para executar a transação foi de 5 unidades de tempo, o último rótulo é 14 unidades de tempo (14-9=5). A análise do grafo de ocorrência e dos diagramas de seqüências de mensagens para este cenário possibilitam concluir que transação executou como especificado (lógica e temporalmente) a atualização do objeto de banco de dados OBDCel.

Cenário: Leitura do Banco de Dados

O procedimento de análise para este cenário é o mesmo para os apresentados anteriormente. É importante observar no diagrama de seqüência de mensagens para esta transação que o tempo de liberação é de 36 unidades de tempo, e o tempo total de execução é de 6 unidades de tempo (42-36=6). Para este caso o grafo de ocorrência possui 67986 nós e a sua análise, bem como dos diagramas de seqüência de mensagens, possibilitaram concluir que a transação executou como especificado.

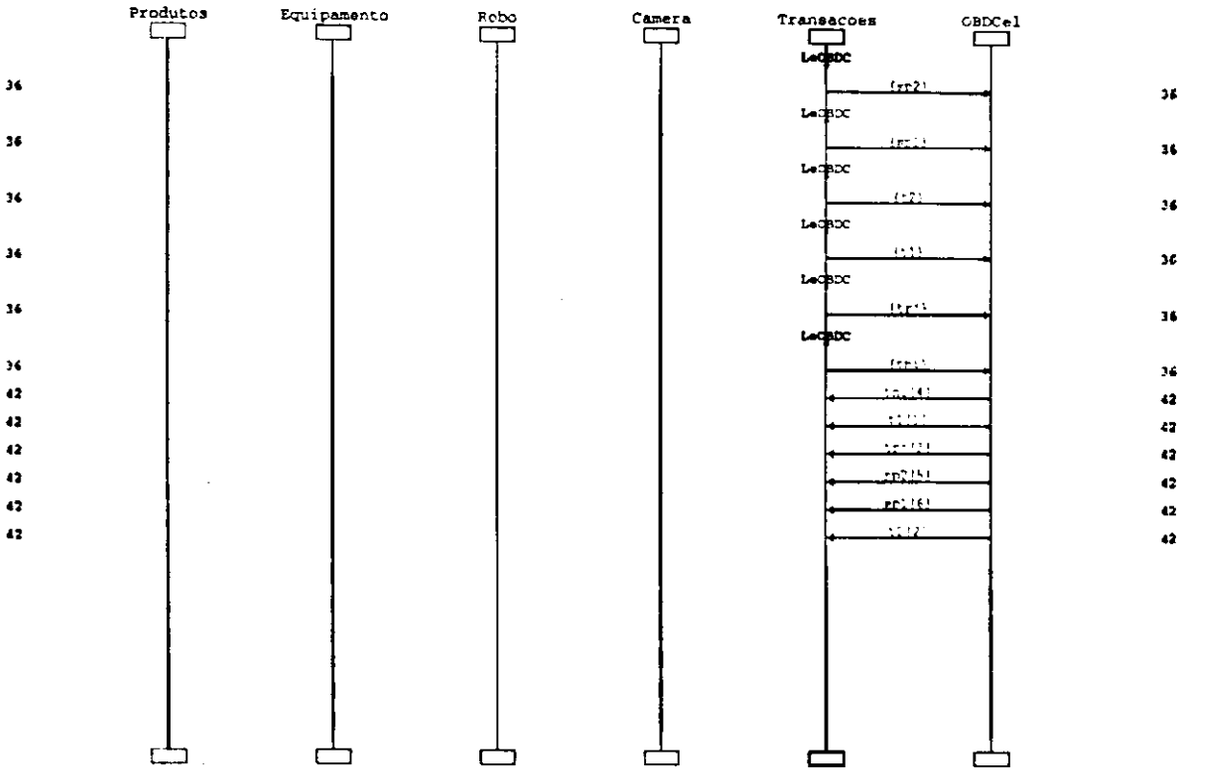

Figura 8.11: Diagrama de seqüência de mensagens para a execução da transação Le Produtos

8.6.2 Análise de Escalonamento

Como apresentado no Capítulo 7 para o objeto Transações são definidas três sub-transações:

1. *Atualização de quantidades de subprodutos produzidos*: para esta transação temos: tempo de liberação $tl = 3$ unidades de tempo; a periodicidade $pe = 12$ unidades de tempo, e; o tempo de execução $tc = 3$ unidades de tempo. A marcação inicial para o *Relógio_Local*, no lugar AtualizaS Subprodutos temos 10 fichas (10'e), a marcação inicial para o *lugar de Ativação* (IniciaAS) é 1'terminada, e o peso do arco de entrada para a transição de ativação (AcessaSP SubProduto) é 12'e.

2. *Atualização de quantidades de produtos produzidos*: para esta transação temos: tempo de liberação $tl = 9$ unidades de tempo; a periodicidade $pe = 18$ unidades de tempo, e; o tempo de execução $tc = 4$ unidades de tempo. A marcação inicial para o *Relógio_Local*, no lugar AtualizaP Produtos temos 10 fichas (10'e), a marcação

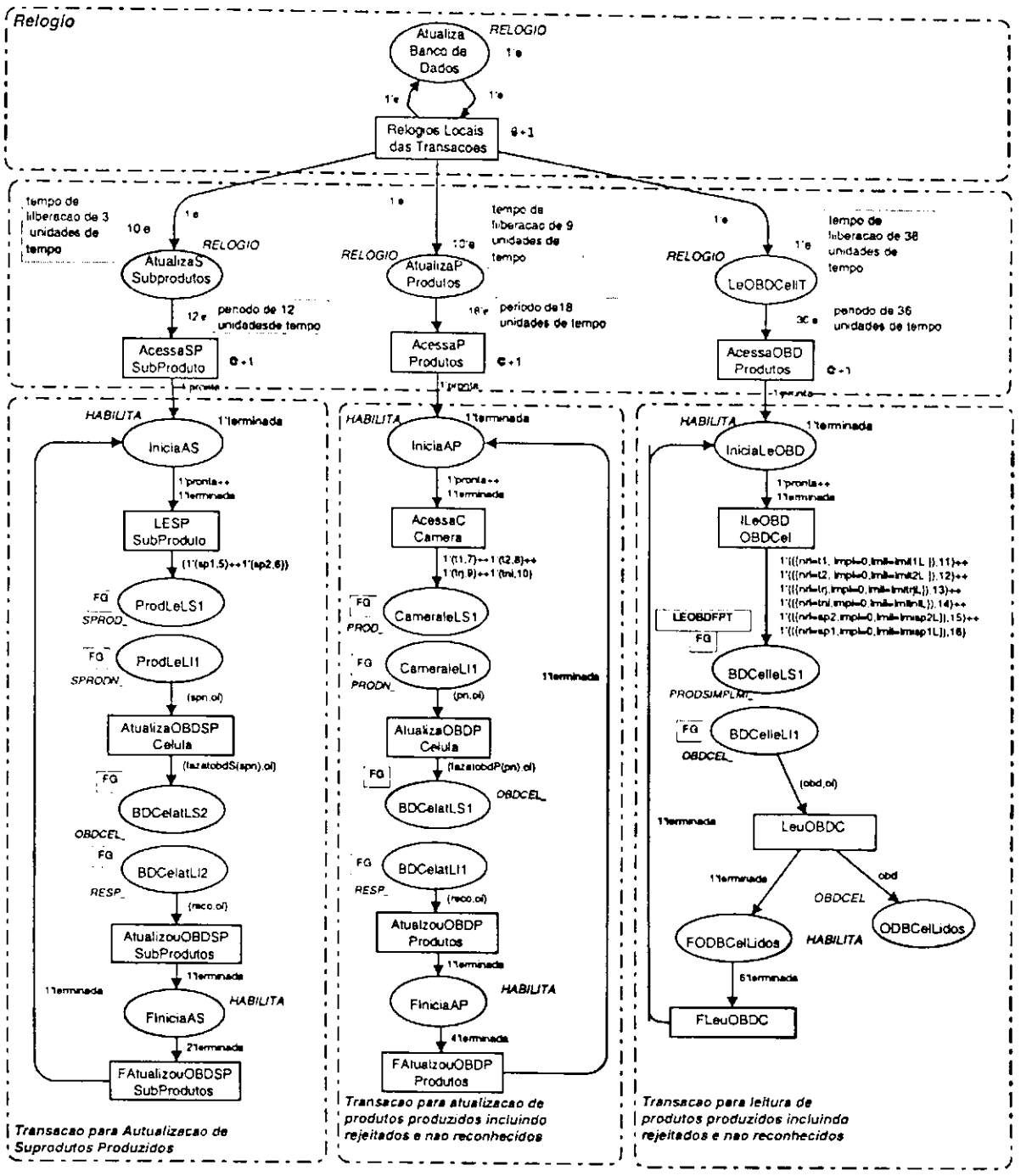

Figura 8.12: Modelo CPN para as transações

inicial para o *lugar de Ativação* (IniciaAP) é 1'terminada, e o peso do arco de entrada para a transição de ativação (AcessaP Produtos) é 18'e.

3. *Leitura do banco de dados:* para esta transação temos: tempo de liberação $tl = 36$ unidades de tempo; a periodicidade $pe = 36$ unidades de tempo, e; o tempo de execução $tc = 6$ unidades de tempo. A marcação inicial para o *Relógio_Local*, no lugar LeOBDCellT temos 1 ficha (1'e), a marcação inicial para o *lugar de Ativação* (IniciaLeOBD) é 1'terminada, e o peso do arco de entrada para a transição de ativação (AcessaOBD Produtos) é 36'e.

A página do modelo HCPN que corresponde a ECPN Transacoes é apresentada na Figura 8.12.

O algoritmo 8.6 detalha o escalonamento. Inicialmente constrói-se o grafo de ocorrência para o modelo HCPN. A seguir pesquisa-se se as marcações terminais são alcançáveis. Uma marcação terminal é definida pela existência de uma ficha 1'terminada em uma dada marcação, seguida, em algum estado posterior, de uma marcação 1'terminada++1'pronta. Isto significa que uma dada transação foi executada dentro do seu prazo e alcançou um estado em que pode ser novamente executada. Determinadas às marcações terminais para todas as transações, encontra-se a menor seqüência de ocorrência de transições, no caso de haver mais de uma, que transforma a marcação inicial nas marcações terminais para todas as transações. Tal seqüência define um escalonamento realizável para o conjunto de transações. O algoritmo foi implementado como uma função escrita em ML97 utilizando as facilidades para análise do Design/CPN.

Algoritmo 8.6 Escalona Transações com Prazo Estrito

```
1: Constrói o Grafo de Ocorrência
2: para todo Marcação∈ Grafo de Ocorrência faça
3:    Verifica se Marcação Terminal é alcançável
4:    se Marcação Terminal alcançável então
5:       Constrói as seqüências de ocorrências que levam a Marcação Terminal
6:       Encontra a menor seqüência de ocorrência
7:    senão
8:       Escalonamento não é possível para nenhuma seqüência de ocorrência
9:    fim se
10: fim para
```

É importante observar que para definir a seqüência, ou seqüências de escalonamento, utilizamos o grafo de ocorrência considerando a execução concorrente tanto dos métodos

definidos para os objetos de controle como de banco de dados.

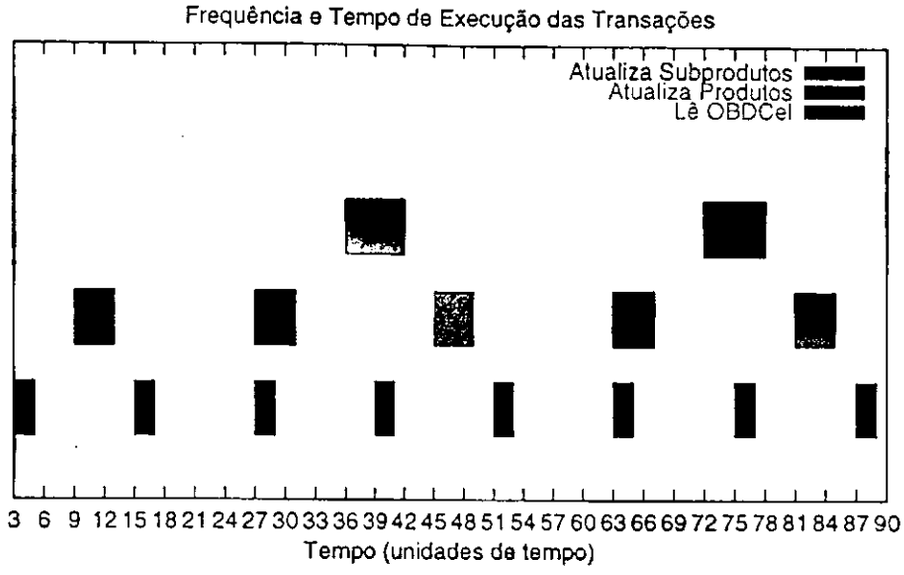

Figura 8.13: Diagrama de tempo para as transações

Na Figura 8.13 apresentamos o diagrama de tempo para a execução das transações. Este diagrama foi obtido utilizando a ferramenta para análise de desempenho do Design/CPN. Os dados foram obtidos através de simulação, e pode-se observar que as restrições temporais das transações foram satisfeitas. Observe que para o eixo indicando o tempo inicia com o valor de 3 unidades de tempo, isto se deve ao fato de que a transação para atualização de quantidades de subprodutos produzidos está definida com um tempo de liberação 3.

8.7 Conclusão

Neste Capítulo introduzimos um procedimento de análise para modelos ECPN que baseia-se na transformação de um tal modelo em uma HCPN. Para analisar esta HCPN resultante utilizamos o Design/CPN, que é um conjunto de ferramentas para edição e análise de modelos CPN. Introduzimos algoritmos que determinam como mapear um modelo ECPN para o HCPN e ilustramos a aplicação deles utilizando o exemplo da célula de manufatura introduzida no Capítulo 7. Para a análise do modelo HCPN utilizamos diversos cenários e os mais significativos foram apresentados. Para cada um dos cenários foram gerados e analisados diagramas de seqüências de mensagens e o espaço

de estados. Por fim discutimos como determinar o escalonamento para as transações com prazos estritos como introduzido no Capítulo 7.

Capítulo 9

Conclusão e Perspectivas

Neste capítulo apresentamos as conclusões da pesquisa conduzida. Para tanto elaboramos inicialmente uma reflexão com uma perspectiva global sobre os resultados obtidos na solução dos problemas declarados. Além disso, discorremos sobre as contribuições mais relevantes desta tese bem como definimos encaminhamentos para os desdobramentos e pesquisas futuras a serem desenvolvidas.

9.1 Contribuições Centrais da Tese

A pesquisa apresentada e desenvolvida nesta tese teve como objetivo central definir um método com base formal para a modelagem de BDTR. Além disso, detalhou-se a aplicação do método a um problema no domínio de sistemas flexíveis de manufatura.

Com o desenvolvimento desta tese contribuiu-se para a área de modelagem e análise de sistemas complexos de software, num contexto particular de aplicações em tempo-real. A partir dos requisitos necessários aos SGBD-TR e do estudo de alguns modelos de BDTR, identificamos as características desejáveis para um método de modelagem de BDTR. Desta forma, estabelecemos um método de modelagem, incluindo mecanismos de alto-nível, que proporcionam ao projetista facilidades para a descrição de BDTR.

Introduzimos um modelo de redes de Petri baseado em objetos denominado ECPN, incluindo as definições sintáticas e semânticas para tal modelo. Assim, definimos os conceitos necessários ao estabelecimento de uma base formal possibilitando modelar BDTR, considerando tanto restrições lógicas como temporais, tanto para acesso aos dados como para a execução das transações.

O modelo ECPN permite incorporar controle de concorrência semântico para BDTR. O controle de concorrência é distribuído para cada objeto do sistema, ou seja, para cada objeto define-se uma função de compatibilidade que controla o acesso concorrente aos seus métodos. A função de compatibilidade especifica quando sacrificar consistência lógica por consistência temporal, ou vice-cersa, e especifica o acúmulo e limites de alguma imprecisão.

Com o objetivo de modelar aplicações complexas no contexto de BDTR introduzimos um método orientado a objetos para modelagem de BDTR. O método é baseado em OMT, no modelo RTSORAC, e em ECPN. A aplicação do método resulta na obtenção de um *modelo de objetos*, para capturar os aspectos estáticos do sistema e, um *modelo de processos*, para capturar os aspectos dinâmicos do sistema. O método suporta o tratamento de consistência lógica e temporal, e controle de concorrência semântico, com base em funções de compatibilidade definidas pelo projetista do sistema e declaradas na interface dos modelos para os objetos. Definiu-se também os passos necessários para a obtenção dos dois modelos.

O método introduzido considera também o estabelecimento de um modelo de transações com prazos estritos. Desta forma definimos claramente como o projetista deve modelar as transações considerando as restrições de tempo para cada uma delas.

Do ponto de vista de aplicação desenvolvemos um modelo para uma célula de um sistema flexível de manufatura.

Uma vez que uma das motivações centrais do método introduzido nesta tese é a utilização de métodos formais como suporte para aumentar a segurança no seu funcionamento, introduzimos um procedimento para a análise de um modelo ECPN. Para a análise de um modelo ECPN, definimos um procedimento e os algoritmos para transformar uma ECPN em uma HCPN. Este modelo HCPN é então analisado utilizando cenários e diagramas de seqüências de mensagens, e através da geração e análise do espaço de estados para os cenários.

Ainda, considerando o exemplo para a célula de manufatura, exemplificamos o procedimento de análise utilizando o pacote de ferramentas Design/CPN. Neste contexto detalhamos ainda resultados relativos ao escalonamento de transações periódicas com prazos estritos.

9.2 Pespectivas

No desenvolvimento do método introduzido nesta tese, há algumas questões que ainda não foram discutidas ou pouco elaboradas. Desta forma, a partir desta tese, abre-se um caminho promissor de desenvolvimentos a serem explorados. Tais desenvolvimentos são tanto no nível conceitual e teórico, no sentido de enriquecer o modelo, quando no nível de implementação, de modo a disponibilizar um ferramental de análise e verificação.

Portanto, alguns direcionamentos já considerados com o objetivo de enriquecer o modelo são considerados, e incluem os seguintes:

1. implementação dos algoritmos para transformar um sistema ECPN em uma HCPN.

2. considerar transações periódicas com prazos firmes e suaves para o escalonamento. Neste sentido algumas técnicas específicas têm sido discutidas na literatura, e um estudo mais aprofundado deve ser conduzido [Jon98].

Bibliografia

[AKZ96] M. Awad, J. Kuusela, and J. Ziegler, *Object-oriented technology for real-time systems : A practical approach using "omt" and "fusion"*, Kluwer Academic Publishers, 1996.

[AMBC84] M. Ajmone Marsan, D. Balbo, and G. Conte, *A class of generalised stochastic petri nets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems **2** (1984), no. 2, 93–122.

[BB94] Kenneth Baclawski and Indurkhya Bipin, *The notion of inheritance in object-oriented programming*, Communications of the ACM **37** (1994), no. 9, 118–119.

[BD91] B. Berthomieu and M. Diaz, *Modeling and verification of time dependent systems using time petri nets*, IEEE Transactions on Software Engineering **17** (1991), no. 3, 259–273.

[BLS97] A. Bestravos, K.-J. Lin, and Son. S.H., *Real-time database systems: Issues and applications*, Kluwer Academic Publishers, Boston, 1997.

[Boo94] G. Booch, *Object oriented design: With applications*, Menlo Park, CA: Benjamin/Cummings, 1994.

[BP94] Andrew Black and Jens Palsberg, *Foundations of object-oriented languages*, ACM SIGPLAN Notices **29** (1994), no. 3, 3–11, Workshop Report.

[BP98] M. Blaha and W. Premerlani, *Object-oriented modeling and design for database applications*, Prentice Hall, 1998.

[BR88] B.R. Bradinath and K. Ramamrithman, *Syncrhoning transactions on objects*, IEEE Transactions on Computers **37** (1988), no. 5, 541–547.

[BR90] I.I. Bestuzheva and V.V. Rudnev, *Timed petri nets: Classification and comparative analysis*, Automation and Remote Control **51** (1990), no. 10, 1303 – 1318.

[CdF99] S. Christensen and J.C.A de Figueiredo, *Message sequence chart lybrary*, Comunicação Pessoal, 1999.

[CGGC96] A. Choquet-Geniet, D. Geniet, and F. Cottet, *Exhaustive computation of the scheduled task execution sequences of a real-time application*, Lecture Notes in Computer Science (B. Honsson and Joachim Parrow, eds.), Springer Verlag, 1996, pp. 246–262.

[CSW97] S. Cook, B. Selic, and J. Warmer, *UML-semantics eval report*, disponível em ftp://ftp.omg.org/pub/dos/ad/97-02-16.txt, Object Management Group, 1997.

[CY90] P. Coad and E. Yourdon, *Object-oriented analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[DAJ94] Alan A. Desrochers and Robert Y. Al-Jaar, *Applications of petri nets in manufacturing systems : Modeling, control, and performance analysis*, IEEE Press, New York, 1994.

[DiP95a] L.C. DiPippo, *Real-time "databases"*, Tech. report, University of Rhode Island, TR96-249, 1995.

[DiP95b] L.C. DiPippo, *Semantic real-time object-based concurrency control*, Ph.D. thesis, Department of Computer Science and Statistics, University of Rhode Island, 1995.

[Dou98] B.P. Douglass, *Real-time "uml": Developing efficient objects for embedded systems*, Addison Wesley, Reading, Massachusets, 1998.

[DW97] L.C. DiPippo and V.F. Wolfe, *Object-based semantic real-time concurrency control with bounded imprecision*, Tech. report, University of Rhode Island, TR96-247, 1997.

[EN94] R.A. Elmasri and S.B. Navathe, *Fundamentals of database systems, 2nd edition*, Addison–Wesley Pub. Co., New York, 1994.

[Fig94] J.C.A. de Figueiredo, *Redes de petri com temporização nebulosa*, Ph.D. thesis, Curso de Pòs Graduação em Engenharia Elétrica, Universidade Federal da Paraíba, Campina Grande, PB, August 1994.

[GdFP97] D.D.S. Guerrero, J.C.A. de Figueiredo, and A. Perkusich, *Object-based high-level petri nets as a formal approach to distributed information systems*, Proc. of IEEE Int. Conf. on Systems Man and Cybernetics (Orlando, USA), October 1997, pp. 3383–3388.

[GdFP00] D.D.S. Guererro, J.C.A. de Figueiredo, and A. Perkusich, *An object-based modular "cpn" approach: its application to the specification of a cooperative editing environment*, Advances on Petri Nets: Object Orientation and Models of Concurrency, Lecture Notes in Computer Science, in press (2000).

[Gen89] H.J. Genrich, *Equivalence transformations of PrT-Nets*, Advances in Petri Nets 1989 (G. Rozenberg, ed.), Lecture Notes in Computer Science, vol. 424, Springer-Verlag, 1989, pp. 179–208.

[Gen91] H.J. Genrich, *Predicate/Transition nets*, High-Level Petri Nets: Theory and Application (K. Jensen and G. Rozenberg, eds.), Springer-Verlag, 1991.

[GFPdF97] D.D.S. Guerrero, J. P. Fernandes, A. Perkusich, and de Figueiredo, *An object based petri net model: Application to manufacturing systems*, Proc. of IEEE Int. Conf. on Systems Man and Cybernetics (Orlando, USA), October 1997, pp. 2735–2740.

[GGW99a] Holger Giese, Jörg Graf, and Guido Wirtz, *Closing the Gap Between Object-Oriented Modeling of Structure and Behavior*, Tech. report, University Münster, Computer Science Department, May 1999, 16/99-I.

[GGW99b] Holger Giese, Jörg Graf, and Guido Wirtz, *Closing the Gap Between Object-Oriented Modeling of Structure and Behavior*, UML'99 - The Second International Conference on The Unified Modeling Language, Fort Collins, Colorado, USA (Robert France and Bernhard Rumpe, eds.), Lecture Notes in Computer Science, vol. 1723, October 1999, pp. 534–549.

[GMMP89] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze, *A general way to put time in petri net*, 5th International Workshop on Software Specifications and Design, May 1989, pp. 60–66.

[GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze, *A unified high-level petri net formalism for time-critical systems*, IEEE Transactions on Software Engineering **17** (1991), no. 2, 160–171.

[Gor93] K. Gordon, *Diswg database management systems requirements*, Tech. report, Alexandria, Virginia: NGCR SPAWAR 331 2B2, 1993.

[GPdF97] D.D.S. Guerrero, A. Perkusich, and J.C.A. de Figueiredo, *Modeling a co-operative environment based on an object-based modular petri net*, Proc. of The 9th International Conference on Software Engineering and Knowledge Engineering (Madrid, Spain), June 1997, pp. 240–247.

[Gra92] M. H. Graham, *Issues in real-time data management*, The Journal of Real-Time Systems (1992), no. 4, 185–202.

[Gue97] D.D.S. Guerrero, *Sistemas de redes de petri modulares baseadas em objetos*, Dissertação de mestrado, Curso de Mestrado em Informática - Universidade Federal da Paraíba, Campina Grande, Paraíba, 1997.

[Har87] D. Harel, *Statecharts: a visual formalism for complex systems*, Science of Computer Programming **8** (1987), 231–274.

[HS86] P.J. Haas and G.S. Shedler, *Regenerative stochastic petri nets*, Performance Evaluation **6** (1986), no. 3, 189–204.

[Ja96] K. Jensen and et. al, *"design/cpn" manuals*, Meta Software Corporation and

Department of Computer Science, University of Aarhus, Denmark, 1996, On-line version:http://www.daimi.aau.dk/designCPN/.

[Ja99] K. Jensen and et. al, *"design/cpn" 4.0*, Meta Software Corporation and Department of Computer Science, University of Aarhus, Denmark, 1999, On-line version:http://www.daimi.aau.dk/designCPN/.

[Jal94] P. Jalote, *Fault tolerance in distributed systems*, Prentice Hall, New Jersey, 1994.

[Jen87] K. Jensen, *Coloured petri nets*, Lecture Notes in Computer Science, Advances in Petri Nets: Petri Nets, Central Models and Their Properties (W. Brauer, W. Reisig, and G. Rozenberg, eds.), vol. 254, Springer-Verlag, 1987, pp. 248–299.

[Jen92] K. Jensen, *Coloured petri nets - basic concepts, analysis methods and practical use - vol. 1 : basic concepts, ver. 4*, Springer-Verlag, Paris, 1992.

[Jen97] K. Jensen, *Coloured petri nets-basic concepts, analysis methods and practical use*, vol. 2, Springer-Verlag, 1997.

[Jen98a] K. Jensen, *First workshop on practical use of coloured petri nets and Design/CPN, disponivel em http://www.daimi.au.dk/cpnets/workshop98/*, DAIMI, Aarhus, Denmark, 1998.

[Jen98b] K. Jensen, *An introduction to the practical use of coloured petri nets*, Lectures on Petri Nets II: Applications, Springer, 1998.

[Jen99] K. Jensen, *Second workshop on practical use of coloured petri nets and Design/CPN, disponivel em http://www.daimi.au.dk/cpnets/workshop99/*, DAIMI, Aarhus, Denmark, 1999.

[Jon98] Greg Jones, *Scheduling with temporal and logical constraints*, Ph.D. thesis, Department of Computer Science and Statistics, University of Rhode Island, 1998.

[KS96] C. Krishna and K. G. Shin, *Real-time systems*, MacGraw Hill Series in Computer Science, 1996.

[Mar95] T. Marx, *Netcase - a petri net based method for database application design and generation*, Tech. report, University of Koblenz, Germany, Research Report 11-95, 1995.

[Mer79] P.M. Merlin, *Specification and validation of protocols*, IEEE Transactions on Communications COM-27 (1979), no. 11, 1671–1680.

[MF76] P.M. Merlin and D.J. Farber, *Recoverability of communication protocols - implications of a theoretical study*, IEEE Transactions on Communication COM-24 (1976), no. 9, 1036–1043.

[MJ92] K. Majmudar and M.A. Jafari, *Functional and performance analysis of time petri nets*, International Conference on Systems, Man, and Cybernetics, vol. 2, October 1992, pp. 980 - 985.

[Mol81] M.K. Molloy, *On the integration of dealy and throughput measures in distributed processing models*, Ph.D. thesis, UCLA, 1981.

[Mur89] T. Murata, *Petri nets: Properties, analysis and applications*, Proc. of the IEEE 77 (1989), no. 4, 541–580.

[Nat80] S. Natkin, *Les reseaux de petri stochastiques et leur application a l'evaluation des sistemes informatiques*, Ph.D. thesis, These de Docteur Ingegneur, CNAM, 1980.

[NP98] L. Nigro and F. Pupo, *Using "design/cpn" for the schedulability analysis of actor systems with timing constraints*, Proc. of First Workshop on Practical Use of Coloured Petri Nets and Design/CPN (Aarhus, Denmark, disponível em: http://www.daimi.au.dk/CPnets/workshop98/papers), 1998.

[NRA95] B. Nielsen, S. Ren, and G.A. Agha, *Rtsynchronizer: Language support for real-time specifications in distributed systems*, ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (La Jolla, USA), 1995, pp. 45–51.

[OS95] G. Ozsoyoglu and R.T. Snodgrass, *Temporal and real-time databases: A*

survey, IEEE Transactions on Data and Knowledge Engineering 7 (1995), no. 4, 47–56.

[PdFC94] A. Perkusich, J.C.A. de Figueiredo, and S.K Chang, *Embedding fault-tolerant properties in the design of complex systems*, Journal of Systems and Software 2 (1994), no. 25, 23–37.

[Pet81] J.L. Peterson, *Petri net theory and modeling of systems*, Prentice-Hall, 1981.

[PPC96] A. Perkusich, M.L.B. Perkusich, and S.K Chang, *Object oriented design, modular analysis, and fault-tolerance of real-time control software systems*, International Journal of Software Engineering and Knowledge Engineering 6 (1996), no. 3, 447–476.

[PPS95a] M.L.B Perkusich, A. Perkusich, and U. Schiel, *Integrated design of object-oriented real-time control and database systems*, Proc. of The Seventh International Conference on Software Engineering and Knowledge Engineering, SEKE'95 (Washington, USA), June 1995, pp. 150–152.

[PPS95b] M.L.B. Perkusich, A. Perkusich, and U. Schiel, *Object-oriented real-time database design and hierarchical control systems*, Proc. of International Workshop on Active and Real-Time Databases, ARTDB-95, Wokshop Series in Computing, Springer (Skovde, SE), June 1995, pp. 104–121.

[Pri95] J.J. Prichard, *"rtsql": Extending the "sql" standard to support real-time databases*, Ph.D. thesis, Department of Computer Science and Statistics, University of Rhode Island, 1995.

[PTP98] M.L.B. Perkusich, M.F.Q.V. Turnell, and A. Perkusich, *Object-oriented real-time database design based on petri nets*, Proc. of IEEE Int. Conf. on Systems Man and Cybernetics (San Diego, USA), October 1998, pp. 202–207.

[PWPD96] J. Peckham, V.F. Wolfe, J.J. Prichard, and L.C. DiPippo, *Design of a real-time object-oriented database system*, Tech. report, University of Rhode Island, TR94-231, 1996.

[RA96] S. Ren and G.A. Agha, *Specification of real-time interaction constraints*, Proceedings of the First International Symposium on Object-Oriented Real-Time Computing, 1996, pp. 123–128.

[Ram74] C. Ramchandani, *Analysis of asynchronous concurrent systems by petri nets*, Tech. Report Project MAC-TR120, M.I.T., Cambridge, MA, 1974.

[Ram93] K. Ramamrithman, *Real-time databases*, International Journal of Distributed and Parallel Databases (1993).

[RAS96] Shangping Ren, Gul A. Agha, and Masahiko Saito, *A modular approach to programming distributed real-time systems*, Journal of Parallel and Distributed Computing **36** (1996), no. 1, 4–12.

[RH80] C.V. Ramamoorthy and G.S. Ho, *Performance evaluation of asynchronous concurrent systems using petri nets*, IEEE Transactions on Software Engineering **SE-6** (1980), no. 5, 440–449.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch, *"the unified modeling language reference manual 1st edition"*, Addison Wesley, Reading, Massachusets, 1999.

[RP95] K. Ramamritham and C. Pu, *A formal characterization of epsilon serializability*, IEEE Transactions on Data and Knowledge Engineering **6** (1995), no. 7, 997–1007.

[Rum91] J. et al. Rumbaugh, *Object-oriented modeling and design*, Englewood Cliffs, N.J. : Prentice Hall, 1991.

[Sif80] J. Sifakis, *Performance evaluation of systems using nets*, Net Theory and Applications, vol. 84, Lecture Notes in Computer Science, no. Project MAC-TR120, Springer-Verlag, 1980.

[Sim96] E. Simon, *Distributed information systems*, McGraw-Hill, 1996.

[SJ91] V.S. Srinivasan and M.A. Jafari, *Monitoring and fault detection in shop floor using time petri nets*, International Conference on Systems, Man, and Cybernetics, 1991, pp. 355 –360.

[SJ92] V.S. Srinivasan and M.A. Jafari, *Fault detection/monitoring using time petri nets*, submitted to IEEE Transactions on Systems, Man and Cybernetics, 1992.

[SP] P.D. Stotts and T.W. Pratt, *Coverability graphs for a class of synchronously executed unbounded petri net*, to appear in Journal of Parallel and Distributed Computing.

[SSH99] J. Stankovic, S. Son, and J. Hansson, *Misconceptions about real-time databases*, IEEE Computer **32** (1999), no. 6, 29–36.

[Sta88] J. A. Stankovic, *Misconception about real-time computing: A serious problem for the next generation systems*, IEEE Computer Magazine **21** (1988), no. 10, 10–19.

[SWG94] Bran Selic, Paul T. Ward, and Garth Gullekson, *Real time object-oriented modeling*, John Wiley and Sons, 1994.

[Tai96] Antero Taivalsaari, *On the notion of inheritance*, ACM Computing Surveys **28** (1996), no. 3, 438–479.

[TK78] D. Tsichritzis and A. Klug, *The ANSI/X3/SPARC DBMS framework report of the study group on database management systems*, Information Systems **3** (1978), no. 3, 173–191.

[Ull93] Jeffrey Ullman, *Elements of "ml" programming*, Prentice-Hall, 1993.

[Ull98] Jeffrey Ullman, *Elements of "ml" programming, "ml97" edition*, Prentice-Hall, 1998.

[Uni96] University of Aarhus, Aarhus, Denmark, *Design cpn - overview of cpn ml syntax, version 3.0*, 1996.

[vdA92] W.M.P. van der Aalst, *Timed coloured petri nets and their application to logistic systems*, Ph.D. thesis, Eindhoven University of Technology, 1992.

[vdA93] W.M.P. van der Aalst, *Interval timed coloured petri nets and their analysis*, Application and Theory of Petri Nets 1993 (M. Ajmone Marsan, ed.),

Lecture Notes in Computer Science, vol. 691, Springer-Verlag, June 1993, pp. 453 – 472.

[WA92] M. Wong and D. Agrawal, *Tolerating bounded inconsistency for increasing concurrency in database systems*, Proc. of 11th Principles of Databases Systems (Tucson, AZ), 1992, pp. 236–245.

[Weg88] P. Wegner, *Inheritance as an incremental modification mechanism, or what like is and isn't like*, Proceedings of ECOOP'88 - European Conference on Object Oriented Programming (Oslo, Norway), Lecture Notes in Computer Science, vol. 322, Springer-Verlag, 1988, pp. 55–77.

[Weg92] Peter Wegner, *Dimensions of object-oriented modeling*, Computer **25** (1992), no. 10, 12–20.

[Weg95] Peter Wegner, *Interactive foundations of object-based programming*, IEEE Computer (1995).

[Weg96] Peter Wegner, *Interactive software technology*, Internet, May 1996.

[YWLS94] P.S. Yu, K.-L. Wu, K.-J. Lin, and S.H. Son, *On real-time databases: Concurrency control and schedulling*, Proceedings of IEEE (1994), 140–157.

[Zho97] Mengchu Zhou (ed.), *Petri nets in flexible and agile automation*, Kluwer Academic Pub., New York, 1997.

[Zub91] W.M. Zuberek, *Timed petri nets: Definitions, properties, and applications*, Microelectronics and Reliability **31** (1991), no. 4, 627 – 644.

Apêndice A

Modelo CPNH

Neste apêndice apresentamos as páginas do modelo CPN hierárquico resultante da transformação do modelo ECPN apresentado no Capítulo 7.

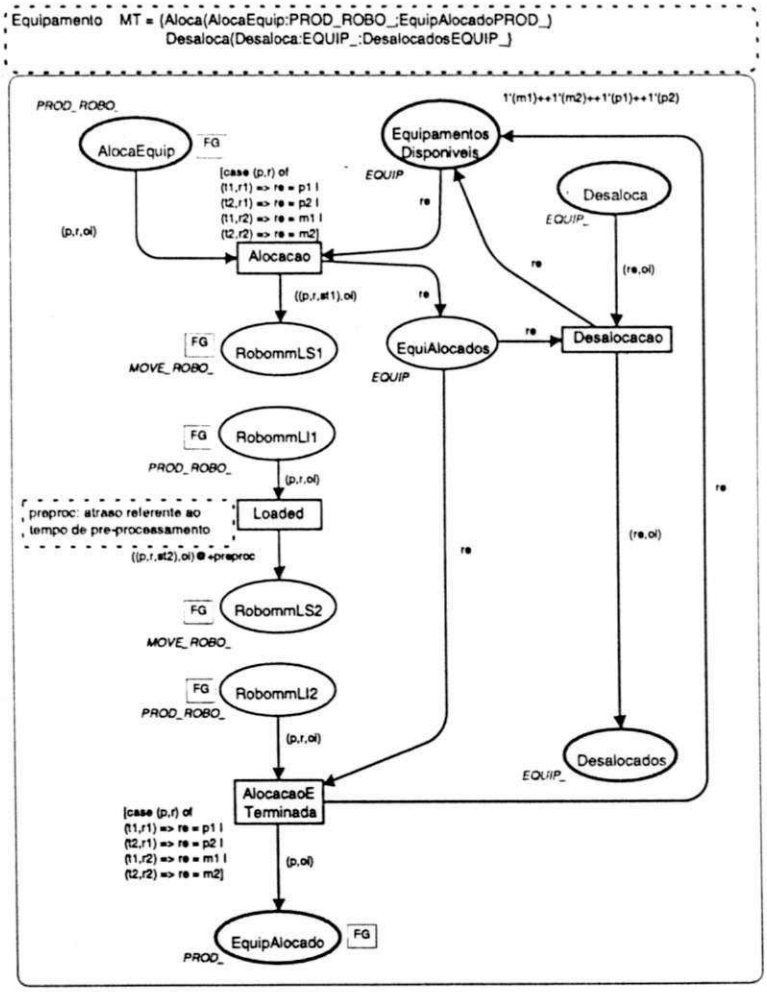

Figura A.1: Modelo CPN para a ECPN Equipamento

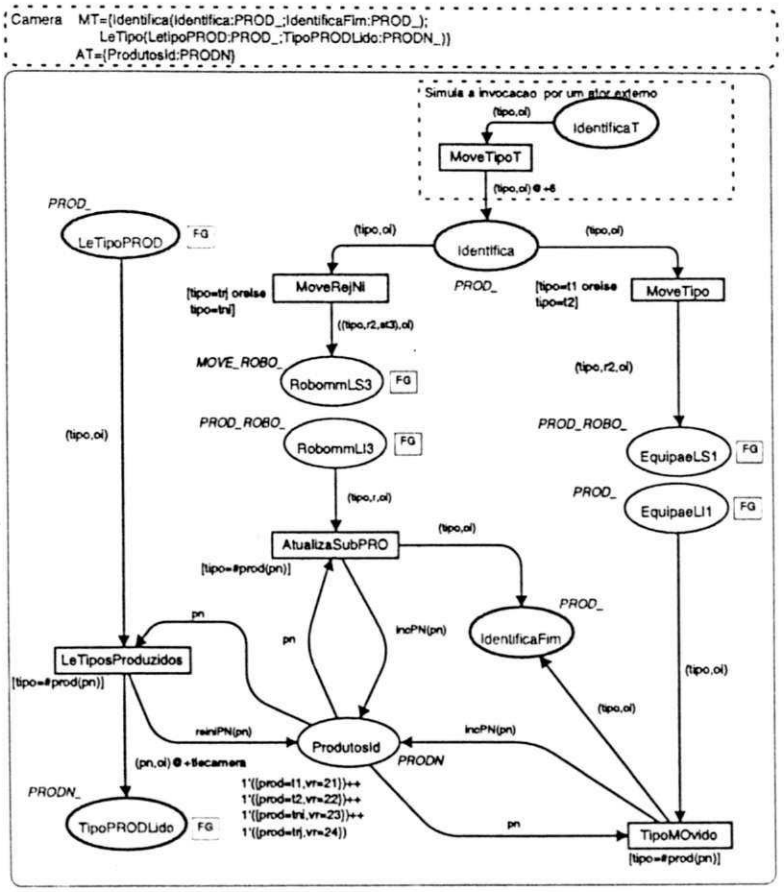

Figura A.2: Modelo CPN para a ECPN Câmera

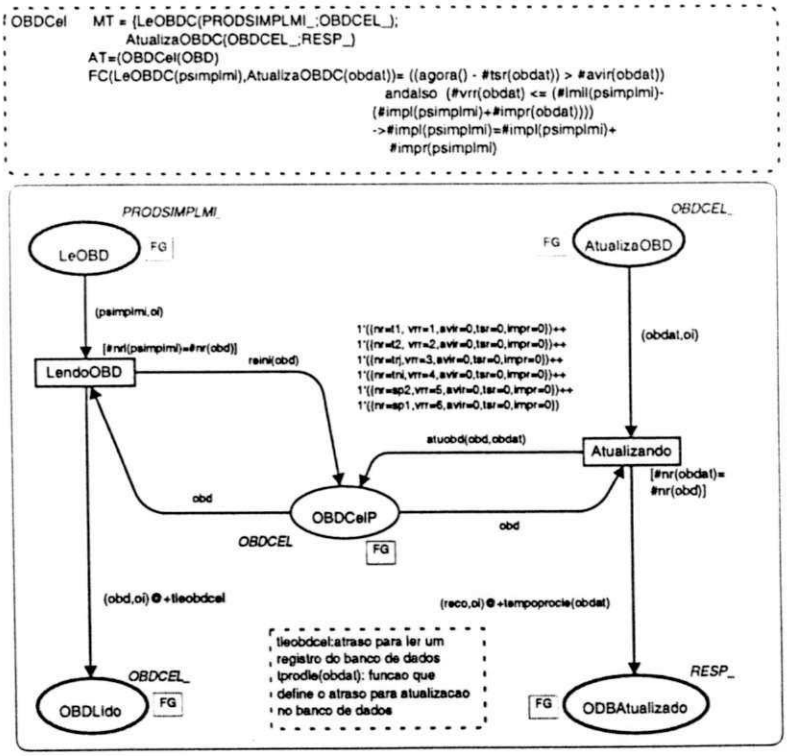

Figura A.3: Modelo CPN para a ECPN OBDCel

```
color PRODSIMPLMIXAVALFC = product PRODSIMPLMI*AVALFC;

(*
    Tipos definidos para as consultas das transacoes em tempo-real
*)

color HABILITA = with pronta | terminada timed;
color RELOGIO = with e timed;
(*
    Tipos definidos para a Rede Colorida Hierarquica Equivalente a ECPN (incluem o
    contexto, definido pelo tipo OCC
*)

color OCC = int;
color ISP = with robomm1 | robomm2 | robomm3 | equipma1 | equipma2 | bdcelle1 |
                          bdcelat1| bdcelat2 | camerle1 | prodle1;
color ISPOCC2 = product ISP*OCC*OCC;
color TPROD_ = product TPROD*OCC;
color TPRODN_ = product TPRODN*OCC;
color PROD_ = product PROD*OCC;
color PRODN_ = product PRODN*OCC;
color SPROD_ = product SPROD*OCC;
color SPRODN_ = product SPRODN*OCC;
color PRODS_ = product PRODS*OCC;
color PRODSN_ = product PRODSN*OCC;
color ROBOT_ = product ROBO*OCC;
color PROD_ROBO_ = product PROD*ROBO*OCC;
color TPROD_ROBO_ = product TPROD_*OCC;
color EQUIP_ = product EQUIP*OCC;
color MOVE_ROBO_ = product MOVE_ROBO*OCC;
color MOVENDO_ = product PROD*ROBO*TIPO_MOVE*OCC;
color RESP_ = product RESP*OCC;
color OBJ = with robomm | equipma | bdcelle | bdcelat | camerle | prodle;
color RETORNO = union pR:PROD + prR:PROD_ROBO + resR:RESP + obdR:OBDCEL +
pnR:PRODN + spnR:SPRODN;
color RETORNOS_ = product RETORNO*OCC;
color ATIVA = union  spA:SPROD + pA:PROD + obdS:OBDCEL+
psimplmiA:PRODSIMPLMI + prA:PROD_ROBO + mrA:MOVE_ROBO;
color ATIVACAOS_ = product OBJ*ATIVA*OCC;
color TP = union tp1:PROD_ROBO_ + tp2:PROD_;
color OBDCEL_ = product OBDCEL*OCC;
color PRODSIMPLMI_ = product PRODSIMPLMI*OCC;
color PRODSIMPLMI_XAVALFC = product PRODSIMPLMIXAVALFC*OCC;
(* Variaveis do modelo *)

var p,tipo : PROD; var pn : PRODN;
var ps,ps1: PRODS;
var oi,or,oc,ox : OCC;
var isp : ISP;
var tp,tpa : TPROD;
```

```
var resp : RESP;
var sp : SPROD; var spn : SPRODN;
var m,n,on,os,i : INT;
var re : EQUIP;
var r,robo1,robo2,robo: ROBO;
var mv: TIPO_MOVE;
var st: ESTAGIOS;
var nts,ts : INT;
var avi : AVI;
var obj: OBJ;
var obd,obdat,obdcel : OBDCEL;
var psimplmi : PRODSIMPLMI;
var lmi,lmile : LMI; var estado : ESTADO;
var avalfc : AVALFC; var estadoprods : ESTADOPRODS;
(* Retorna o valor do relogio de simulacao como um inteiro (INT) *)

fun agora():INT=
IntInf.toInt(time());

(*
Atualiza o rotulo de tempo (tsr) e adiciona quantidade (m) ao registro do
objeto banco de dados (obd)
*)

fun atuobd(obd:OBDCEL,obdat:OBDCEL):OBDCEL=
{nr=#nr(obdat),
vrr = #vrr(obdat) + #vrr(obd),
avir= #avir(obdat),
tsr = agora(),
impr = #impr(obdat)
};

(* Reinicializa objeto de banco de dados *)

fun reini(obd:OBDCEL) =
{nr =#nr(obd),
vrr = 0,
avir= #avir(obd),
tsr = #tsr (obd),
impr = #impr(obd)
};

fun fazatobdS(spn:SPRODN):OBDCEL=
{nr=#sprod(spn),
vrr=#vr(spn),
avir=10,
tsr=0,
impr=10
};
```

```
fun fazatobdP(pn:PRODN):OBDCEL=
{nr=#prod(pn),
vrr=#vr(pn),
avir=10,
tsr=0,
impr=10
};


fun incPN(pn:PRODN) = {prod=#prod(pn),vr=#vr(pn)+1};

fun reiniPN(pn:PRODN) = {prod=#prod(pn),vr=0};

fun incSPN(spn:SPRODN) = {sprod=#sprod(spn),vr=#vr(spn)+1};

fun reiniSPN(spn:SPRODN) = {sprod=#sprod(spn),vr=0};

fun ajustaimp(obdcel:OBDCEL,psimplmi:PRODSIMPLMI):OBDCEL=
{nr=#nr(obdcel),
vrr=#vrr(obdcel),
avir=#avir(obdcel),
tsr=#tsr(obdcel),
impr=#impr(obdcel)+#impl(psimplmi)
};

(************************************************************
Cores, Variaveis e Funcoes estatiscas
************************************************************)
color EXECUTANDO = INT;
var esatdoatsprod : EXECUTANDO;
val esatdoatsprod = 0;

fun iniciaatsprod()= esatdoatsprod = 1;
fun fimatsprod()= esatdoatsprod = 0;

(************************************************************
Variaveis de inicializacao utilizadas no modelo
************************************************************)
(* Limites maximos de imprecisao para leitura de OBDCel *)

val lmit1L = 5; (* limite de imprecisao para leitura tipo t1 *)
val lmit2L = 5; (* limite de imprecisao para leitura tipo t2 *)
val lmitrjL = 5; (* limite de imprecisao para leitura tipo trj *)
val lmitniL = 5; (* limite de imprecisao para leitura tipo tni *)
val lmisp1L = 5; (* limite de imprecisao para leitura tipo sp1 *)
val lmisp2L = 5; (* limite de imprecisao para leitura tipo sp2 *)
val impt1L = 5; (* limite de imprecisao para leitura tipo t1 *)
val impt2L = 5; (* limite de imprecisao para leitura tipo t2 *)
val imptrjL = 5; (* limite de imprecisao para leitura tipo trj *)
val imptniL = 5; (* limite de imprecisao para leitura tipo tni *)
```

```
val impsp1L = 5; (* limite de imprecisao para leitura tipo sp1 *)
val impsp2L = 5; (* limite de imprecisao para leitura tipo sp2 *)
(* Tempos de Processamentos para os Equipamentos *)

val expirou = 2;  (* time out para ciclo de producao de peca para o Produtos *)

val roboproc = 1; (* tempo de processamento para o robo *)
val preproc = 1; (* tempo para executar preprocesssamento *)
val maqproc = 1; (* tempo para executar processsamento *)
val tlecamera = 1; (* tempo para executar leitura em camera *)
val tleprod = 1; (* tempo para executar leitura em produtos *)
val tleobdcel = 6; (* tempo para executar leitura em obdcel *)
val tatuobdcelsp = 2; (* tempo para executar atualizacao de sprod em obdcel *)
val tatuobdcelp = 4; (* tempo para executar atualizacao de prodss em obdcel *)

(* Funcoes para calcular tempo de processamento *)

fun tempoprocle(obdat:OBDCEL):INT=
if (#nr(obdat)=sp1 orelse #nr(obdat)=sp2)
then tatuobdcelsp
else tatuobdcelp;

(* Caminho para o diretorio base do modelo *)

val caminho = "/home/nirvana/ligia/tese/models/";

(* Carrega funcoes para gerar "message sequence chart" *)
use (caminho^"funcoes/MSC.sml");

(* Carrega funcoes usadas nos codigos das transicoes para MSC*)
use (caminho^"funcoes/rtdbcelr.sml");
```