Caio Cesar Fattori

Implementação de uma arquitetura de controle distribuído para sistema produtivo

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências

São Paulo

2010

Caio Cesar Fattori

Implementação de uma arquitetura de controle distribuído para sistema produtivo

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências

Área de Concentração: Engenharia Mecatrônica e de Sistemas Mecânicos

Orientador: Prof. Titular Paulo Eigi Miyagi

São Paulo

2010

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de agosto de 2010

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Fattori, Caio Cesar

Implementação de uma arquitetura de controle distribuído para sistema produtivo / C. C. Fattori. – ed. rev. – São Paulo, 2010.

111 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Sistemas de manufaturas 2. Sistemas distribuídos 3. World Wide Web 4. Serviços 5. Controle de processos I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

Agradecimentos

Ao professor Paulo Eigi Miyagi, pela orientação e pelo constante estímulo transmitido durante todo o trabalho.

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

Aos professores Diolino José dos Santos Filho e Fabrício Junqueira, pelo apoio técnico e acadêmico constantes ao trabalho.

Aos amigos José Idídro Garcia Melo, Marcosiris Pessoa, Reinaldo Squillante Jr., André Cavalheiros, Osvaldo Asato e Carolina Furlanetto Mendes, que apoiaram o desenvolvimento do trabalho.

Aos meus pais Cesar André Fattori e Nielse Cristina de Melo Fattori; meus irmãos Glaucia Cristina de Melo Fattori, Diogo Cesar Fattori e Giovana Nara Barbosa Fattori; aos meus avós Celso Aparecido Fattori, Irene Caseiro Fattori e Nancy Baracat de Melo; e todos os meus amigos, pela paciência durante esse tempo em que não pude estar presente.

E a todos que colaboraram direta ou indiretamente na execução deste trabalho.

Resumo

Os mercados estão se tornando independentes de barreiras geográficas e as indústrias têm procurado novas configurações de sistemas produtivos (SPs), passando de estruturas centralizadas para estruturas distribuídas, deslocando suas plantas produtivas para países com reservas de energia e baixos custos operacionais. Para permitir a coordenação e gerenciamento deste tipo de SP disperso, aproveita-se dos avanços das tecnologias mecatrônicas e de informação, as quais permitem uma maior cooperação entre as partes do sistema e entre os atores (clientes, operadores, administradores, etc.) envolvidos. Cada parte do SP disperso que também é um SP tem seu grau de autonomia operacional. Esse tipo de sistema apresenta novos problemas de integração e coordenação de componentes, que têm que ser superados para se chegar a uma efetiva implementação. A falta de dados de testes já realizados com estruturas distribuídas dificulta o desenvolvimento prático de SPs dispersos. Este trabalho inicialmente adota uma arquitetura de controle para a negociação entre usuários de um SP disperso. Para a implementação da arquitetura foram desenvolvidos modelos computacionais explorando o potencial da rede de Petri e do PFS (production flow schema) para sistematizar a construção dos modelos. Pela análise dos modelos com base nas propriedades da rede de Petri avaliou-se a arquitetura de controle e estabeleceu-se as especificações que foram adotadas para sua implementação prática. A implementação e os testes foram realizados considerando os subsistemas autônomos de um sistema flexível de montagem que emula um SP disperso. Os estudos, análises e testes realizados foram fundamentais para adquirir maior experiência prática relacionado a concepção, projeto, implementação e operação de arquiteturas de controle distribuído aplicadas a SPs dispersos.

Palavras-chave: sistema produtivo, sistema distribuído, sistema disperso, orientação a serviço, arquitetura de controle.

Abstract

The markets are becoming independent of geographic and industry have sought new configurations of productive systems, from centralized structures to distributed structures, shifting their production plants to countries with energy reserves and low operating costs.To allow the coordination and management of this type of dispersed productive system, takes advantage of advances in mechatronics and information technologies, which allow greater cooperation between parts of the system and among stakeholders (customers, operators, administrators, etc..) involved. Each part of disperse productive system, that is also a productive system, has its own level of operational autonomy. This type of system presents new problems of integration and coordination of components that must be overcome to achieve effective implementation.The lack of data from tests already carried out with distributed structures hinders the practical development of disperse productive systems. This work initially adopts a control architecture for negotiation between users of a disperse productive system. For the implementation of the architecture were developed computational models exploring the potential of Petri networks (PN) and the production flow schema (PFS) to systematize the construction of models. For the analysis of models based on the PN properties we evaluated the control architecture and established the specifications that were used for its practical implementation. The implementation and the tests were performed considering the autonomous subsystems of a flexible mounting system that emulates a disperse productive system. The studies, analysis and tests were essential to acquire more practical experience related to conception, design, implementation and operation of distributed control architectures applied to disperse productive systems.

Keywords: productive systems, distributed systems, disperse system, service-oriented, control architecture

Lista de Figuras

Lista de Tabelas

Lista de listagens

Lista de abreviaturas e siglas

BPEL4WS	*Business Process Execution Language for Web Services*
CAD	*Computer-aided design*
CAM	*Computer-aided manufacturing*
CIM	*computer integrated manufacturing*
CNC	*computadorized numerical control*
FMS	*Flexible Manufacturing System*
JDBC	*Java Database Conectivity*
JIT	*Just-In-Time*
MES	*Manufacturing Execution System*
PFS	*Production Flow Schema*
RdP	rede de Petri
SED	Sistema a Eventos Discretos
SFMA	Sistema Flexível de Montagem Automatizado
SOA	*Service-oriented Architecture*
SoaML	*Service oriented architecture Markup Language*
SOAP	*Simple Object Access Protocol*
SODA	*Service-Oriented Database Architecture*
SP	Sistema Produtivo
TI	Tecnologia de Informação
UDDI	*Universal Description, Discovery and Integration*
WS	*Web Services*
WSDL	*Web Service Definition Language*

Lista de símbolos

PN	rede de Petri
P	conjunto de lugares da rede de Petri
T	conjunto de transições da rede de Petri
F	conjunto de arcos orientados da rede de Petri
W	conjunto de pesos dos arcos orientados da rede de Petri
M	conjunto das marcas nos lugares da rede de Petri
N	estrutura da rede de Petri
R	conjunto das marcas alcançáveis de rede de Petri
S	*Web Service*
$NameS$	nome do serviço
$Desc$	descrição do serviço
Loc	localização do serviço
URL	endereço de chamada do serviço
CS	componentes do serviço
SN	modelo em rede de Petri do serviço

Conteúdo

1 Introdução

De acordo com Villani, Miyagi e Valette (2007), os sistemas produtivos (SPs) são definidos como sendo aqueles que realizam processos utilizando recursos materiais, equipamentos, recursos humanos, informações e outras entidades físicas, para a produção de bens ou serviços. Neste sentido, os sistemas de manufatura integram um grupo especial de SPs que envolvem serviços e processos de transformação física de matéria-prima para agregar valor e gerar um produto.

Localmente, os SPs são compostos por um sistema físico (que é chamado de planta) e um sistema de controle. Os sistemas físico e de controle comunicam-se pela troca de informações em que a planta transmite seu estado ao sistema de controle e o mesmo transmite comandos aos atuadores da planta. A alteração dos dados nos sistemas de controle, em geral, gera modificações no estado de uma entidade física e, diferentemente do que ocorre em sistemas de informação, muitas das modificações na planta podem ser irreversíveis.

Segundo Shimizu, Zhang e Batres (2007), uma empresa que implementa um SP usa recursos com funcionalidades específicas como energia, materiais, recursos monetários, trabalho, máquinas e conhecimento para produzir um produto com valor agregado. A natureza dos SPs pode ser estudada a partir do processo de produção como uma composição de sub-processos, cada um dedicado a uma função específica. Na década de 1940 foi introduzida a noção de pensamento sistêmico, que dizia que o desempenho do todo é afetado pela interação sinérgica de suas partes. Posteriormente, o desenvolvimento da ciência da computação possibilitou a análise de modelos, a otimização e a resolução de problemas matemáticos complexos relacionados a SPs. No início da década de 1990, muitos SPs adotaram a estratégia de produção enxuta para aumentar a sua competitividade global. Essa estratégia é derivada do conceito de *Just-In-Time* (JIT), desenvolvido pela empresa Toyota na década de 1960, e propõe que os SPs minimizem o consumo de recursos em todas as etapas de produção. Essa estratégia induz a redução da flexibilidade do SP, pois quanto mais rígido o processo produtivo maior a eficiência atingida (LASA; CASTRO; LABURU, 2009). Por outro lado, muitos trabalhos afirmam que a flexibilidade do SP é fundamental para a sobrevivência das empresas no atual

contexto de economia globalizada. Assim, com o crescimento da tecnologia da informação (TI), existem muitos trabalhos buscando maior eficiência na produção sem aumentar a rigidez dos SPs (BRUUN; MEFFORD, 2004), isto é, procura-se desenvolver SPs capazes de produzir com maior diversidade e eficiência. Máquinas de comando computadorizado como *computadorized numerical control* (CNC) e robôs industriais permitem uma produção mais flexível e mais eficiente. As tecnologias auxiliadas por computador, como o CAD/CAM (*Computer-aided design*/*Computer-aided manufacturing*), foram incorporadas aos SPs contribuindo para o desenvolvimento dos sistemas flexíveis (FMS - *Flexible Manufacturing System*) e os sistemas integrados por computador (CIM - *computer integrated manufacturing*). Além da parte mecânica e eletro-eletrônica dos FMS, o projeto de seu sistema de controle é ainda um campo amplamente discutido na comunidade científica. Os múltiplos e, por vezes, contraditórios requisitos de produtividade, qualidade e diversificação de produtos influenciam o projeto do FMS e seu controle. De qualquer modo, os trabalhos tem convergido para a idéia que um SP deve ser reprogramável, operando com processamento de dados distribuídos e fluxo de materiais automatizado usando uma arquitetura de controle retro-alimentada altamente flexível, controlada por computador (AUSFELDER; CASTELAIN; GENTINA, 1994).

Além disso, desde a década de 1990, os SPs e respectivas cadeias de suprimentos têm alterado a forma de produção tradicional em massa para a customização em massa, causada pelo aumento da competição do mercado global (THOMAS; ARTIBA, 2009). Com mercados mais exigentes para os produtos de alta qualidade a custos mais baixos, altamente customizados e com ciclos de vida relativamente curtos, foram impostos novos requisitos para as empresas, como qualidade, tempo de resposta, agilidade e flexibilidade, que são cruciais para uma empresa permanecer no negócio. Neste tipo de mercado, as empresas não devem atuar independente, sendo forçadas a reconsiderar a maneira como elas se organizam para aumentar sua competitividade (NASSAR et al., 2008) (LEITAO, 2009). Ao mesmo tempo, houve uma descentralização da produção causada pelo deslocamento físico das plantas, isto é, SPs para locais em que os custos de energia e operação são menores (SHAH, 2005). O crescimento da complexidade dos novos produtos também contribui com a descentralização da produção por fazer com que as empresas precisem colaborar, umas com as outras, na produção, na cadeia de suprimentos ou em redes de serviços, em que cada parceiro na cadeia ou na rede contribui para a produção do produto final (GREFEN et al., 2009). Consequentemente, a pesquisa em controle de SPs mudou da abordagem tradicional centralizada para uma estrutura em que os SPs ou parte desses SPs ficam espalhados por todo o mundo (JUNQUEIRA; VILLANI; MIYAGI, 2005), tornado-se um sistema disperso. Neste contexto, é fundamental que os SPs sejam concebidos visando diferentes formas de flexibilidade para seus produtos e processos produtivos,

explorando devidamente as novas tecnologias, tais como: a infraestrutura disponibilizada pelo programa Tidia-KyaTera da FAPESP, que visa a criação de um ambiente colaborativo, baseado em uma rede de comunicação (de fibra óptica) com uma largura de banda prevista para 1 Terabit/s. Um SP disperso é, em princípio, uma coleção de máquinas autônomas, chamadas de nós do sistema. Os nós são conectados via uma rede de comunicação. Nesta estrutura tem-se uma entidade chamada de *middleware*, que permite aos nós coordenar suas atividades e compartilhar os recursos do sistema, como em uma cadeia de suprimentos (BONDAVALLI et al., 2010). O rápido avanço em redes de comunicação e em tecnologias computação embarcadas tem feito os SPs dispersos (geograficamente distribuídos) se difundirem na prática de engenharia: na pesquisa de multi-robôs, nas redes de sensores sem fio e nos veículos guiados automaticamente. Esses sistemas consistem de múltiplos agentes autônomos interagindo entre si por um objetivo global. Assim, a estratégia de controle distribuído faz com que cada agente tenha sua própria observação local e estratégias de controle (CAI; WONHAM, 2010) (BASIRI; BISHOP; JENSFELT, 2010).

Os SPs estão assim evoluindo para ambientes mais ágeis em que as respostas rápidas podem gerar novas mudanças de requisitos dos clientes e do ambiente de mercado. Com isso, tem crescido o número de empresas com foco apenas em suas principais competências e assim dependem de outras para prover especialidades complementares e recursos. Assim, a colaboração em SPs faz com que empresas, clientes e parceiros trabalhem juntos para obter maior ganho. A colaboração em SPs com orientação a serviços tem atraído a atenção de pesquisadores (YAN et al., 2010). Desta forma, se tipicamente, os SPs são organizados em estruturas hierárquicas, Figura 1.1, alguns trabalhos mais recentes (GROBA et al., 2008) sugerem que entre as camadas 2 e 3, isto é, entre o sistema de execução de manufatura (MES - *Manufacturing Execution System*) e o controle, deve-se considerar uma camada de integração, como na Figura 1.2, sendo essa um *middleware* (uma aplicação com ferramentas e interfaces disponíveis para uma aplicação comunicar-se com outras aplicações em outras camadas) que permite a integração de múltiplos serviços em camadas diferentes.

Luo, Tzou e Lan (2003) dizem que os SPs estão apoiadas em 4 (quatro) processos de negócios centrais: (a) processo de desenvolvimento de produtos; (b) processo de atendimento de pedidos; (c) processo de atendimento ao cliente; e (d) processo de aquisição de clientes; e o aumento da velocidade de transmissão da *internet* permite o aprimoramento desses processos pelas empresas, especialmente nos que se referem aos clientes. Entretanto, essa visão de processos de negócios não contempla completamente os processos em SPs dispersos, pois surgem novos processos do ponto de vista gerencial, como: processo de gerenciamento de fornecedores e processo de gerenciamento de transporte de materiais; pois neste caso as etapas do processo

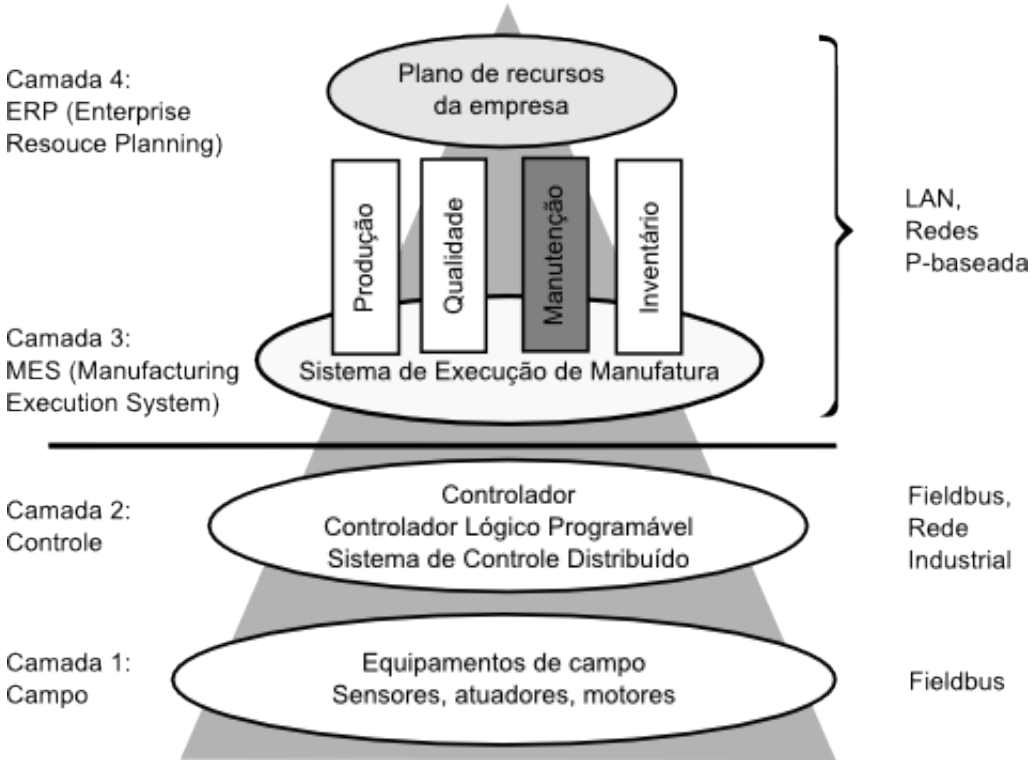


Figura 1.1: Estrutura típica em camadas de SPs (GROBA et al., 2008).

produtivo não ocorrem fisicamente no mesmo local. Nos SPs dispersos os processos produtivos devem ser integrados. Para isso, os SPs devem coordenar suas operações de modo que haja sincronização das etapas dos processos produtivos. Em Palomera-Perez e Benitez-Perez (2009), a coordenação de tarefas é aplicada a sistemas distribuídos e a composição de tarefas coordenadas gera um fluxo de dados entre os nós (ou SPs) envolvidos, modelados como uma rede de processos.

Em Cheng et al. (2010), os requisitos para a integração de sistemas (no trabalho, os sistemas usados foram de cadeia de suprimentos) utilizando a TI são:

- **Facilidade de instalação e configuração.** Inclui 3 (requisitos) que são: acomodar os membros com uma variedade de graus de sofisticação, oferecer uma ampla faixa de funcionalidades e permitir uma constante mudança da lista de empresas e clientes. Em especial, o terceiro requisito faz com que a integração de sistemas seja flexível, permitindo uma instalação rápida de novos elementos;

- **Baixo custo.** A maior parte das empresas utilizam poucos funcionários e são pequenas ou médias. Para viabilizar a participação dessas, o custo das aplicações de software para a integração dos sistemas não pode ser elevado;

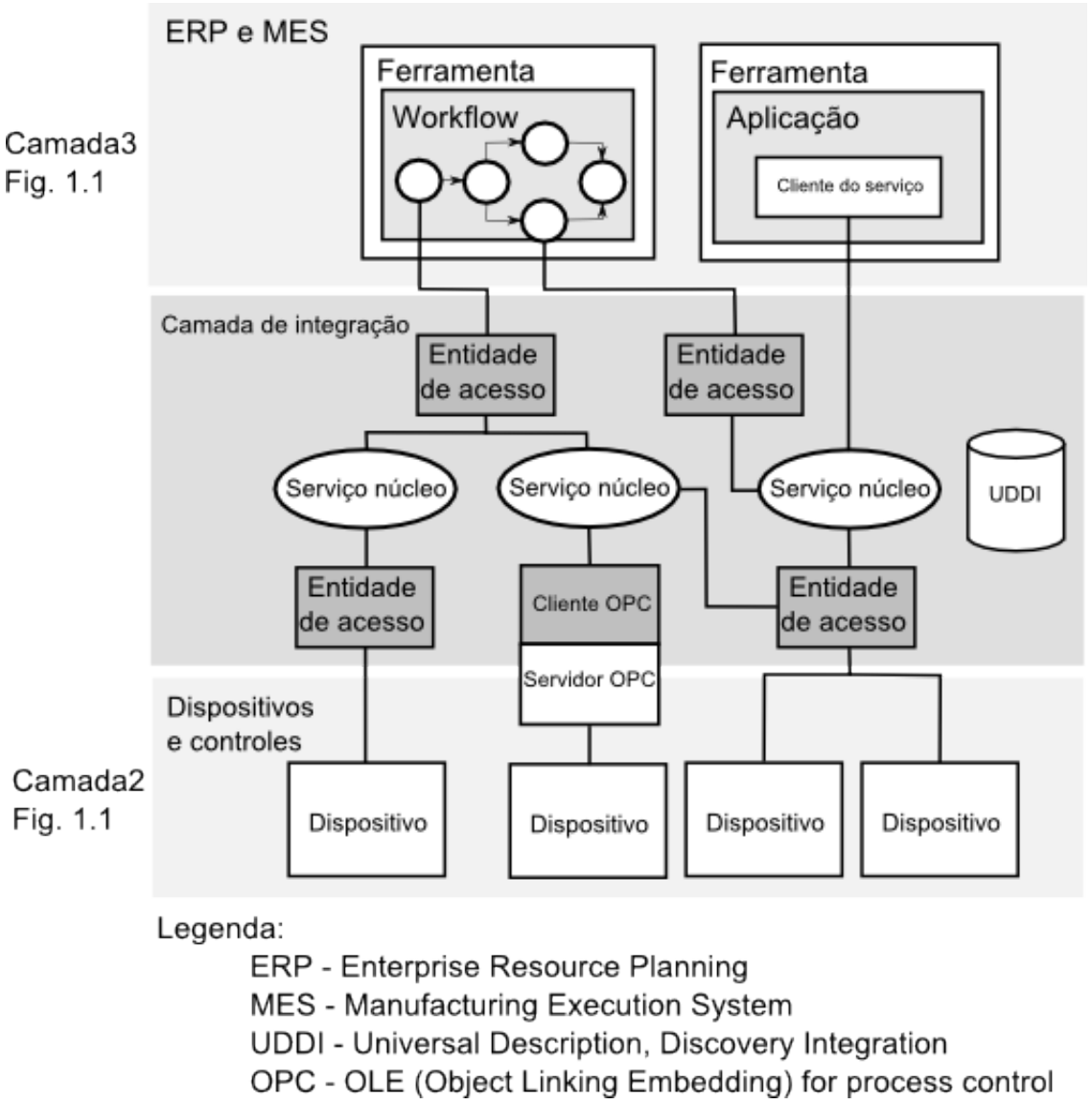


Figura 1.2: Estrutura de uma camada de integração como *middleware* (GROBA et al., 2008).

- **Facilidade de conexão e integração.** Esse requisito interfere diretamente na variedade de graus de sofisticação. Ele se deve ao fato de que em alguns ramos de atuação, os usuários se diferenciam em experiência e em níveis educacionais prévios. Isso significa que o sistema de integração deve oferecer um ambiente com diferentes níveis de complexidade para atender usuários.

- **Capacidade de integrar sistemas diferentes tipos de externos e de informação.** Com sistemas distribuídos geograficamente, cada um pode usar e manter suas informações separadamente. Para esses, não só é desejável expor suas aplicações internas para sistemas externos como é benéfico permitir a integração e a conexão com esses, por troca de funções e informações, em tarefas colaborativas.

- **Acesso personalizável a informações e aplicações.** Alguns usuários podem ser relutantes ao compartilhamento de informações com outros que não possuem relações comerciais com estes. A integração deve permitir ao usuário personalizar o acesso a informação e aplicações que este oferece.

Alguns trabalhos de sistemas distribuídos têm se concentrado no uso de uma estrutura orientada a serviços, em que as funções e as informações que as partes (que compõem o sistema) oferecem são vistas como serviços prestados. Uma das motivações disso é o aumento na quantidade de serviços baseados em internet e de usuários de internet que oferecem estes serviços na internet. Com isso, têm surgido comunidades em volta de interesses comuns. O aumento de serviços disponíveis representa um aumento da escala da internet e ao mesmo tempo gera uma maior expectativa da recuperação da informação da mesma (WANG; TSE; ZHOU, 2009). Esses serviços, por sua vez, podem ser entendidos como recursos computacionais da internet e podem ser compartilhados entre os usuários da internet para atender os interesses comuns. Entretanto, segundo Han et al. (2008), a computação convencional não era capaz de atender o compartilhamento de recursos computacionais, o que motivou o desenvolvimento de aplicações baseadas em *Web Services* (WS) que tem vantagens no compartilhamento de recursos e facilita o desenvolvimento de aplicações de maior complexidade. Os métodos relevantes de WS têm sido estudados e muitos resultados de pesquisas têm sido usados na indústria.

Em sistemas distribuídos, os elementos de um subsistema não têm contato direto com os elementos de outros subsistemas. Entretanto, no caso das máquinas de um planta de SPs é necessário o envio de comandos por operadores. Bailey (1995) mostra a aplicação de teleoperação de máquinas de prototipagem rápida. O uso de operadores humanos se deve ao fato de que esses possuem habilidades e inteligência para tomar decisões que em muitas situações práticas são efetivas e confiáveis (CHO et al., 2010). A demora no tempo de resposta e a perda de dados transmitidos na rede de comunicação são fatores importantes e que influenciam na qualidade do serviço, especialmente nos parâmetros de estabilidade e eficiência do sistema (WANG; TAN, 2009). Segundo Goradia, Xi e Elhajj (2005), teleoperação implica no controle do sistema de um local distinto das instalações físicas dos equipamentos envolvidos no processo produtivo. Um sistema de teleoperação tem os seguintes componentes básicos: sistemas remotos com sensores, atuadores, controladores e sistemas de potência; uma interface que permite ao teleoperador gerar comandos e receber os sinais dos sensores da planta; e uma comunicação entre teleoperador e planta em que a informação possa trafegar entre os ambientes: local (planta) e remoto.

A TI oferece um poderoso *framework* para o desenvolvimento da teleoperação, utilizando

as tecnologias disponíveis na *internet*, indicando soluções para implementar as funções da camada de integração (Fig. 1.2) diretamente relacionadas com a camada de controle (camada 2 da Fig. 1.1). Por outro lado, as funções da camada de integração relacionadas com a camada MES (camada 3 da Fig. 1.1) envolve novas soluções para a negociação de pedidos entre clientes, operadores e SPs dentro desse ambiente de SP disperso.

Do exposto, observa-se que existem várias propostas e soluções, principalmente na área de TI, que podem e devem ser consideradas para a implementação de SPs dispersos. Entretanto, conforme levantamento bibliográfico até aqui realizado não foram identificadas informações específicas sobre implementações reais de SPs dispersos que permitam o estabelecimento de um procedimento de projeto eficiente para esse tipo de sistema, em especial para as funções de negociação.

Objetivo geral

Considerando que a aquisição de experiência prática de concepção, instalação e operação de SPs é fundamental para o estudo e estabelecimento de um procedimento racional de projeto desses sistemas, o objetivo desse trabalho é desenvolver e implementar uma arquitetura de controle dos processos de um SP disperso na negociação de pedidos explorando os conceitos e soluções já desenvolvidas para sistemas computacionais distribuídos.

Essa arquitetura de controle deve considerar a rigidez intrínseca de processos de manufatura que, em geral, não são reinicializáveis, como no caso de sistemas de informação. Nesses processos produtivos pode também haver interferência de um teleoperador e a arquitetura de controle deve fornecer as ferramentas necessárias para processar essa interferência.

Metodologia de pesquisa

A Figura 1.3 sintetiza o método de desenvolvimento de pesquisas, citada em Jensen (1992) e que também foi adotada no presente trabalho.

Esta abordagem evidencia a forte relação entre teoria, ferramentas e aplicações, estando os três aspectos integrados no processo de desenvolvimento do presente trabalho e contribuindo sinergeticamente nos processos associados a cada um deles.

É uma abordagem de engenharia que, considerando os aspectos formais associados à definição do modelo e dos métodos de análise aplicáveis, utiliza as aplicações como motor de desenvolvimento para ferramentas de suporte à aplicação prática considerada.

Uma caracterização tripartida, como a proposta em (JENSEN, 1992), é também seguida nas metodologias de projeto de sistemas. Citando outra abordagem, tem-se, por exemplo, o

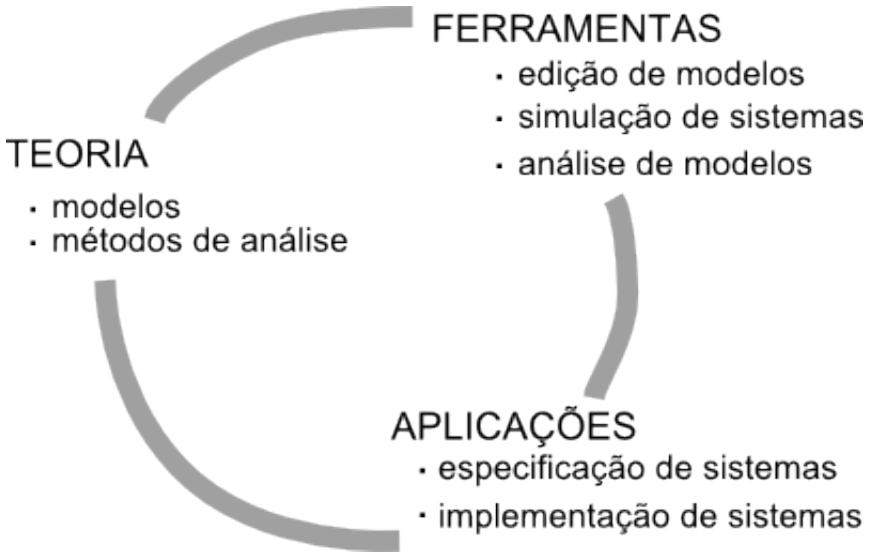


Figura 1.3: Ciclo de vida para desenvolvimento da pesquisa (JENSEN, 1992)

trabalho de Calvez (1993) que identifica três aspectos básicos para a resolução de um problema: os métodos, as ferramentas e as técnicas (Figura 1.4). As técnicas fornecem o suporte direto à implementação, enquanto as ferramentas são utilizadas para implementá-las e os métodos permitem evoluir da definição do problema até a sua implementação.

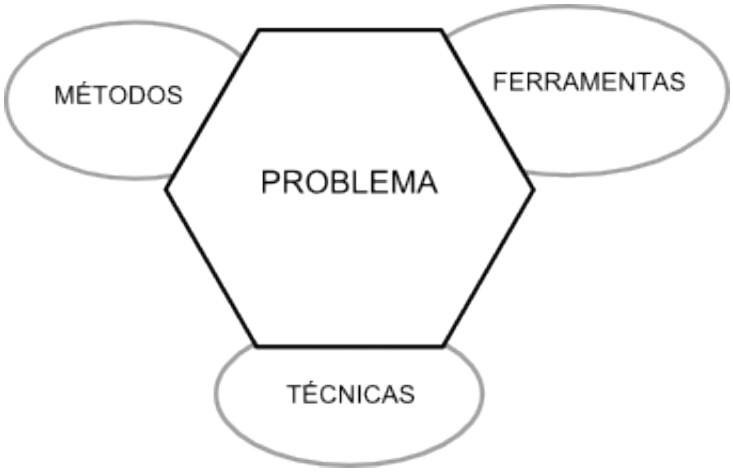


Figura 1.4: Aspectos principais na resolução de um problema (CALVEZ, 1993)

No trabalho de Boehm (1988), o autor propõe o modelo de ciclo de vida de projeto em espiral. Neste modelo, as fases de projeto são 4 (quatro) e iterativas (Figura 1.5) sendo:

- planejamento da fase seguinte;
- definição dos objetivos, alternativas e restrições;

- avaliação da solução e análise de riscos; e
- desenvolvimento do produto e verificação.

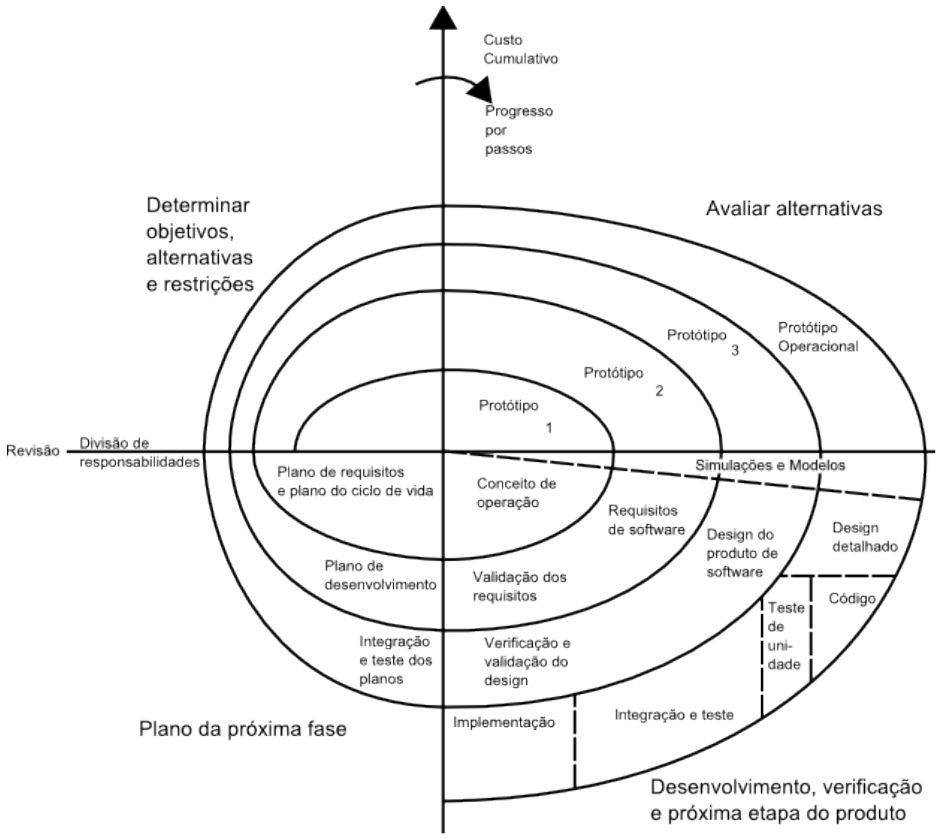


Figura 1.5: Modelo espiral do processo de software (BOEHM, 1988)

Essas fases podem ser combinações de algumas abordagens como a expressão das necessidades, a viabilidade, a prototipagem e o desenvolvimento final.

Também neste trabalho, os três aspectos identificados por Jensen (1992) foram conduzidos simultaneamente, condicionando-se mutuamente. Os desenvolvimentos nas três áreas identificadas beneficiaram-se das sinergias resultantes das atividades em cada uma das outras duas áreas.

A Figura 1.6 sintetiza as principais referências do presente trabalho em termos da trilogia apresentada.

Organização do texto

No presente capítulo, foram introduzidos alguns conceitos como de sistema produtivo (SP) e sistema distribuído e no Capítulo 2, apresenta-se os conceitos adicionais considerados para o desenvolvimento do trabalho: arquitetura orientada à serviços (SOA); *Web Services* (WS); a

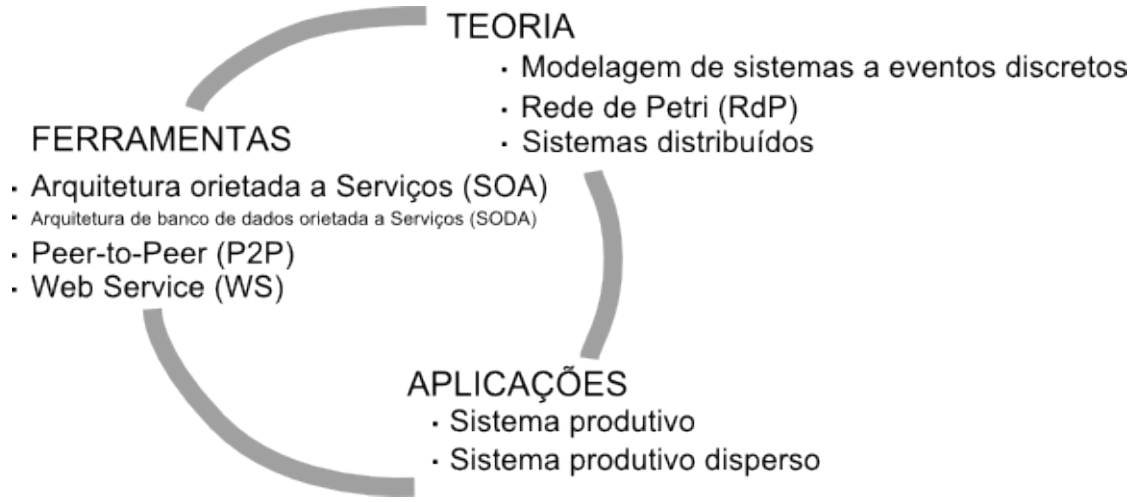


Figura 1.6: Ciclo de desenvolvimento considerado nesta pesquisa

arquitetura de banco de dados orientada à serviços (SODA); rede de Petri (RdP); modelagem de SPs baseado em WSs; comunicação *Peer-to-Peer* (P2P).

No Capítulo 3 é descrito o sistema que foi considerado para os estudos de casos desta pesquisa e que emula um sistema produtivo disperso.

No Capítulo 4 os usuários da arquitetura de controle são definidos e também são estabelecidos os direitos e as responsabilidades de cada um deles para que os pedidos possam ser atendidos.

No Capítulo 5, é descrita a modelagem da arquitetura de controle do sistema que emula um sistema produtivo disperso e são apresentados os resultados da análise dos modelos.

No Capítulo 6, o modelo desenvolvido no capítulo anterior é utilizado para implementar os WSs e a estrutura das funcionalidades é detalhada, São apresentados os resultados dos testes feitos no sistema que emula um SP disperso.

2 Conceitos Básicos

Este capítulo inicialmente apresenta os conceitos de arquitetura orientada à serviços (SOA) e seus derivados, *Web Service* (WS) e arquitetura de banco de dados orientada à serviços (SODA). Os WSs são uma solução computacional para a implementação de uma arquitetura de controle baseada em serviços. A SODA é uma solução que permite o uso de bancos de dados distribuídos utilizando conceitos de serviços.

A seguir, apresenta-se a rede de Petri (RdP), isto é, sua formalização, representação gráfica e propriedades. A RdP é a técnica adotada para modelagem da arquitetura de controle de um sistema baseada em WSs. Assim, além da modelagem de WSs apresenta-se a metodologia *Production Flow Schema* (PFS) para o desenvolvimento dos modelos dos processos envolvidos.

2.1 Service-Oriented Architecture (SOA)

A *Service-oriented Architecture* (SOA) é um paradigma na área de TI para organizar e utilizar funcionalidades computacionais (como processamento, armazenamento, filtragem e outras) distribuídas que podem estar em diferentes domínios físicos (plataformas de *hardware*). Em geral, entidades possuem funcionalidades para resolver ou suportar uma solução para os problemas que eles encontram no curso de seus negócios. É natural imaginar as necessidades de uma pessoa sendo atendidas pelas funcionalidades oferecidas por outra pessoa, ou em computação, os requisitos de um agente computacional de um certo proprietário sendo atendidos por outro agente computacional pertencente a outro proprietário. Não há necessariamente uma correlação de um para um entre necessidades e funcionalidades, ou seja, uma funcionalidade não foi feita necessariamente para atender uma necessidade, mesmo que seja usada para atender essa. A granularidade das necessidades e funcionalidades varia da fundamental à complexa, já que dependem da quantidade de operações computacionais feitas dentro de uma funcionalidade ou da quantidade de operações computacionais requeridas dentro de uma necessidade, e uma necessidade pode requerer a combinação de várias funcionalidades enquanto uma simples funcionalidade pode atender mais do que uma necessidade (OASIS,

2006) (WANG; SONG; XIONG, 2009). Com o conceito de serviço, que é a implementação de uma funcionalidade computacional, um sistema de informação pode ser melhorado para um novo nível e ser mais adequado a ambientes interoperáveis e heterogêneos (WANG; HU, 2009).

A (arquitetura) SOA provê um poderoso *framework* [1] para encontrar necessidades e funcionalidades e para combinar funcionalidades para atender uma necessidade (OASIS, 2006) (ZHU; MELO, 2009). SOA pode ser considerada uma plataforma de integração universal alavancando funcionalidades de *software* existentes. A lógica de integração de orientação a processo é separada da interação do usuário, ou seja, a interface com o usuário é feita de uma forma independente da forma como é feito o controle de atividades. A conectividade é acoplada pelos adaptadores, ou seja, um serviço comunica-se com outro por um adaptador. Um *message broker* (corretor de mensagens) fornece um mecanismo para validação, transformação e filtro de dados para a SOA (MAYERL; VOGEL; ABECK, 2008).

Com as características acima, a SOA adota um tipo de acoplamento considerado "fraco", ou seja, a comunicação entre serviços independe da forma como esses foram criados, mas de adaptadores (SHAN et al., 2009) (MAYERL; VOGEL; ABECK, 2008) (ZHU; MELO, 2009) (WANG; HU, 2009) e usa coleções de serviços reusáveis que podem ser recombinados para definir novos processos ou refinar um já existente. A idéia fundamental é similar a de modelos baseados em agentes: um serviço de baixo nível pode ser parte de alguns serviços de alto nível ou de processos de negócios colaborando com outros serviços enquanto tem um certo grau de autonomia (MEDINA et al., 2009) (WANG; HU, 2009) (ZHU; MELO, 2009). Ambientes fracamente acoplados apresentam uma redução de dependências entre aplicações com interação de estilo de mensagem, semântica e sintaxe. Como a SOA é, em geral, concebida sobre um *middleware* existente, a implementação e as tecnologias específicas do detalhamento do serviço são ocultados atrás da definição de interfaces abstratas cedendo à forte autonomia, ou o fraco acoplamento, dos serviços a serem integrados, (MAYERL; VOGEL; ABECK, 2008). Os serviços na SOA devem ser auto-contidos, ou seja, devem executar tarefas pré-determinadas (ZHU; MELO, 2009) (WANG; HU, 2009). A SOA provê um padrão de desenvolvimento flexível independente da complexidade da granularidade do serviço (SHAN et al., 2009).

Alguns conceitos descritos em OASIS (2006) relacionados com o paradigma SOA são:

- **Visibilidade:** refere-se ao poder das necessidades e das funcionalidades de uma ver as outras. É feita pela descrição de aspectos, como funcionais e requisitos técnicos; restrições e políticas relacionadas; e mecanismos de acesso e resposta.

[1] uma estrutura que pode ser usada como base de sustentação de um certo trabalho

- **Semântica:** refere-se a forma de como uma necessidade é descrita. A semântica e a sintaxe de uma necessidade devem ser amplamente acessíveis e entendíveis.

- **Interação:** é a atividade de usar uma funcionalidade. Tipicamente mediada pela troca da mensagens, uma interação prossegue por uma série de trocas de informações e chamadas de ação.

- **Contexto de execução:** é um conjunto de elementos técnicos e de negócios que fica entre as necessidades e funcionalidades. Este permite ao provedor (que fornece uma funcionalidade) e usuário (que tem uma necessidade de serviço) interagirem e estabelecerem um ponto de decisão para qualquer política e contrato que podem estar em vigor.

A SOA é usada em trabalhos da área de TI e processos de negócios (MAYERL; VOGEL; ABECK, 2008) como alocação adaptativa de recursos computacionais (YAU; AN, 2009) e integração de aplicações corporativas (KELLER; DIETRICH, 2009) ou em áreas de gerenciamento de produção agrícola (WOLFERT et al., 2010) e, mais recentemente, também existem estudos da aplicação de SOA para SPs (MELO et al., 2009).

2.1.1 Web Services (WSs)

Os WSs foram concebidos para permitir o acesso consistente a recursos computacionais em múltiplas plataformas heterogêneas, suportando certo grau de colaboração inter-organizacional no gerenciamento de processos. WS estabelece meios padrões de inter-operação entre diferentes aplicações de *software* que podem ser executados em uma variedade de plataformas ou *frameworks*. Sistemas heterogêneos podem ser facilmente integrados por WSs desde que os serviços que os integram possuam mecanismos padrões para separar as definições da parte das interfaces da parte com protocolos de transferência particulares e formatos de codificação de dados (KIM et al., 2006). Os WSs possuem um conjunto de padrões que são utilizados para sua especificação. Na Figura 2.1 são mostrados alguns exemplos de padrões de especificação para composição de serviços publicados por um grupo de empresas que inclui MICROSOFT, IBM e outras (FARAHBOD, 2001), em que cada quadro mostra um conjunto de padrões para cada função que se deseja implementar no serviço, como protocolos de transferência e formato de mensagem.

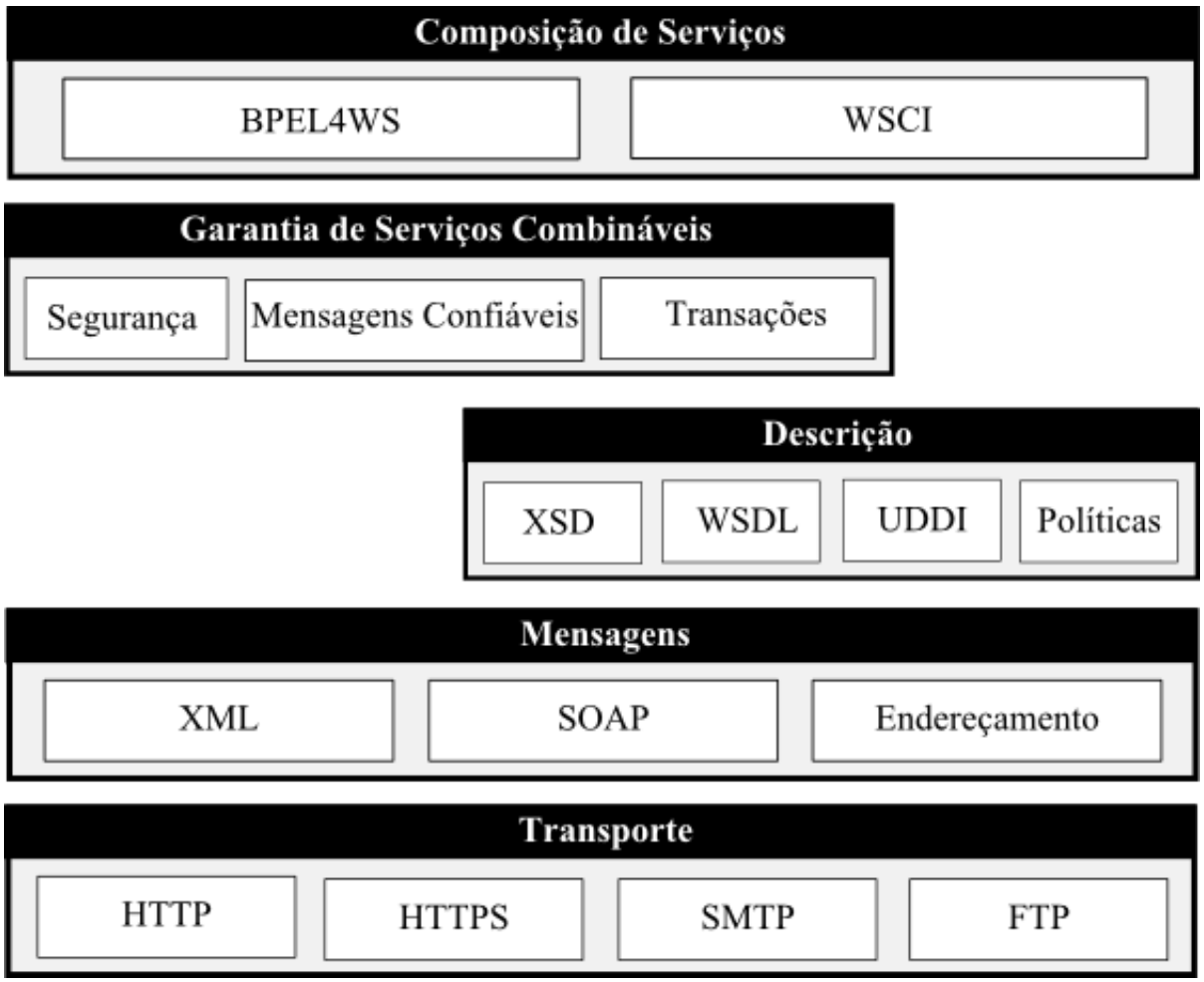


Figura 2.1: Especificações de WSs (FARAHBOD, 2001)

2.1.2 Service-Oriented Database Architecture (SODA)

A *Service-Oriented Database Architecture* (SODA) é implementada a partir do conceito de *middleware* para ambientes de banco de dados distribuídos. A idéia é distribuir o processamento da consulta dentro de um sistema aberto baseado em normas comuns, permitindo a todos juntar e prover suas próprias implementações de algoritmos em uma base WS para estender a capacidade de processamento de consultas, como na Figura 2.2 (BERAN; HABEL; SCHIKUTA, 2008).

A integração de dados pode ser ágil e configurável, assim como novos serviços podem ser adicionados ao sistema sem necessidade de interromper a execução de aplicações e novas aplicações podem ser compostas a partir de serviços existentes alterando a especificação das mesmas, sem mudar a estrutura do sistema de gerenciamento de banco de dados (SGBD) em cada banco de dados. Uma SODA pode ser adotada para um sistema que possui a capacidade de suportar serviços de busca, consulta, armazenamento e remoção de dados (CHOW et al.,

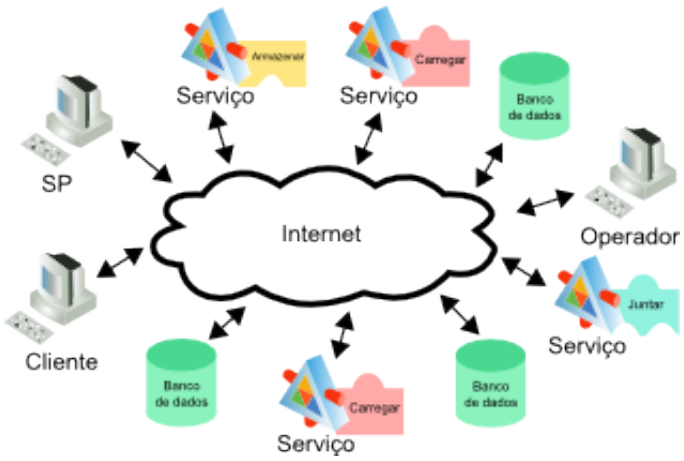


Figura 2.2: Arquitetura distribuída SODA

2009). Um sistema com SODA possui as seguintes características:

- Cada componente de banco de dados pode ser classificado de acordo com seus tipos de funções como operações de dados (inserir, apagar, alterar), controle de concorrência, provisionamento e formatação de dados e cada função pode ser visto como um serviço.
- As informações hospedadas no sistema podem ser atadas aos serviços, e não aos sistemas de banco de dados específicos. A idéia de armazenar e representar dados usando serviços provê um mecanismo flexível de armazenamento de dados em que a informação não está mais sujeita a um esquema estrutural. Isso provê a capacidade de acessar e integrar dados de sistemas heterogêneos.
- A construção de um mecanismo de auto-configuração permite que serviços individuais sejam combinados, configurados e/ou reconfigurados quando em execução.

Alguns sistemas de gerenciamento de banco de dados foram propostos usando a SODA em Campbell (2005) e Tok e Bressan (2006).

2.2 Rede de Petri (RdP)

A rede de Petri (RdP), como uma ferramenta gráfica e matemática, provê um ambiente uniforme para modelagem, análise formal e projeto de sistemas a eventos discretos (SEDs) (ZURAWSKI; ZHOU, 1994) (ADAM; ATLURI; HUANG, 1998) (NASSAR et al., 2008) e é muito efetiva como técnica de descrição e especificação de processos (MORALES; MELO;

MIYAGI, 2007) (YOO; JEONG; CHO, 2010) (HAMADI; BENATALLAH, 2003). É uma representação que pode ser usada tanto no nível conceitual quanto no nível funcional, em que o sistema pode ser analisado e validado antes de prosseguir com o projeto detalhado e a implementação (ADAM; ATLURI; HUANG, 1998). A RdP é uma ferramenta de comunicação entre pessoas relacionadas ao projeto, permitindo uma fácil interpretação e nítida identificação dos estados e ações (NASSAR et al., 2008). Possui a vantagem de que um mesmo modelo pode ser usado para análise das propriedades comportamentais e para validação de desempenho, assim como desenvolvimento sistemático de simuladores e controladores (ZURAWSKI; ZHOU, 1994). Possibilita a representação da dinâmica do sistema e sua estrutura em muitos níveis de abstração (NASSAR et al., 2008). Pode ser usada para modelar propriedades de sistemas como sincronização de processos, eventos assíncronos, operação concorrente e conflito ou compartilhamento de recursos (ZURAWSKI; ZHOU, 1994) (ADAM; ATLURI; HUANG, 1998) (NASSAR et al., 2008). Como ferramenta gráfica, provê um meio de comunicação entre os usuários, como o engenheiro de requisitos, e o cliente (ZURAWSKI; ZHOU, 1994). Matematicamente, a RdP pode ser descrita como um conjunto de equações algébricas lineares e, por isso, pode ser usada para a verificação formal de relações de precedência entre eventos, operações concorrentes, sincronização apropriada, inexistência de *deadlock*, atividades repetitivas e exclusão mútua de recursos compartilhados (ZURAWSKI; ZHOU, 1994) (NASSAR et al., 2008).

Desde sua apresentação por Carl Adam Petri (BRAUER; REISIG, 2006), a RdP tem sido usada na modelagem e análise de diferentes tipos de sistemas e aplicações tais como: protocolos distribuídos (KANESHIRO et al., 2008), aplicações industriais (ZURAWSKI; ZHOU, 1994) (NASSAR et al., 2008), tratamento de falhas (MORALES; MELO; MIYAGI, 2007), fluxo de processos (KIEPUSZEWSKI; HOFSTED; AALST, 2003), controle supervisório (LEE; ZHOU; HSU, 2005) (MELO; MORALES; MIYAGI, 2008), entre outras.

2.2.1 Formalização

A RdP é uma 5-tupla $PN = (P, T, F, W, M_0)$ (MURATA, 1989) (ZURAWSKI; ZHOU, 1994) (ADAM; ATLURI; HUANG, 1998), com P e T como conjuntos finitos, não nulos e disjuntos. P é o conjunto de lugares e T é o conjunto de transições. F é o conjunto dos relacionamentos entre lugares e transições e é também chamado de conjunto de arcos orientados. W é um conjunto de números naturais não nulos que representam os pesos de cada arco orientado. M_0 é um conjunto de números naturais que representa a quantidade inicial de marcas nos lugares, que também é conhecido como marcação dos lugares (MURATA, 1989) (LI; ZHOU,

2008) (ZURAWSKI; ZHOU, 1994) (YOO; JEONG; CHO, 2010) (HAN et al., 2008) (XU et al., 2007).

A marcação num lugar p é denotada por $M(p)$ e a marcação na rede é indicada por M.

A estrutura de uma RdP é uma 4-tupla $N = (P,T,F,W)$ (MURATA, 1989). Uma RdP pode ser dita ordinária se $\forall f \in F, W(f) = 1$ e, por isso, sua estrutura é denotada como $N = (P,T,F)$ (LI; ZHOU, 2008).

A rede $N = (P,T,F,W)$ é dita pura se, e somente se, não há *loops* $\nexists (x,y) \in (P \times T) \cup (T \times P)$, $(x,y) \in F \wedge (y,x) \in F$ (LI; ZHOU, 2008).

Outra representação formal da RdP é por meio de matriz de incidência e equações de estados. A matriz de incidência numa RdP $N = (P,T,F,W)$ é representada por $[N] = [c_{ij}]$ indexada por P e T tal que $p_i \in P$, $t_j \in T$ e $f_{ij} \subseteq (P \times T) \cup (T \times P)$ com f_{ij} o arco orientado ligando o lugar p_i à transição t_j ou a transição t_j ao lugar p_i, $c_{ij} = W(f_{ij})$ (MURATA, 1989).

No caso de uma RdP ordinária e pura, a matriz $[N] = [c_{ij}]$ é tal que $c_{ij} = 1$ se $t_j \in {}^\bullet p_i \setminus p_i^\bullet$, $c_{ij} = -1$ se $t_j \in p_i^\bullet \setminus {}^\bullet p_i$ e $c = 0$ nos demais casos (LI; ZHOU, 2008) (ZURAWSKI; ZHOU, 1994) (DAVID; ALLA, 1994).

A dinâmica de uma RdP indica a forma de mudança de estados da mesma e as regras de mudança de estados da RdP é a base do sucesso dessa ferramenta na modelagem de sistemas dinâmicos (MURATA, 1989).

Em uma rede $N = (P,T,F,W)$ o pré-conjunto $x \in P \cup T$ é definido como ${}^\bullet x = y \in P \cup T | (y,x) \in F$. O pós-conjunto $x \in P \cup T$ é definido como $x^\bullet = y \in P \cup T | (x,y) \in F$. Os pré-conjuntos (pós-conjuntos) de um conjunto de elementos são definidos como a união dos pré-conjuntos (pós-conjuntos) desses elementos (LI; ZHOU, 2004).

Numa RdP ordinária e pura, uma transição $t \in T$ está habilitada em M (simbolicamente $M[t>$) se, e somente se, $\forall p \in {}^\bullet t : M(p) > 0$. Uma transição $t \in T$ habilitada pode ser disparada e como resultado desse disparo tem-se uma nova marcação M', que é denotado por $M[t > M'$, com $M'(p) = M(p) - 1$ se $p \in {}^\bullet t \setminus t^\bullet$; $M'(p) = M(p) + 1$ se $p \in t^\bullet \setminus {}^\bullet t$; e $M(p) = M'(p)$, nos demais casos, para todo $p \in P$ (LI; ZHOU, 2004) (ADAM; ATLURI; HUANG, 1998).

(N,M) é chamado de rede marcada e limitada se, e somente se, $\exists k \in \mathbb{N}^+$, $\forall M \in R(N,M_0)$ e $\forall p \in P$, tal que $M(p) \leq k$, em que $R(N,M_0)$ é o conjunto das marcações na rede N alcançáveis a partir de M_0 (LI; ZHOU, 2008).

2.2.2 Representação gráfica

A representação gráfica da RdP é composta de 2 elementos principais, os lugares e as transições. Os lugares são representados por círculos e as transições por barras (alguns autores representam as transições por caixas), como pode ser visto na Figura 2.3. Os lugares e as transições são conectados por arcos orientados de um lugar para uma transição ou de uma transição para um lugar. Em outras palavras, uma RdP é um grafo bipartido (DAVID; ALLA, 1994) (PETERSON, 1977) (MURATA, 1989).

Outro elemento gráfico da RdP é a marca que é representada cada uma por pequenos pontos sólidos dentro dos círculos que representam os lugares (PETERSON, 1977).

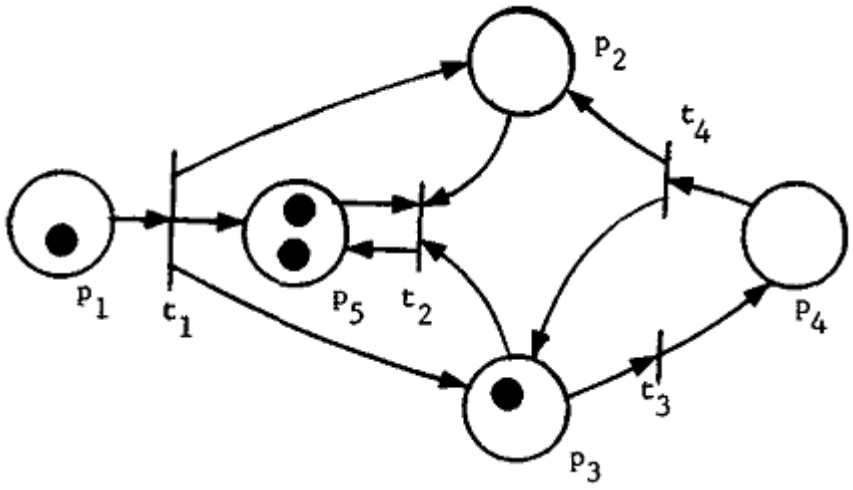


Figura 2.3: Elementos gráficos da RdP (PETERSON, 1977)

Graficamente a dinâmica da RdP se dá pelo consumo de marcas de um lugar ligado a uma transição habilitada, em que há um arco orientado do lugar para a transição, e a geração de marcas em outro lugar ligado a mesma transição habilitada, em que há um arco orientado da transição para o lugar, como na Figura 2.4. A figura ilustra também a influência do peso dos arcos no consumo e geração de marcas.

2.2.3 Propriedades

A RdP possui propriedades quando interpretadas no contexto do sistema modelado, permitem ao desenvolvedor do sistema identificar a presença ou ausência de comportamentos e funcionalidades específicas do domínio da aplicação do sistema em projeto. As propriedades podem ser distinguidas em comportamentais e estruturais (ZURAWSKI; ZHOU, 1994).

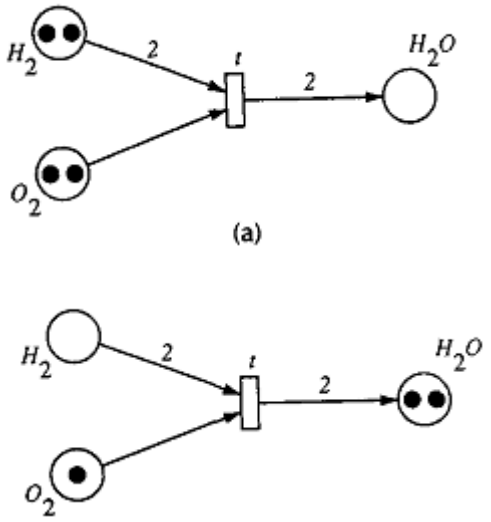


Figura 2.4: Representação da dinâmica em grafo da RdP (MURATA, 1989), em que (a) é a RdP antes do disparo de t e (b) é a RdP depois do disparo de t

As propriedades comportamentais são as propriedades que dependem do estado inicial da RdP, ou seja, do conjunto de marcas iniciais (marcação M_0). Já as propriedades estruturais não dependem do conjunto de marcas iniciais da RdP, mas sim de sua topologia (ZURAWSKI; ZHOU, 1994). As propriedades de interesse nesse trabalho são as comportamentais.

Alcançabilidade

Sistemas distribuídos podem ser avaliados pela sua capacidade de alcançar um estado específico. Uma forma de verificar se um sistema pode alcançar um estado desejado ou indesejado é modelá-lo com RdP e verificar se o modelo é capaz de alcançar tal estado.

Para verificar se o modelo pode alcançar um estado específico é feita uma busca por uma sequência de disparos de transições que resultariam na transformação de uma marcação M em uma marcação M_i, em que M_i representa o estado específico e a sequência de disparos representa o comportamento funcional requerido (ZURAWSKI; ZHOU, 1994).

A existência de mais de uma sequência de disparos de transições que levam uma RdP com marcação de M para uma marcação M_i indica que o modelo em RdP possui formas alternativas para que certos estados sejam alcançados. Pode também indicar a presença de comportamentos funcionais imprevistos do sistema real (ZURAWSKI; ZHOU, 1994).

Dada uma rede (N,M), se $\exists t \in T$ tal que $M[> M'$ então M' é alcançável de M ou, se $\exists [t_i] \in T$ tal que $M[t_1 > M_1[t_2 > M_2[t_3 > \ldots M_{k-1}[t_k > M_k$, então M_k é alcançável de M (HAN

et al., 2008). Em outras palavras, a marcação M_i é dita alcançável a partir de uma marcação M se existe uma sequência de disparos de transições que transformam M em M_i, $M_i \in R(N,M)$ (ZURAWSKI; ZHOU, 1994).

Limitabilidade e Segurança

Na modelagem de diversos sistemas, os lugares da RdP são frequentemente usados para representar áreas de armazenamento de informação e em sistemas de manufatura são usados em geral pra representar armazenamento de produtos e ferramentas. Nestes casos, é preciso garantir que as áreas de armazenamento não excedam sua capacidade (ZURAWSKI; ZHOU, 1994).

Em RdP, a propriedade que verifica a capacidade de um lugar é a limitabilidade. Uma rede é dita k-limitada se o número de marcas em qualquer lugar p, em que $p \in P$, é sempre menor ou igual à k (com $k \in \mathbb{N}^*$) para toda marcação M alcançável do estado inicial M_0 (ZURAWSKI; ZHOU, 1994). Ou então, em uma rede (N,M), $p \in P$, se $\exists B > 0$ tal que $\forall M \in R(N,M_0)$ e $M(p) \leq B$, então p é limitado. Define-se que o limite B de uma rede (N,M) é $max\{B\}$, $\forall p \in P$ (HAN et al., 2008). A Figura 2.5 ilustra uma RdP não limitada, pois o lugar p_4 pode receber infinitas marcas.

Uma rede é dita segura se ela é 1-limitada, ou seja, $B = 1$ para todos os lugares da rede (N,M) (HAN et al., 2008) (ZURAWSKI; ZHOU, 1994). A Figura 2.6 ilustra uma rede segura.

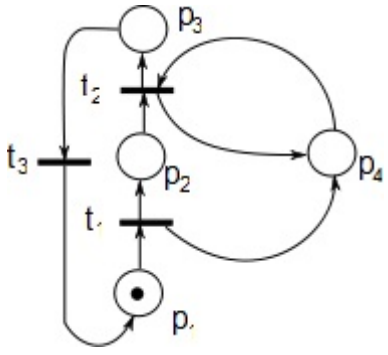


Figura 2.5: Exemplo de rede de Petri não limitada (ZURAWSKI; ZHOU, 1994)

Conservadorismo

Em sistemas reais, o número de recursos em uso é limitado por restrições físicas, financeiras e outras. Se as marcas são usadas para representar os recursos e seu número é limitado e fixo, então o número total de marcas no modelo da RdP deve permanecer sempre inalterado

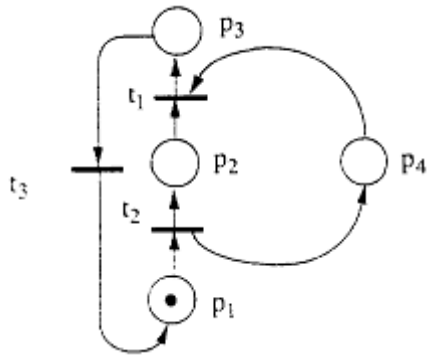


Figura 2.6: Exemplo de rede de Petri segura (ZURAWSKI; ZHOU, 1994)

independentemente do estado (marcação) em que a rede se encontra (ZURAWSKI; ZHOU, 1994).

Uma RdP é dita conservativa se o número de marcas é constante independentemente do disparo das transições. Estruturalmente isso só é possível se o número de arcos de entrada de cada transição é igual ao número de arcos de saída. Porém, em sistemas reais os recursos são frequentemente agrupados e utilizados em um conjunto de tarefas e ao final delas esses recursos são separados (ZURAWSKI; ZHOU, 1994).

Então, se $\exists m$, um vetor de números inteiros associado a cada lugar $p \in P$ $m = [m_1, m_2, \ldots, m_k]$, em que k é o número de lugares da rede e a soma ponderada pelo vetor m das marcações M a partir de uma marcação inicial M_0 permanecer constante a rede é dita conservativa (ZURAWSKI; ZHOU, 1994), dois casos de redes conservativas podem ser vistos nas Figuras 2.7 e 2.8.

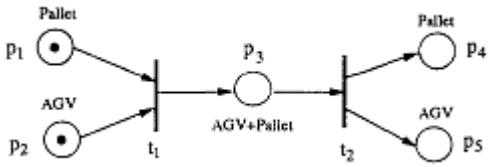


Figura 2.7: Exemplo de rede de Petri conservativa (ZURAWSKI; ZHOU, 1994)

Vivacidade

O conceito de vivacidade é fortemente relacionado à situação de *deadlock* (estado em que a RdP não pode disparar nenhuma de suas transições). Em Zurawski e Zhou (1994) são mostradas 4 condições para ocorrência de *deadlock*:

- **Exclusão mútua:** um recurso ou está disponivel ou atribuído a um processo que tem acesso exclusivo a este recurso;
- **Retenção e espera:** um processo é autorizado a reter um recurso enquanto requer mais recursos;
- **Não preempção:** um recurso alocado para um processo não pode ser removido do processo até que este seja liberado pelo próprio processo;
- **Espera circular:** dois ou mais processos são arranjados em uma cadeia em que cada processo aguarda a liberação de recursos retidos pelo próximo processo na cadeia.

Uma rede é considerada viva se $\forall M$, $M_0[>M$, isto é, ela pode progredir por meio de alguma sequência de disparo. A partir de uma M_0, a rede pode não estar imediatamente em *deadlock*, mas pode existir uma limitação quanto a sequência de disparos que pode levar a rede a um estado de *deadlock* (ZURAWSKI; ZHOU, 1994) (HAN et al., 2008). Por isso existem diferentes níveis de vivacidade para uma transição t e uma marcação M_0, esses diferentes níveis podem ser analisados segundo o conjunto de todas possíveis sequências de disparos a partir de M_0 representada por $L(M_0)$. Uma transição t em uma RdP é dita:

- $L0$-viva (ou morta) se não houver sequência de disparos em $L(M_0)$ em que t possa disparar;
- $L1$-viva (potencialmente disparável) se t pode ser disparada pelo menos uma vez em alguma sequência de disparos em $L(M_0)$;
- $L2$-viva se t pode ser disparada pelo menos k vezes em alguma sequência de disparos em $L(M_0)$, dado qualquer $k > 1$ inteiro positivo;
- $L3$-viva se t pode ser disparada infinitamente, frequentemente, em alguma sequência de disparos em $L(M_0)$;
- $L4$-viva (ou viva) se t é $L1$-viva em todas as marcações em $R(N,M_0)$.

Um caso de rede de Petri viva pode ser visto na Figura 2.8 e todas suas transições são $L4$-vivas. Já na Figura 2.9 a transição t_0 é $L0$-viva, para a marcação inicial M_0, a transição t_1 é $L1$-viva, a transição t_2 é $L2$-viva e a transição t_3 é $L3$-viva.

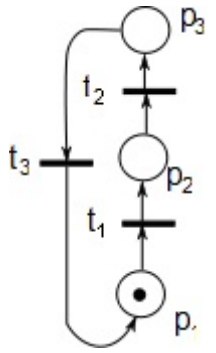


Figura 2.8: Exemplo de rede de Petri conservativa e viva (ZURAWSKI; ZHOU, 1994)

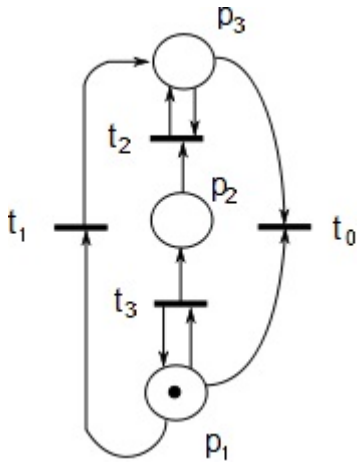


Figura 2.9: Exemplo de RdP com diferentes níveis de vivacidade (ZURAWSKI; ZHOU, 1994)

Reversibilidade

Um importante problema na operação de sistemas reais, como os de manufatura ou de controle de processos é a capacidade esperada destes sistemas de recuperação (retorno ao estado normal) depois da ocorrência de falhas ou erros. Isto é, esses sistemas precisam retornar de um estado de falha para um estado anterior correto. Esse comportamento é intimamente relacionada à propriedade de reversibilidade da RdP (ZURAWSKI; ZHOU, 1994).

Uma rede é considerada reversível se M é alcançável de $R(N,M_0)$ e M_0 é alcançável de $R(N,M)$ (ZURAWSKI; ZHOU, 1994), a Figura 2.6 ilustra uma rede reversível e a Figura 2.5 ilustra uma rede irreversível.

2.3 Modelagem da arquitetura de controle do SP

2.3.1 Modelo de WSs

Segundo (HAN et al., 2008) para modelar um sistema baseado em WSs é necessário considerar os seguintes aspectos:

- Os modelos dos processos devem ter duas abordagens, uma lógica e outra física;
- O método de modelagem lógica do sistema baseado em WSs deve ser de fácil entendimento para o gerente de negócios;
- O método de modelagem lógica deve ser também intuitivo, visualizável e conveniente para ser usado por usuários, especialmente por gerentes e analistas de negócios;
- Deve possuir uma técnica efetiva de análise para avaliar o quão correto é o modelo lógico;
- O modelo lógico deve ser conveniente para ser transformado num modelo físico.

Os sistemas de informação baseados em WSs podem ser abordados de duas formas: do ponto de vista de engenharia de software ou do ponto de vista de engenharia do conhecimento. A engenharia de software preocupa-se com análise de algoritmos e de dados. A engenharia do conhecimento preocupa-se com sintaxe e semântica, em que se identifica os serviços necessários e o relacionamento lógico entre eles (HAN et al., 2008).

De acordo com Hamadi e Benatallah (2003) um WS pode ser descrito por uma tupla $S = (NameS, Desc, Loc, URL, CS, SN)$, em que $NameS$ é o nome do serviço (usado como identificador do mesmo), $Desc$ é a descrição do serviço provido (um resumo do serviço provido), Loc o servidor (ou provedor) onde está localizado o serviço, URL é a chamada do serviço, CS é o conjunto de componentes de serviços (se $CS = NameS$ então S é um serviço básico ou atômico, caso contrário S é um serviço composto de serviços atômicos) e $SN = (P, T, F, W, M_0)$ é o modelo em RdP do comportamento dinâmico do serviço.

A combinação de serviços pode ser descrita algebricamente como:

$$S ::= \varepsilon | X | S \bigodot S | S \bigoplus S | S \diamond S | \mu S | (S|S) \rightsquigarrow S \tag{2.1}$$

em que:

- ε é um serviço vazio, ou que não realiza nenhuma operação, Figura 2.10;
- X é um serviço constante, usado como um serviço atômico ou básico neste contexto, Figura 2.11;
- $S_1 \odot S_2$ representa um serviço composto que executa o serviço S_1 seguido pelo serviço S_2, Figura 2.12;
- $S_1 \oplus S_2$ representa um serviço composto que executa ou o serviço S_1 ou o serviço S_2, Figura 2.13;
- $S_1 \diamond S_2$ representa um serviço composto que executa o serviço S_1 seguido pelo serviço S_2 ou o serviço S_2 seguido pelo serviço S_1, Figura 2.14;
- μS representa um serviço que executa algumas vezes o serviço S iterativamente, Figura 2.15;
- $(S_1 \oplus S_2) \odot S_3$ representa um serviço composto que espera pela execução de um serviço, S_1 ou S_2, para em sequência ativar o serviço S_3 e aguarda que a execução dos 3 serviços (S_1, S_2 e S_3) tenham sido concluídas para obter o resultado, Figura 2.16;

Figura 2.10: Representação de um serviço vazio (HAMADI; BENATALLAH, 2003)

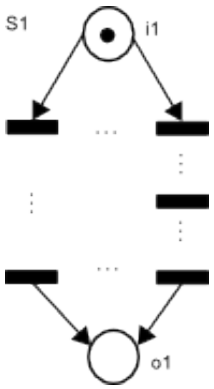


Figura 2.11: Representação de um serviço constante (HAMADI; BENATALLAH, 2003)

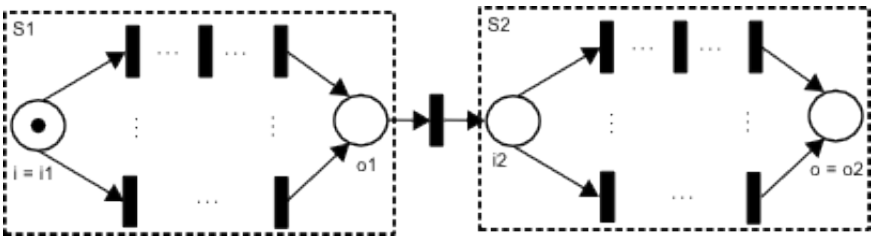


Figura 2.12: Representação de um serviço composto $S_1 \odot S_2$ (HAMADI; BENATALLAH, 2003)

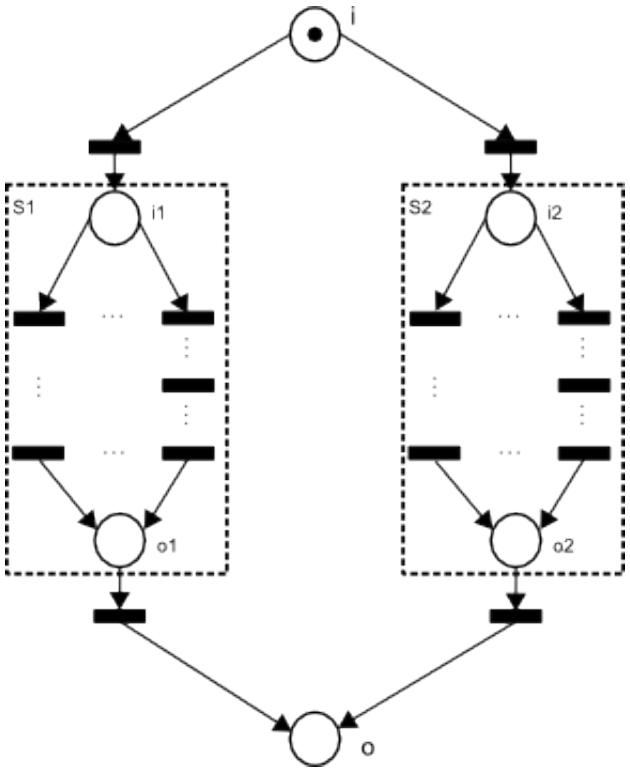


Figura 2.13: Representação de um serviço composto $S_1 \oplus S_2$ (HAMADI; BENATALLAH, 2003)

O serviço atômico da Figura 2.11 é representado por transições que podem estar em série ou em paralelo que são os processos realizados dentro de um único serviço. O serviço composto em sequência da Figura 2.12 é representado por dois serviços atômicos em que a resposta do primeiro serviço atômico é usada como entrada do segundo serviço atômico. O serviço composto em conflito da Figura 2.13 é representado por dois serviços atômicos que para a mesma entrada são capazes de gerar as mesmas respostas e por isso um dos serviços é alternativo ao outro serviço.

O serviço composto da Figura 2.14 é representado por dois serviços atômicos em que a operação destes pode ser dita comutativa, pois os serviços podem ser chamados em qualquer ordem. Na figura, a entrada i marca os lugares $p1$, $p2$ e $p3$. O lugar $p3$ entra em conflito

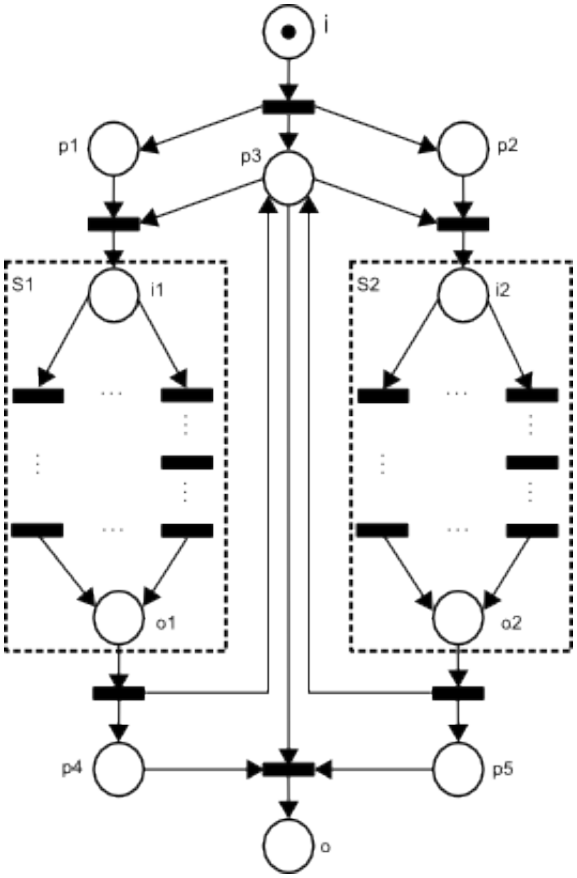


Figura 2.14: Representação de um serviço composto $S_1 \diamond S_2$ (HAMADI; BENATALLAH, 2003)

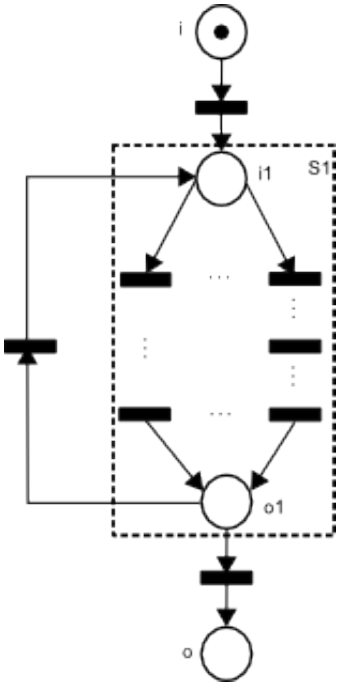


Figura 2.15: Representação de um serviço composto μS (HAMADI; BENATALLAH, 2003)

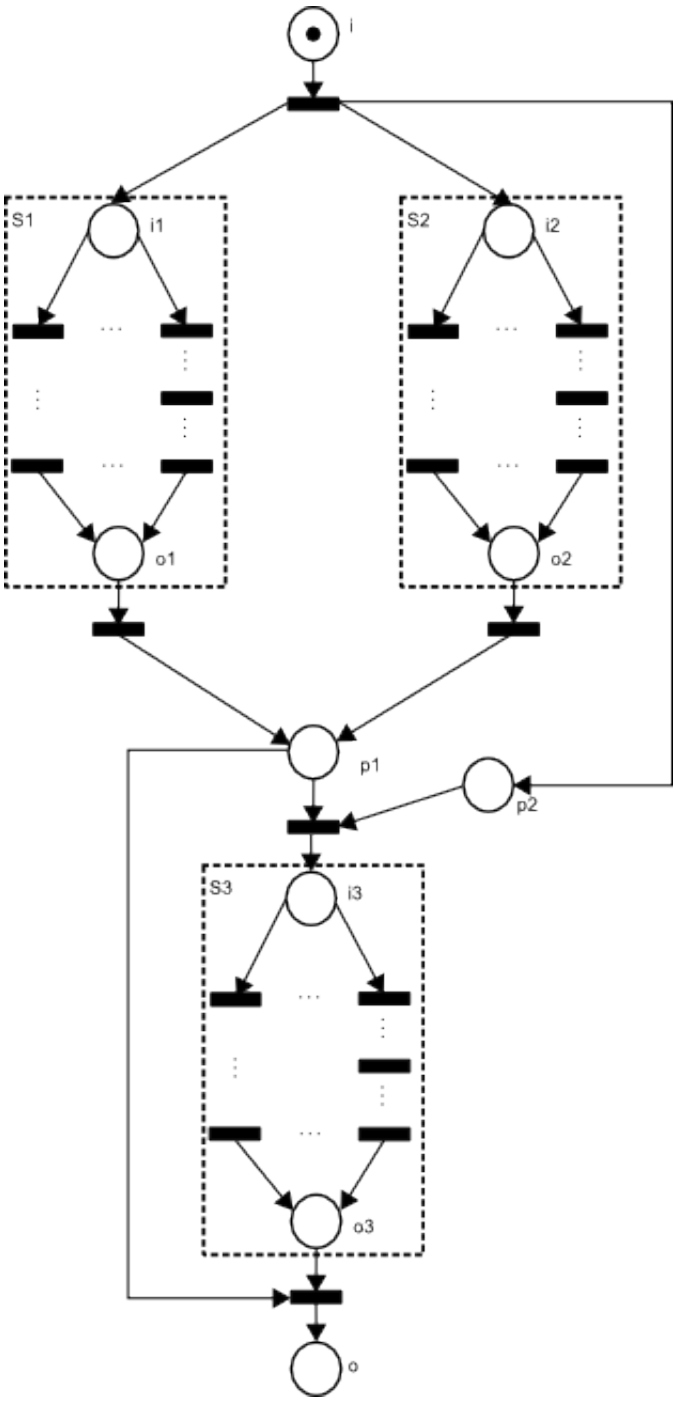


Figura 2.16: Representação de um serviço composto $(S_1 \bigoplus S_2) \bigodot S_3$ (HAMADI; BENATALLAH, 2003)

para escolher o serviço $S1$ ou $S2$. Após escolher um desses, ou o lugar $p1$ ou o $p2$ perde sua marca e o lugar $p3$ também. Ao final desse serviço, uma marca estará no lugar $p4$ se o serviço executado foi o $S1$ ou $p5$ se o serviço executado foi $S2$ e uma marca retorna ao lugar $p3$. Então o outro serviço será executado e ao final as marcas $p3$, $p4$ e $p5$ terminarão o uso do serviço.

O serviço composto da Figura 2.15 é representado por um serviço atômico que pode invocar a si próprio até que alguma condição esteja satisfeita.

O serviço composto da Figura 2.16 é representado por três serviços atômicos. Este pode ser considerado um discriminador que consiste em invocar dois serviços com mesmas funcionalidades paralelamente. Como os serviços possuem atrasos em diferentes níveis, para reduzir o atraso em uma sequência a resposta de um deles é usada para invocar o serviço seguinte e a resposta do outro então não causa atraso a operação.

Os WSs são aplicações de *software* que podem ser encontradas, descritas e acessadas por arquivos baseados em XML e protocolos padrões da *Web*. As tecnologias de associadas ao WS, como *Universal Description, Discovery and Integration* (UDDI) e registros ebXML, permitem às aplicações encontrar dinamicamente informações sobre o WS. A sintaxe de mensagem para um WS é descrita em *Web Service Definition Language* (WSDL). O *Simple Object Access Protocol* (SOAP) é um protocolo de mensagens baseado em XML para acesso ao WS (LI et al., 2010) (SHAN et al., 2009). O *Business Process Execution Language for Web Services* (BPEL4WS) é uma linguagem de descrição de processo baseada em XML, proposta pela IBM, BEA e Microsoft, e é um padrão para integração de WS. O BPEL4WS usa uma estrutura em 4 camadas: (a) a camada de cliente com navegador *Web*; (b) camada de negócios, em que é feita a integração de serviços; (c) camada de serviços, em que os serviços são disponibilizados; e (d) camada de dados, em que as informações são armazenadas. A Figura 2.17 representa a estrutura do BPEL4WS (com uma aplicação em *Java Database Conectivity* (JDBC), que é um conjunto de classes e interfaces para envio de instruções para bancos de dados) (SHAN et al., 2009).

2.3.2 Production Flow Schema (PFS) para estruturação de WSs

O *Production Flow Schema* (PFS) é uma interpretação da RdP do tipo Canal-Agente própria para sistematizar a construção de modelos na área de SPs. O PFS mostra, ao nível conceitual, como os itens envolvidos num processo produtivo são objetos da execução de funções que são necessárias para obter o produto ou serviço desejado. Um importante conceito apresentado pelo PFS é o fluxo de itens (materiais ou informações), que é associado a alocação e desalocação de recursos. O PFS indica que as atividades ou etapas de um processo envolvem a interação entre itens, seus fluxos e outras atividades (HASEGAWA et al., 1999).

O grafo de PFS apresenta os elementos do sistema organizados hierarquicamente, utilizando uma abordagem *top-down* explorando o conceito de macro-eventos. A técnica foi desenvolvida para sistematizar e facilitar a modelagem por RdP (MIYAGI, 1996).

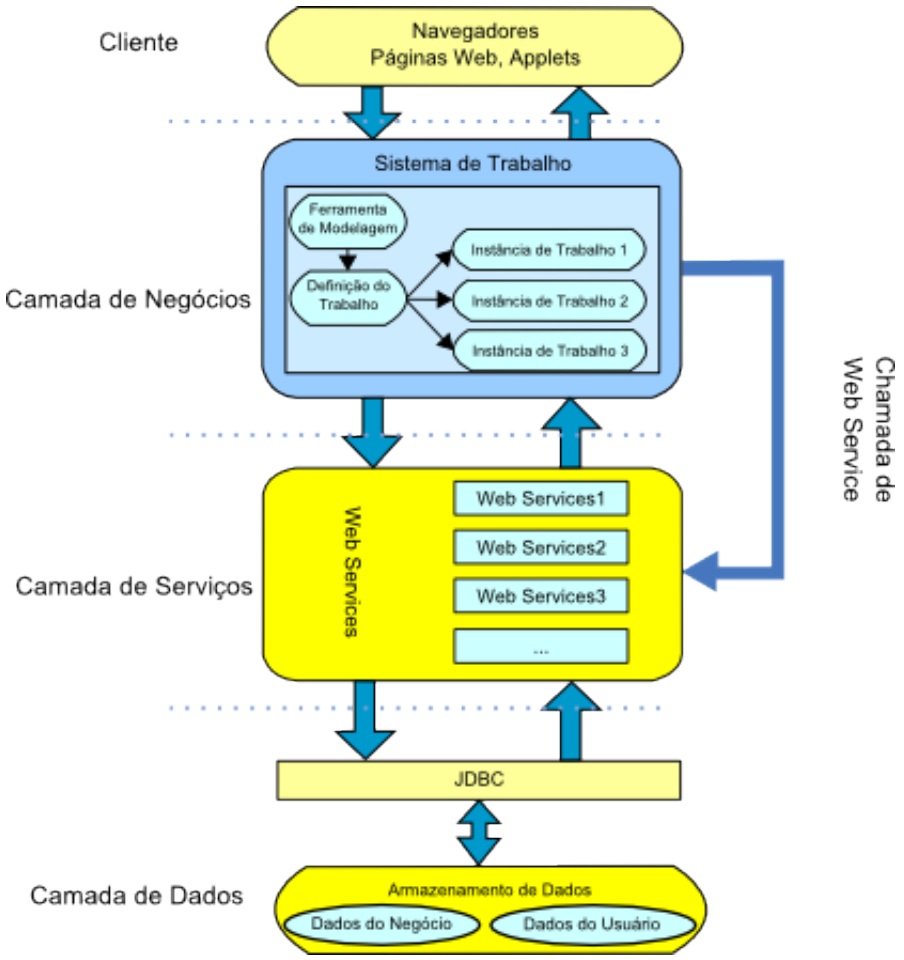


Figura 2.17: Estrutura em 4 camadas do BPEL4WS (SHAN et al., 2009)

O PFS possui 3 elementos estruturais básicos (MIYAGI, 1996):

- **Elementos ativos ou atividades:** Representam os macro-eventos, elementos que podem ser detalhados internamente obtendo um novo PFS ou alguma outra RdP interpretada, graficamente representado como na Figura 2.18;
- **Elementos passivos ou distribuidores:** Similares ao lugares na RdP, esses elementos são capazes de representar o acúmulo de itens e são responsáveis pela habilitação de uma atividade, representado como na Figura 2.19;
- **Arcos:** Elementos que fornecem as relações entre os distribuidores e as atividades, representado na Figura 2.20;

Os distribuidores em PFS são elementos passivos e são responsáveis apenas por indicar os itens que fluem pelo grafo. Já as atividades são elementos ativos e são responsáveis por indicar a ocorrência de um evento como produção, transporte e modificação de itens. Os arcos

[ATIVIDADE]

Figura 2.18: Elementos ativos no PFS

Figura 2.19: Elementos passivos no PFS

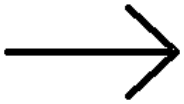


Figura 2.20: Elementos arcos no PFS

são os elementos que indicam o fluxo dos itens dentro do grafo, em que estes podem ir de um distribuidor para uma atividade ou de uma atividade para um distribuidor (MIYAGI, 1996). O PFS pode representar o fluxo principal de itens (Figura 2.21) ou representar relações de comunicação assíncrona (Figura 2.22) que explora o conceito de fluxo secundário de informação (GUSTIN, 2000).

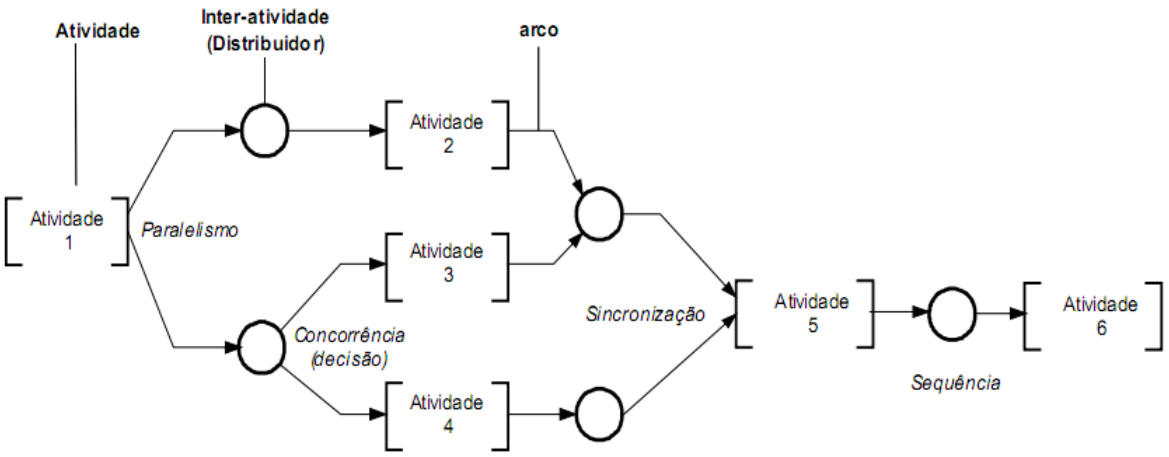


Figura 2.21: Exemplo de fluxo principal e propriedades de SEDs representados no PFS (GUSTIN, 2000)

A metodologia PFS é baseada em um procedimento de refinamento gradativo (refinamentos sucessivos), substituindo uma atividade ou um distribuidor por um PFS ou uma RdP interpretada. Uma condição de contorno para o refinamento é que os arcos que estão conec-

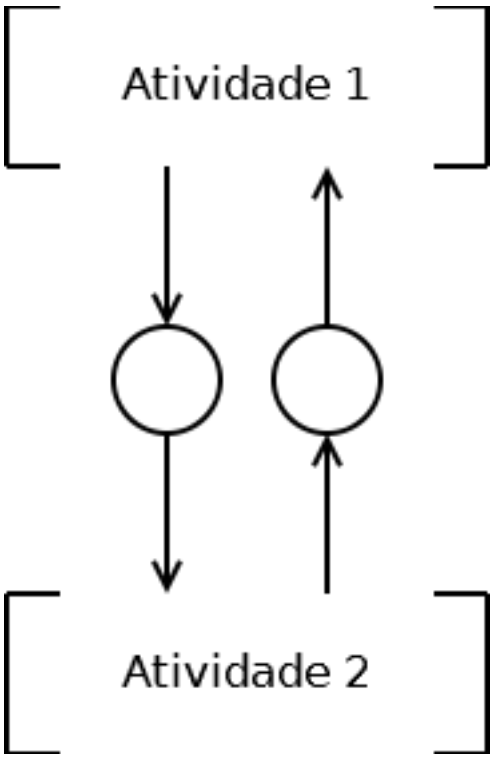


Figura 2.22: Comunicação assíncrona de atividades (GUSTIN, 2000)

tados no modelo original da atividade ou do distribuidor também estejam presentes no modelo refinado. Um refinamento terá sido executado corretamente se a interpretação dos sub-grafos que substituem distribuidores ou atividades mantém as funcionalidades/comportamentos do grafo original (MIYAGI, 1996).

O refinamento de um elemento atividade é feito substituindo essa por um grafo contendo novas atividades (do PFS) ou transições (da RdP) unidas entre si por elementos distribuidores, como na Figura 2.23. O refinamento de um elemento distribuidor é feito substituindo esse por um grafo contendo novos distribuidores (do PFS) ou lugares (da RdP), unidos entre si por elementos atividades, como na Figura 2.24 (GUSTIN, 2000).

Um exemplo do uso da metodologia PFS na modelagem de SPs pode ser visto no trabalho de Melo et al. (2008) e outro exemplo na modelagem da integração de sistemas em edifícios inteligentes pode ser visto em Miyagi et al. (2002).

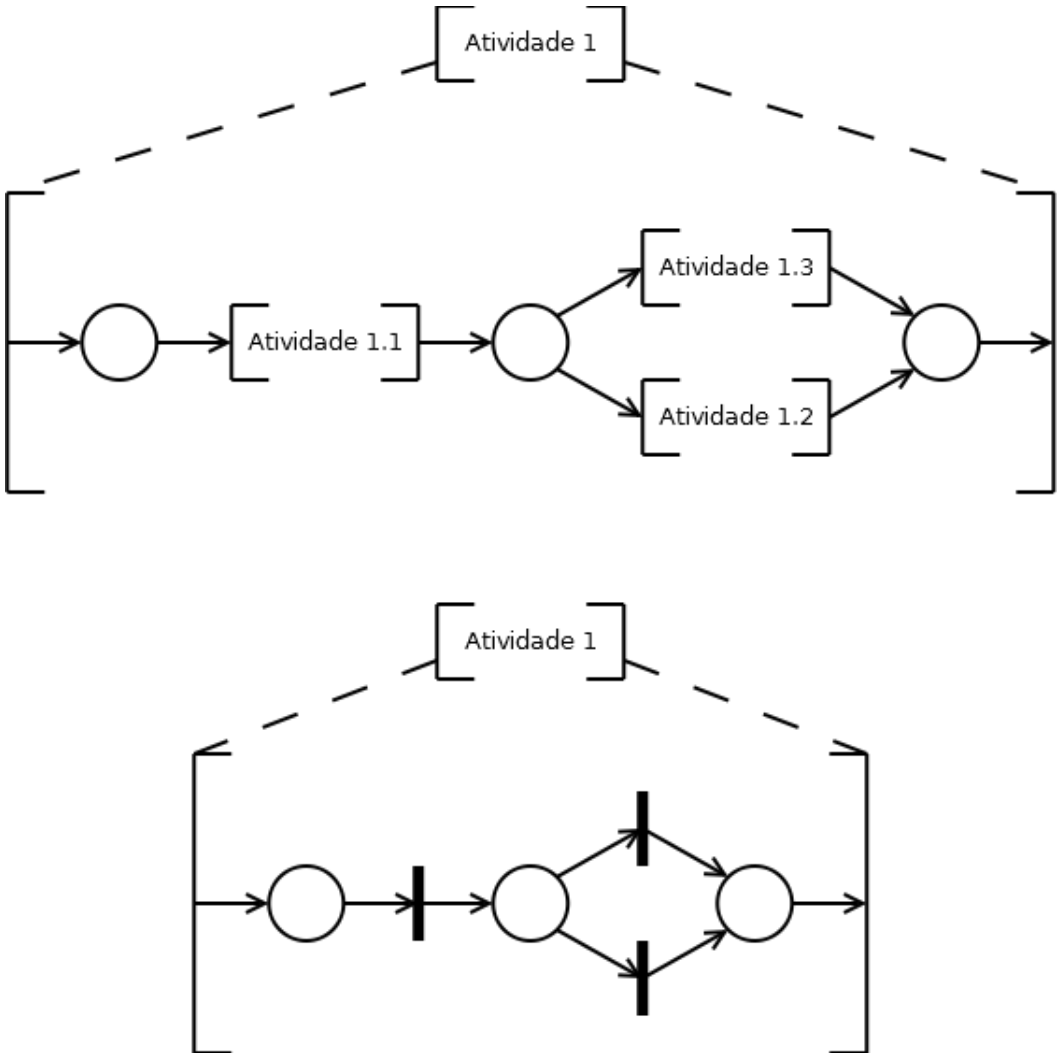


Figura 2.23: Refinamento de atividades

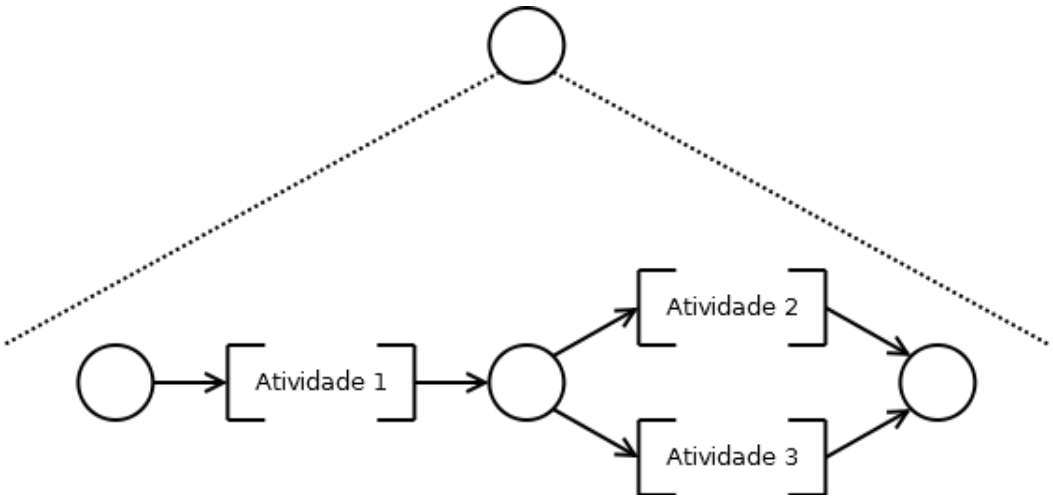


Figura 2.24: Refinamento de distribuidores

3 Sistema produtivo disperso

Conforme anteriormente apresentado, os sistemas produtivos (SPs) são sistemas que realizam processos utilizando recursos materiais, equipamentos, recursos humanos, informações e outras entidades físicas, para a produção de bens ou serviços. Os equipamentos, recursos materiais e recursos humanos apresentam custo elevado dentro dos SPs e por isso não é viável a implementação real da arquitetura de controle estudado no presente trabalho em SPs em operação nas empresas.

Por outro lado é fundamental que as propostas desenvolvidas sejam efetivamente implementadas e testadas em um caso real. Assim, esse trabalho desenvolveu os estudos de caso utilizando um sistema flexível de montagem automatizado (SFMA) que está disponível e em operação no laboratório de Sistemas de Automação de EPUSP e que emula o funcionamento de um SP disperso.

Este sistema é composto por vários módulos, cada um com seu respectivo controlador de modo que podem funcionar de modo independente como se fossem, cada um, uma planta industrial autônoma. Cada um dos módulos (estações de trabalho) é estruturado de forma que cada um dos controladores está conectado a um microcomputador por meio de um *hardware* dedicado. Com isto, cada módulo possui autonomia de funcionamento que emula uma planta e seus processos produtivos. Cada microcomputador, por sua vez, está conectado à internet como se fosse uma conexão de uma fabrica independente. Além disso, esta estrutura permite considerar os operadores também fisicamente distribuídos e atuando no sistema via internet, assim como os clientes que encomendam os pedidos e/ou desejam monitorá-los, como na Figura 3.1.

A Figura 3.2 ilustra o SFMA e os subsistemas que o compõe (*"SP1-Alimentação"*, *"SP2-Inspeção"*, *"SP3-Transporte"* e *"SP4-Montagem"*).

A arquitetura de controle deve assim coordenar de diferentes serviços dos SPs que estão interligados por meio de uma rede de comunicação (internet) para troca de mensagens. Neste caso, cada um dos SPs fornece serviços específicos. Assim, o serviço *"SP2-Inspesão"* encapsula

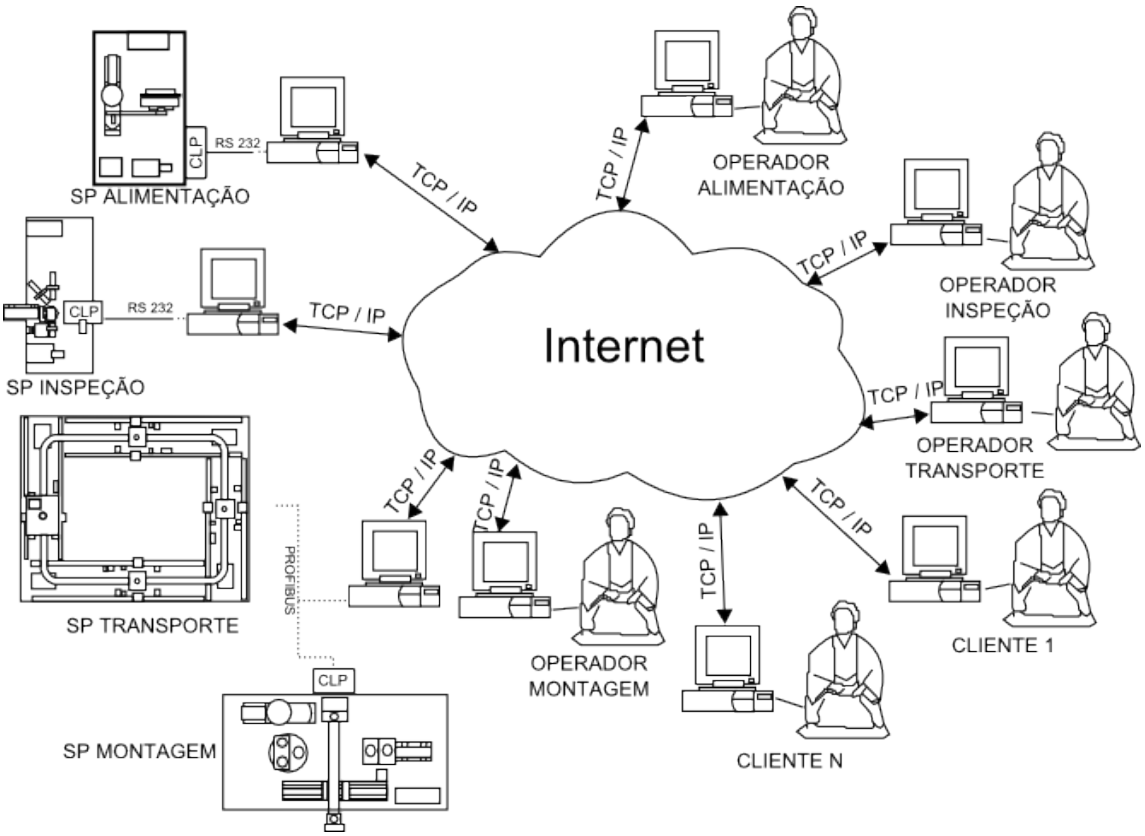


Figura 3.1: Exemplo de SP disperso

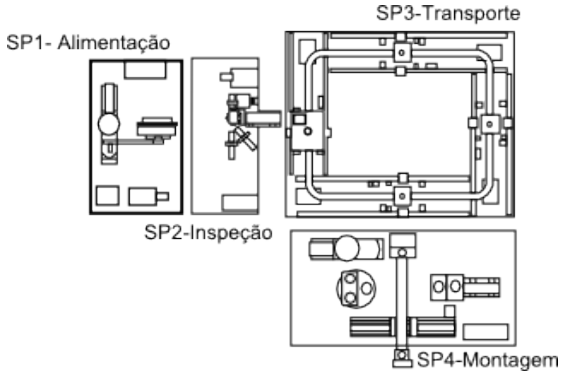


Figura 3.2: Subsistemas do SFMA que emula um SP disperso

o serviço de um sistema produtivo de inspeção. Esse serviço está relacionado com o controle de qualidade e identificação das características físicas do produto fornecido pelo serviço do subsistema *"SP1-Alimentação"*, que fornece da peça *"corpo"*, de maneira que sejam atendidas as especificações do produto requisitado pelo serviço do subsistema *"SP4-Montagem"* onde se realiza o serviço de montagem dos produtos finais ou *"peças montadas"*. Essas peças podem ser de três tipos, dependendo das seguintes variáveis: *"corpo"* prata com *"êmbolo"* preto, *"mola"* e *"tampa"*; *"corpo"* rosa com *"êmbolo"* preto, *"mola"* e *"tampa"*; e *"corpo"* preto com *"êmbolo"*

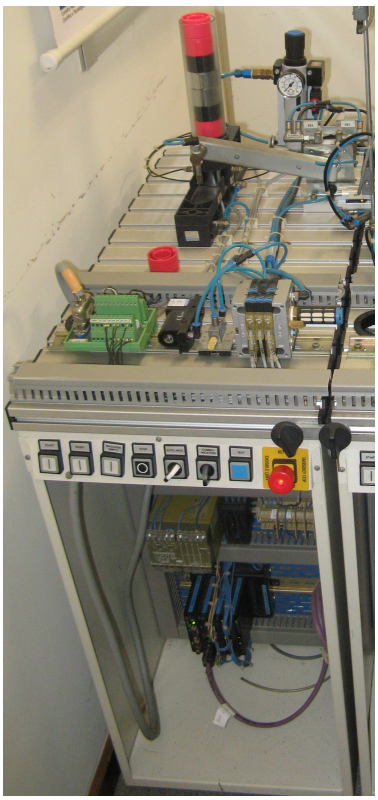


Figura 3.3: Foto do *"SP1-Alimentação"*

prata, *"mola"* e *"tampa"*. O serviço do subsistema *"SP3-Transporte"* executa o transporte de peças do subsistema *"SP2-Inspeção"* para o subsistema *"SP4-Montagem"* e a retirada dos produtos finais do subsistema *"SP4-Montagem"*. As solicitações de tipo e quantidade de produtos finais são efetuadas por um cliente, via internet.

O desenvolvimento de cada atividade produtiva definida no processo produtivo de cada SP é executado em um *"Controle local"*. As peças que compõem o produto final, *"corpo"*, *"pino"*, *"mola"* e *"tampa"*, são ilustradas na Figura 3.7.

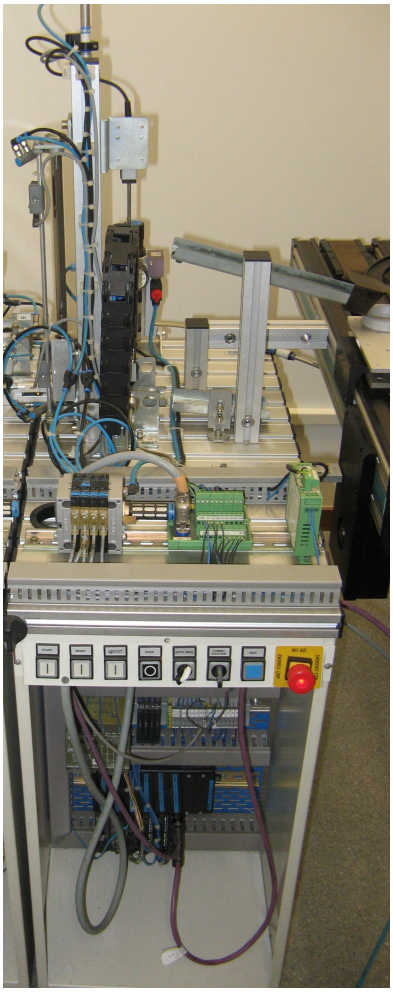


Figura 3.4: Foto do *"SP2-Inspeção"*

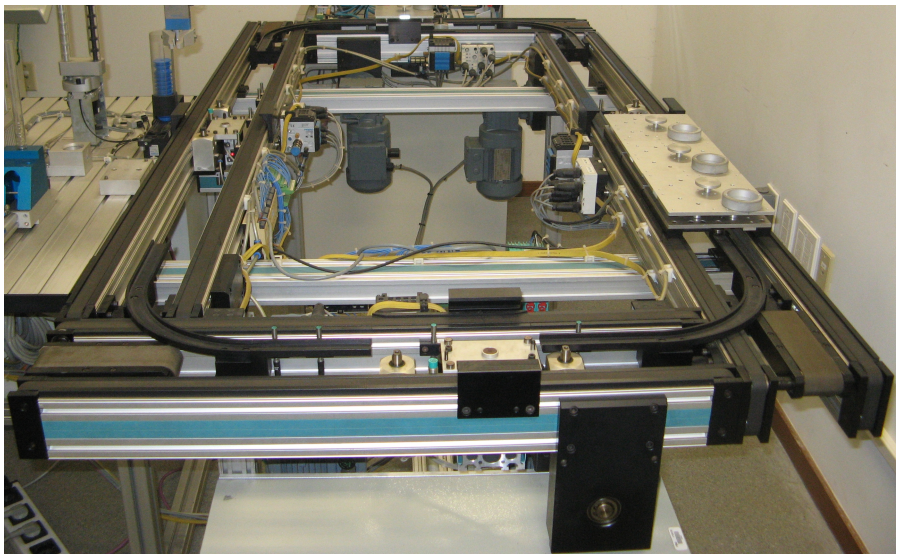


Figura 3.5: Foto do *"SP3-Transporte"*

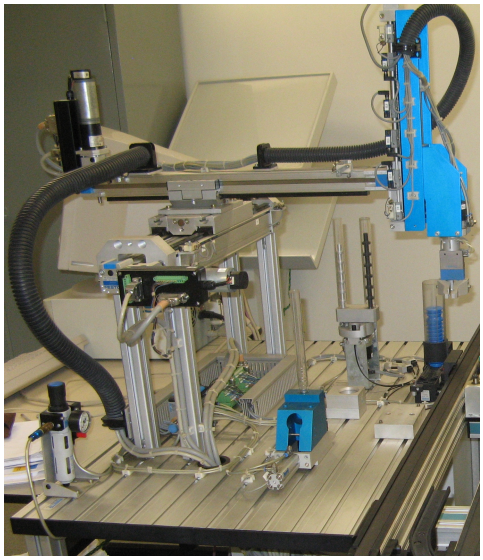


Figura 3.6: Foto do *"SP4-Montagem"*

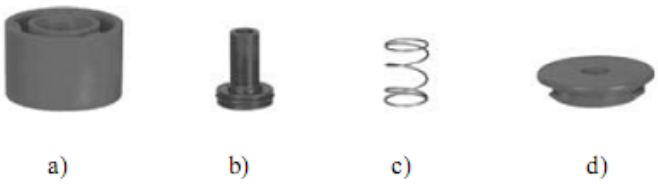


Figura 3.7: Peças que compõem o produto final: a) *"corpo"*; b) *"pino"*; c) *"mola"*; d) *"tampa"*

4 Usuários

Dentro do sistema produtivo (SP) estudado podem ser observados 3 tipos de usuários com características e responsabilidades específicas e estes devem se comunicar para negociar a produção. Estes usuários da arquitetura de controle são chamados de *"cliente"*, *"operador"* e *"sistema produtivo"*.

4.1 "Cliente"

O *"cliente"* é um tipo de usuário que possui uma solicitação de serviço a ser atendida, chamada de pedido, e é representado pela Figura 4.1.

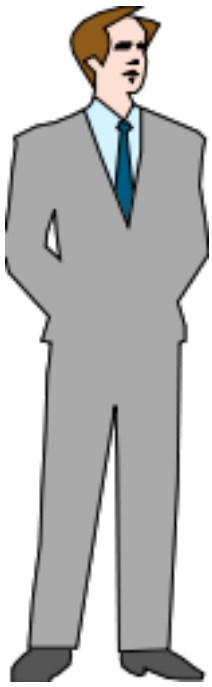


Figura 4.1: Representação de um *"cliente"*

O pedido do *"cliente"* é uma quantidade específica de produtos que deve ser fabricada utilizando os recursos humanos e materiais disponíveis no sistema como um todo. Na proposta,

assume-se que a responsabilidade de verificar a viabilidade de atendimento ao pedido é do *"cliente"*.

O *"cliente"* tem conhecimento das atribuições necessárias para o atendimento do seu pedido e por isso é capaz de procurar os usuários que possuam essas atribuições e que sejam capazes de atender o seu pedido. A busca das capacidades de usuários é feita em repositórios de dados distribuídos em que esses usuários estejam cadastrados.

Quando um pedido a ser atendido necessita de mais de uma etapa de atividades, a ordenação dessas etapas é de responsabilidade do *cliente*. A negociação do atendimento ao pedido inclui informações de prazo para produção, então, o *"cliente"* deve garantir que as etapas sigam a sequência necessária para que seu pedido seja atendido e ele utiliza os prazos para produção como ferramenta para garantir essa sequência.

Caso um *"cliente"* negocie as etapas em ordem errada ou com prazos incompatíveis de uma para outra, o mesmo assume os erros cometidos e os prejuízos destes erros, ou seja, em uma produção composta por 3 etapas $etapa_1$, $etapa_2$ e $etapa_3$ em que a ordem de execução das etapas segue $etapa_1 \rightarrow etapa_2 \rightarrow etapa_3$, caso o *"cliente"* negocie a execução das etapas segundo $etapa_1 \rightarrow etapa_3 \rightarrow etapa_2$ este receberá o pedido atendido da forma incorreta e terá que arcar com seus custos. Assim como, caso o *"cliente"* negocie a execução da $etapa_1$ com término previsto para um prazo superior ao prazo para o início da $etapa_2$ de forma que inviabilize a execução dessa dentro dos prazos negociados, o *"cliente"* deverá arcar com o prejuízo deste atraso mesmo que seu pedido não tenha sido completamente atendido.

4.2 "Operador"

O *"operador"* é um tipo de usuário que é capaz de operar um ou mais equipamentos para a produção de bens e é representado pela Figura 4.2.

O usuário *"cliente"* negocia com o usuário *"operador"* as condições de uma etapa do atendimento ao seu pedido. Na proposta desse trabalho, o *"operador"* deve teleoperar equipamentos utilizando os serviços disponíveis.

O usuário *"operador"* possui uma série de capacidades, chamadas de atribuições, que indicam seu conhecimento técnico sobre o funcionamento de um certo equipamento. Essas atribuições devem ser publicadas pelo *"operador"* em repositórios de dados para que *"clientes"* possam localizá-las e, com isso, negociar o atendimento ao seu pedido. O *"operador"* registra suas atribuições da forma que considerar mais conveniente ao seu trabalho e é o único

Figura 4.2: Representação de um *"operador"*

responsável pelas informações contidas nessas.

Ao receber um pedido de um *"cliente"*, o *"operador"* deve registrar o pedido recebido em sua estação de trabalho e deve montar e organizar uma fila de pedidos segundo os critérios que ele considera relevantes. Essa fila pode influenciar na ordem de atendimento a solicitações de pedidos e, com isso, os prazos negociados com os *"clientes"* devem levar essa fila em consideração.

Ao concluir a negociação com o *"cliente"* o *"operador"* deve garantir que, nos prazos negociados e com os custos negociados, a solicitação poderá ser atendida e caso a falha no atendimento seja por responsabilidade do *"operador"* esse se torna o responsável pelos prejuízos deste erro. Ou seja, caso o *"operador"* negocie com o *"cliente"* que atenderá sua solicitação em um certo tempo t a partir de um certo instante T_i e não cumpre esse tempo t, caso esse atraso gere prejuízos em etapas seguintes do atendimento do pedido do *"cliente"*, o *"operador"* deve corrigir o problema gerado por seu erro. Caso os custos de produção ultrapassem o combinado entre *"cliente"* e *"operador"*, o *"operador"* deverá garantir que para o *"cliente"* o custo final será o combinado.

4.3 "Sistema produtivo"

O *"sistema produtivo"* é um tipo de usuário que possui um ou mais equipamentos, recursos materiais e outras entidades físicas para a produção de bens e é representado pela Figura 4.3.

O usuário *"cliente"* negocia com o usuário *"sistema produtivo"* as condições de uma etapa do atendimento ao seu pedido. Na proposta desse trabalho, o *"sistema produtivo"* deve disponi-

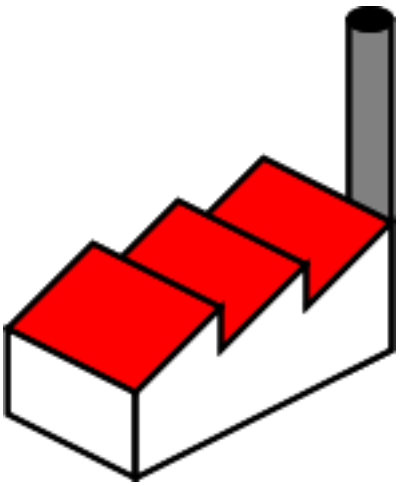


Figura 4.3: Representação de um *"sistema produtivo"*

bilizar equipamentos para serem teleoperados utilizando os serviços que o próprio *"sistema produtivo"* disponibiliza.

O usuário *"sistema produtivo"* possui uma série de capacidades, chamadas de atribuições, que indicam seus equipamentos, recursos materiais e outras entidades físicas. Essas atribuições devem ser publicadas pelo *"sistema produtivo"* em repositórios de dados para que *"clientes"* possam localizá-las e, com isso, negociar o atendimento ao seu pedido. O *"sistema produtivo"* registra suas atribuições da forma que considerar mais conveniente ao seu trabalho e é o único responsável pelas informações contidas nessas.

Os equipamentos de um *"sistema produtivo"* são capazes de executar funções específicas e essas funções devem ser publicadas pelo *"sistema produtivo"* na forma de serviço para que os *"operadores"* sejam capazes de teleoperar os equipamentos a fim de que estes executem as funções publicadas.

Ao receber um pedido de um *"cliente"*, o *"sistema produtivo"* deve registrar o pedido recebido em sua estação de trabalho e deve montar e organizar uma fila de pedidos segundo os critérios que ele considera relevantes. Essa fila pode influenciar na ordem de atendimento a solicitações de pedidos e, com isso, os prazos negociados com os *"clientes"* devem levar essa fila em consideração.

Ao concluir a negociação com o *"cliente"* o *"sistema produtivo"* deve garantir que, nos prazos negociados e com os custos negociados, a solicitação poderá ser atendida e caso a falha no atendimento seja por responsabilidade do *"sistema produtivo"* esse se torna o responsável pelos prejuízos deste erro. Ou seja, caso o *"sistema produtivo"* negocie com o *"cliente"* que atenderá sua solicitação em um certo tempo t a partir de um certo instante T_i e não cumpre esse tempo

t, caso esse atraso gere prejuízos em etapas seguintes do atendimento do pedido do *"cliente"*, o *"sistema produtivo"* deve corrigir o problema gerado por seu erro. Caso os custos de produção ultrapassem o combinado entre *"cliente"* e *"sistema produtivo"*, o *"sistema produtivo"* deverá garantir que para o *"cliente"* o custo final será o combinado.

4.4 Síntese do capítulo

O relacionamento entre os 3 usuários da arquitetura de controle depende das responsabilidades assumidas por cada um dos usuários e pelas funcionalidades que esses disponibilizam para a utilização pelos demais. Esse relacionamento é semelhante ao relacionamento entre os participantes da comunicação *peer-to-peer*.

Os *"clientes"* são usuários responsáveis por verificar a viabilidade do atendimento das etapas do pedido. Os *"operadores"* são usuários responsáveis por garantir que as condições negociadas com o *"cliente"* sejam cumpridas. Os *"sistemas produtivos"* são usuários responsáveis por garantir que as condições também negociadas com o *"cliente"* sejam cumpridas sob comando dos *"operadores"*.

Os *"operadores"* disponibilizam funcionalidades para os *"clientes"* que permitem a negociação entre esses usuários. Os *"sistemas produtivos"* disponibilizam funcionalidades para os *"clientes"* que permitem a negociação entre esses usuários. Os *"sistemas produtivos"* também disponibilizam funcionalidades para os *"operadores"* que permitem a teleoperação de seus equipamentos.

5 Modelagem a análise

Para projetar e implementar uma arquitetura de controle de sistemas produtivos (SPs) é fundamental desenvolver e analisar o modelo do sistema de controle desejado.

Baseado nos conceitos, técnicas e ferramentas anteriormente apresentados no texto e no estudo de caso considerado, este capítulo inicialmente apresenta a estruturação da arquitetura de controle de SP disperso utilizando o *production flow schema* (PFS) e o refinamento desta estrutura até o nível em que cada atividade representa uma funcionalidade.

A partir deste modelo em PFS, é feito um novo refinamento para obter modelos em RdP em que se detalha três diferentes situações que são consideradas fundamentais para a implementação prática de arquitetura de controle.

Por fim, é apresentado o resultado da análise dos modelos utilizando o *software* HPSim.

5.1 Representação em PFS

A negociação entre os usuários da arquitetura pode ser dividida em três atividades básicas em que cada uma é realizada por um dos usuários envolvidos (*"cliente"*, *"operador"* e *"sistema produtivo"*) e três distribuidores que representam os recursos que inicializam as atividades, como na Figura 5.1. As atividades mantém comunicação assíncrona, com fluxo secundário de informação, em que fluem os dados específicos de cada sub-atividade entre as atividades de cada usuário. As atividades são chamadas de [*Atividades do Operador*], com o conjunto de atividades realizadas pelo *"operador"*, [*Atividades do Cliente*], com o conjunto de atividades realizadas pelo *"cliente"*, e [*Atividades do SP*], com o conjunto de atividades realizadas pelo *"sistema produtivo"*, como na Figura 5.2.

Na Figura 5.2, a atividade do *"cliente"* comunica-se com a atividade do *"operador"* e com a atividade do *"sistema produtivo"*. Nessa comunicação fluem conjuntos de informações diferentes para cada sub-atividade. Para detalhar melhor essa comunicação a [*Atividades do Cliente*] pode ser dividida em 2 sub-atividades, uma que se comunica com a [*Atividades do Operador*]

Figura 5.1: Fluxo de atividades com três atividades básicas.

e outra que se comunica com a [*Atividades do SP*], como na Figura 5.3 (nesta figura foram removidos os distribuidores para melhorar a visualização da mesma).

Na Figura 5.3, a atividade do *"cliente"*[*Acordo com Operador*] comunica-se com a atividade do *"operador"* [*Atividades do Operador*] e a atividade do *"cliente"* [*Acordo com SP*] comunica-se com a atividade do *"sistema produtivo"* [*Atividades do SP*]. Novamente, nas comunicações assíncronas entre as atividades fluem conjuntos de informações diferentes para cada sub-atividade. Esses diferentes conjuntos de informações se devem a etapas diferentes da negociação. Um refinamento dessas etapas de negociação pode ser dividindo essas atividades em 3 sub-atividades, como na Figura 5.4.

A atividade [*Atividades do Operador*] é refinada dividindo-se em atividades mais básicas: uma para recebimento de pedidos (chamada de [*Recebe Pedido*]), uma de proposta (chamada de [*Envia Proposta*]) e uma de confirmação (chamada de *"Atende Pedido"*), conforme ilustrado na Figura 5.4.

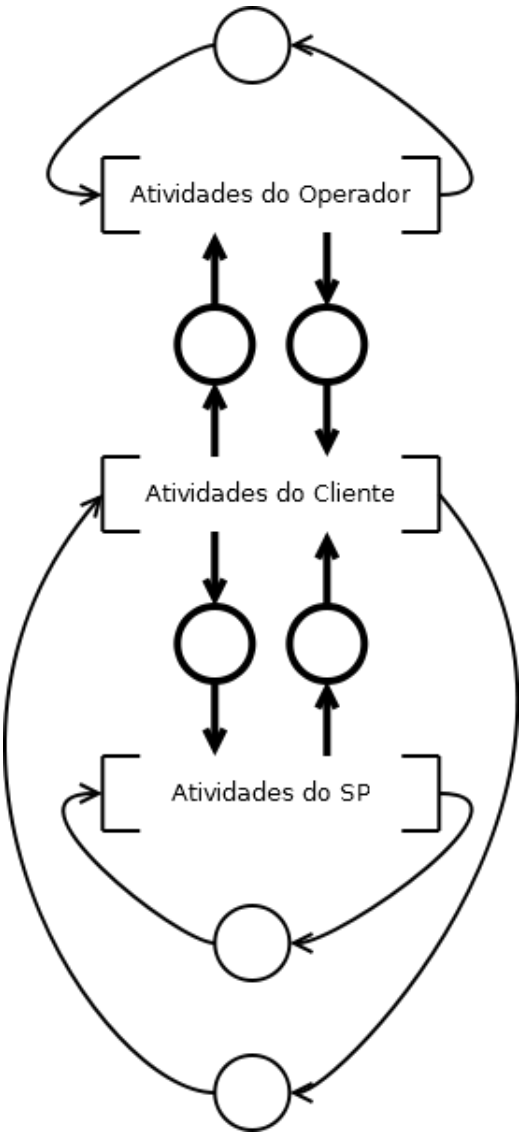


Figura 5.2: Fluxo de atividades com três atividades básicas e suas comunicações.

Da mesma forma a atividade [*Atividades do SP*] é refinada dividindo-se em atividades mais básicas: uma para recebimento de pedidos (chamada de [*Recebe Pedido*]), uma de proposta (chamada de [*Envia Proposta*]) e uma de confirmação (chamada de *"Atende Pedido"*), conforme ilustrado na Figura 5.4.

As atividades [*Acordo com Operador*] e [*Acordo com SP*] são refinadas dividindo-se em atividades mais básicas: uma de cada para envio de pedidos (chamadas de [*Solicita Operador*] e [*Solicita SP*], respectivamente), uma de cada para proposta (chamadas de [*Negocia com Operador*] e [*Negocia com SP*], respectivamente) e uma de cada para confirmação (chamadas de [*Responde ao Operador*] e [*Responde ao SP*], respectivamente), conforme ilustrado na Figura 5.4.

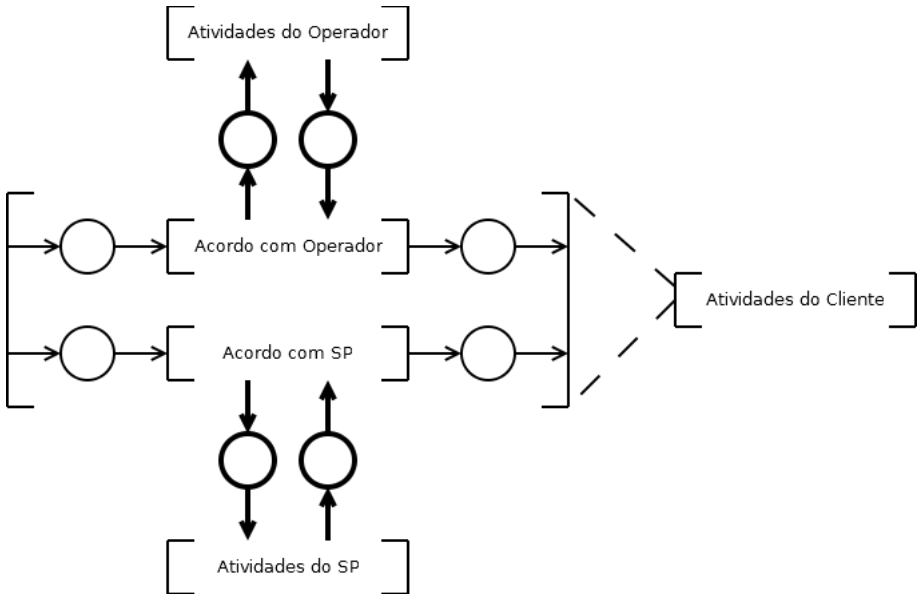


Figura 5.3: Primeiro refinamento feito nas *Atividades do Cliente*.

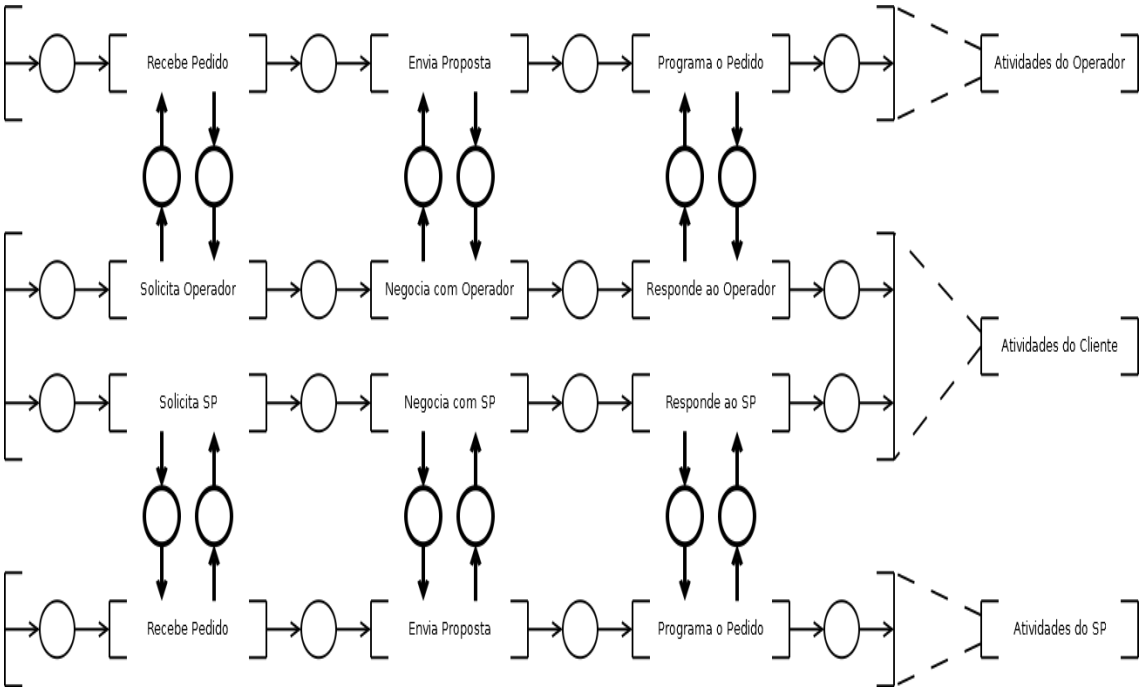


Figura 5.4: Refinamento das atividades para detalhamento das etapas da negociação.

Na Figura 5.4 é descrito também o fluxo secundário de informações entre essas atividades.

A seguir, refinam-se mais algumas dessas atividades. Na Figura 5.5 as atividades [*Recebe Pedido*], da [*Atividades do Operador*] e da [*Atividades do SP*], [*Solicita Operador*] e [*Solicita SP*] são refinadas.

Na Figura 5.5, as atividades [*Solicita Operador*] e [*Solicita SP*] (Fig. 5.4) são refinadas

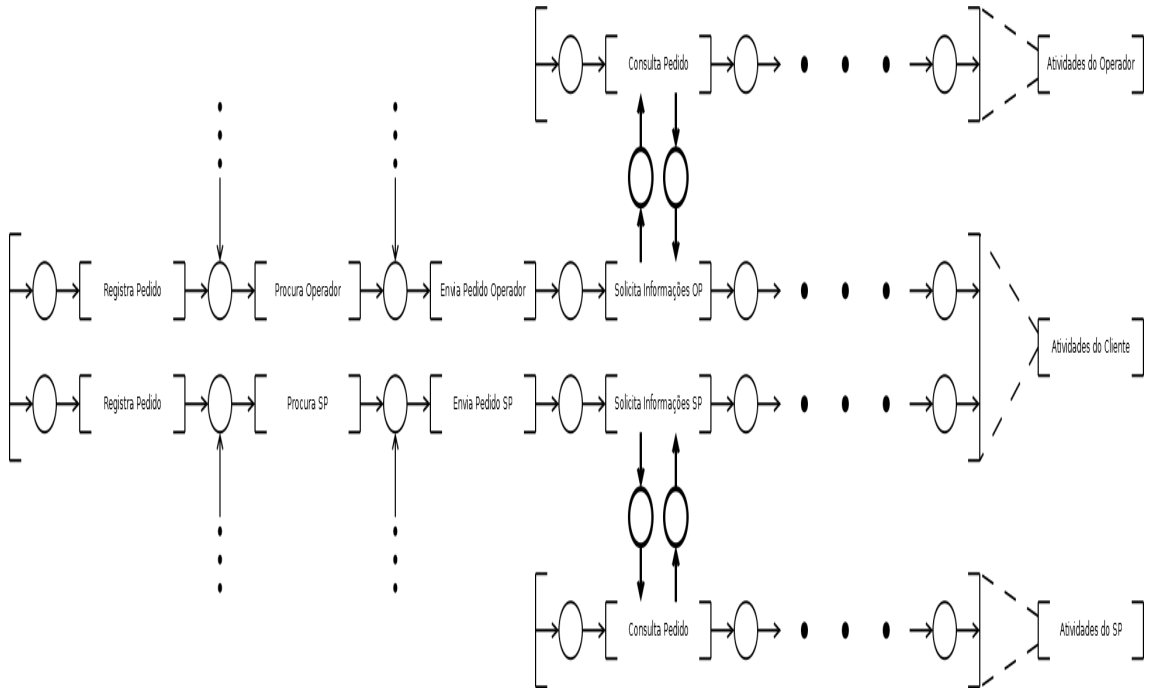


Figura 5.5: Refinamento das atividades que se referem ao envio e recebimento de pedidos.

em diferentes atividades: uma atividade, de cada uma das refinadas, é para publicação do pedido do *"cliente"* em repositórios de dados para futura consulta dos *"operadores"* e *"sistemas produtivos"* (ambas chamadas de [*Registra Pedido*]); uma atividade, de cada uma das refinadas, é para consulta em repositórios de dados por *"operadores"* e *"sistemas produtivos"* com serviços publicados (chamadas de [*Procura Operador*] e [*Procura SP*], respectivamente); outra atividade, de cada uma das refinadas, é para envio informações do repositório de dados em que está registrado o pedido do *"cliente"* para ser gravado nos bancos de dados dos *"operadores"* e dos *"sistemas produtivos"* (chamadas de [*Envia Pedido Operador*] e [*Envia Pedido SP*], respectivamente); a última atividade, de cada uma das refinadas, é para solicitação dos *"operadores"* e dos *"sistemas produtivos"* de vistas ao pedido registrado nos repositórios de dados (chamadas de [*Solicita Informações OP*] e [*Solicita Informações SP*], respectivamente). O distribuidor que está entre as atividades [*Registra Pedido*] e [*Procura Operador*] recebe um arco vindo do início de uma nova etapa de atendimento ao pedido, posteriormente representada, assim como o distribuidor que está entre as atividades [*Registra Pedido*] e [*Procura SP*]. O distribuidor que está entre as atividades [*Procura Operador*] e [*Envia Pedido Operador*] recebe um arco vindo de uma negociação concluída sem sucesso para a etapa de atendimento ao pedido, posteriormente representada, assim como o distribuidor que está entre as atividades [*Procura SP*] e [*Envia Pedido SP*].

Nessa mesma figura, as atividades [*Recebe Pedido*], da [*Atividades do Operador*] e da

[*Atividades do SP*] (Fig. 5.4), são alteradas para uma função de consulta aos repositórios de dados para acesso a leitura do pedido do *"cliente"* (chamadas igualmente de [*Consulta Pedido*]).

Pelo fluxo secundário de informação representado, as atividades [*Consulta Pedido*], da [*Atividades do Operador*] e da [*Atividades do SP*], são acionadas pelas atividades [*Solicita Informações OP*] e [*Solicita Informações SP*], respectivamente.

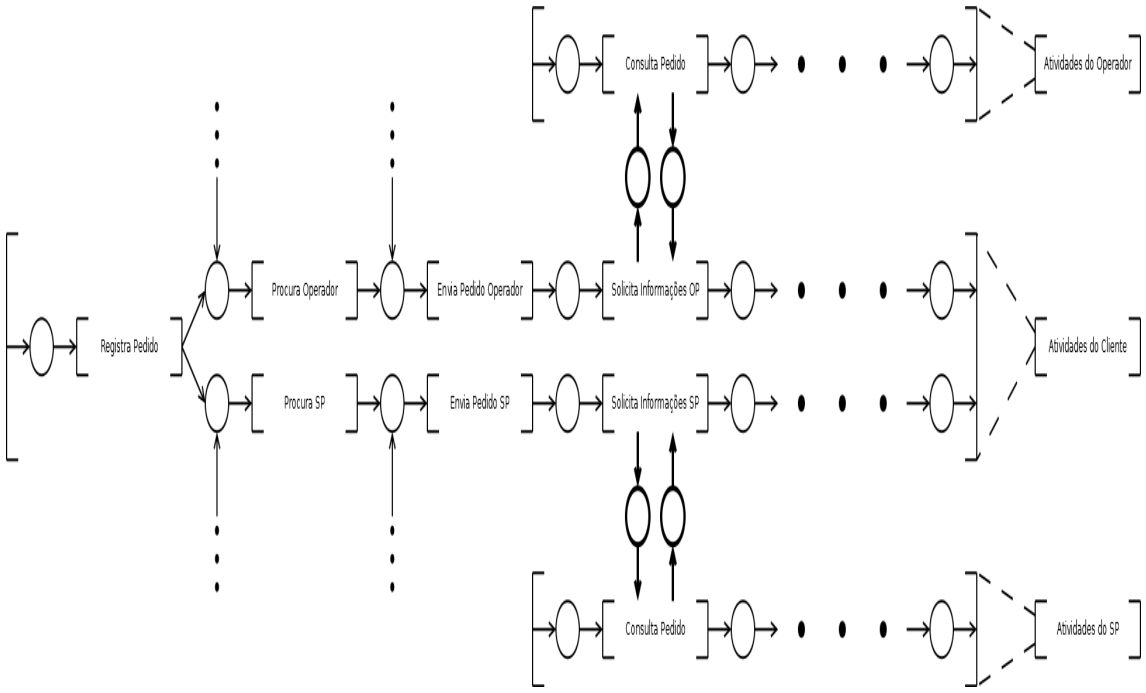


Figura 5.6: Agrupamento de atividades comuns de [*Atividades do Cliente*].

Duas atividades da Figura 5.5 possuem o mesmo nome dentro do mesmo bloco de atividades e possuem a mesma função. Por isso essas 2 atividades podem ser agrupadas em uma só. As atividades são as de [*Registra Pedido*] da parte de [*Atividades do Cliente*].

Na Figura 5.7 é feito o refinamento de outro conjunto de atividades da Figura 5.4.

As atividades [*Envia Proposta*], que fazem parte da [*Atividades do Operador*] e da [*Atividades do SP*] (Fig. 5.4), são refinadas por 2 atividades. Estas atividades apresentam uma relação de conflito que fornece ao *"cliente"* duas possibilidades de resposta, uma em que o *"operador"* ou *"sistema produtivo"* se dizem incapazes de atender ao pedido do mesmo (uma negociação concluída sem sucesso), sem necessariamente especificar os motivos, e outra em que enviam as especificações a negociar com o *"cliente"* (como preço, prazo para conclusão e outras informações que julgar relevante em seu serviço). Essas duas atividades são chamadas de [*Disponível*] e [*Indisponível*], como na Figura 5.7.

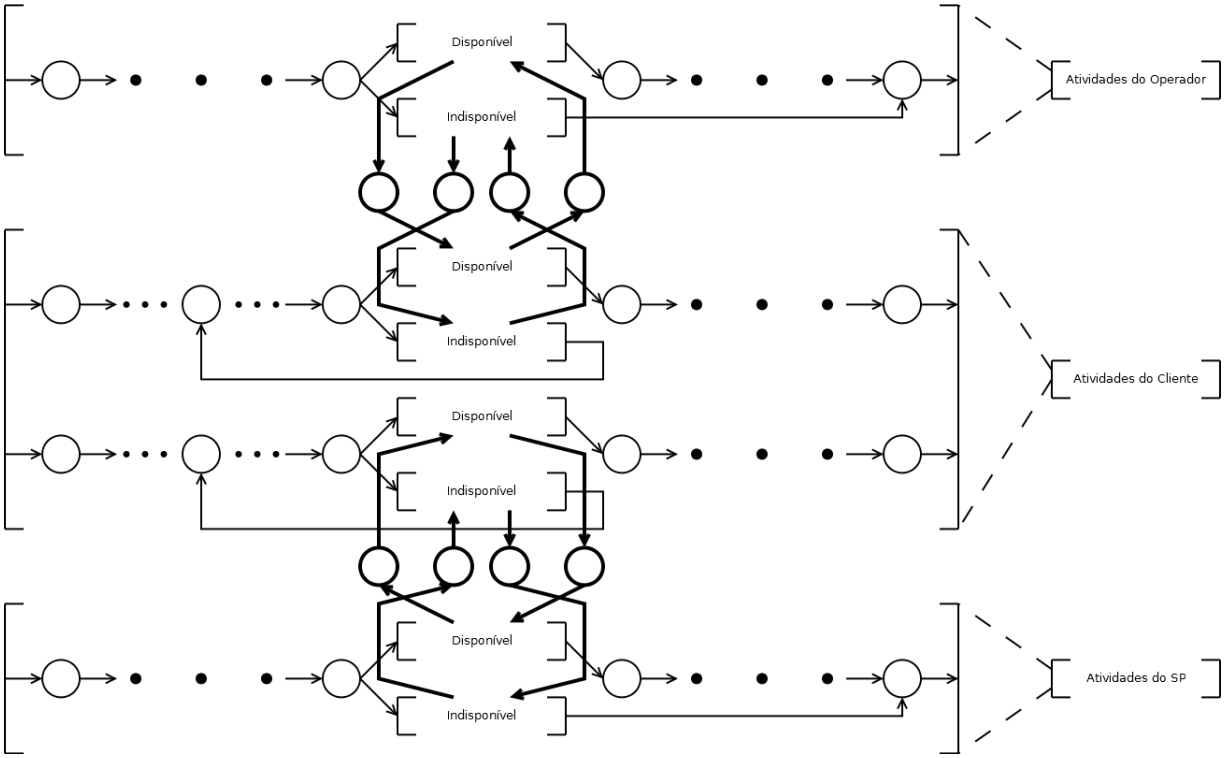


Figura 5.7: Refinamento das atividades que se referem a proposta.

De forma equivalente a anterior, as atividades [*Negocia com Operador*] e [*Negocia com SP*], de [*Atividades do Cliente*] (Fig. 5.4), são refinadas por duas atividades em conflito que retornam ao *"cliente"* a resposta do *"operador"* ou do *"sistema produtivo"*. As atividades também são chamadas de [*Disponível*] e [*Indisponível*] (Fig. 5.7).

Os fluxos secundários de informação estão entre atividades de nomes iguais, ou seja, as atividades [*Disponível*] e [*Indisponível*] de [*Atividades do Operador*] enviam informações para a atividade [*Disponível*] e [*Indisponível*] de [*Atividades do Cliente*], respectivamente. As atividades [*Disponível*] e [*Indisponível*] de [*Atividades do SP*] enviam informações para a atividade [*Disponível*] e [*Indisponível*] de [*Atividades do Cliente*], respectivamente (Fig. 5.7).

O último passo de refinamento pode ser visto na Figura 5.8, que trata da etapa de conclusão das negociações.

As atividades [*Atende ao Pedido*], de [*Atividades do Operador*] e de [*Atividades do SP*] (Fig. 5.4), são refinadas por 3 atividades. Estas atividades apresentam uma relação de conflito que fornece ao *"operador"* ou *"sistema produtivo"* duas possibilidades de resposta, uma em que o *"cliente"* se diz insatisfeito com a proposta (uma negociação concluída sem sucesso) e outra em que se diz satisfeito e envia informações necessárias para inserir informações de quem está atendendo ao pedido. Em sequência a essa segunda atividade, o *"operador"* ou

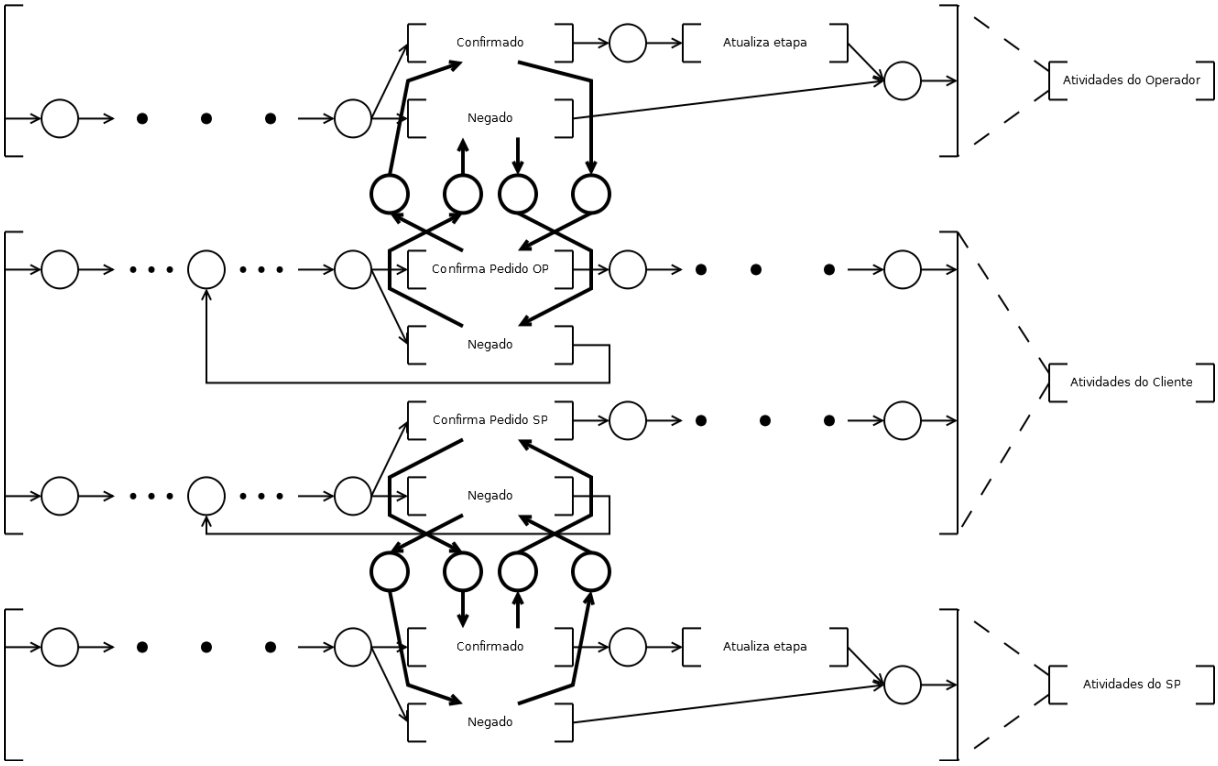


Figura 5.8: Refinamento das atividades que se referem a conclusão da negociação.

"sistema produtivo" fazem as alterações necessária nas informações do repositório de dados do pedido. Essas 3 atividades são chamadas de [*Negado*], [*Confirmado*] e [*Atualiza etapa*], respectivamente (Fig. 5.8).

De forma equivalente a anterior, as atividades [*Responde ao Operador*] e [*Responde ao SP*], de [*Atividades do Cliente*] (Fig. 5.4), são refinadas por 2 atividades que apresentam uma relação de conflito que retornam ao *"operador"* ou ao *"sistema produtivo"* a resposta do *"cliente"* sobre a proposta por ele recebida. As atividades também são chamadas de [*Negado*] e [*Confirmado*], respectivamente (Fig. 5.8).

Os fluxos secundários de informação estão entre atividades de nomes iguais, ou seja, as atividades [*Confirma Pedido OP*] e [*Negado*] de [*Atividades do Cliente*] enviam informações para a atividade [*Confirmado*] e [*Negado*] de [*Atividades do Operador*], respectivamente. As atividades [*Confirma Pedido SP*] e [*Negado*] de [*Atividades do Cliente*] enviam informações para a atividade [*Confirmado*] e [*Negado*] de [*Atividades do SP*], respectivamente (Fig. 5.8).

Por fim, os distribuidores que finalizam as [*Atividades do cliente*] podem ser refinados criando duas atividades com funções distintas. Uma atividade finaliza todas as negociações das etapas de atendimento ao pedido e outra inicia a negociação de uma nova etapa de atendimento ao pedido. Na Figura 5.9, a finalização das negociações é feita pela atividade

[*Fim da negociação*] e a inicialização de uma nova negociação é feita pela atividade [*Próxima etapa*].

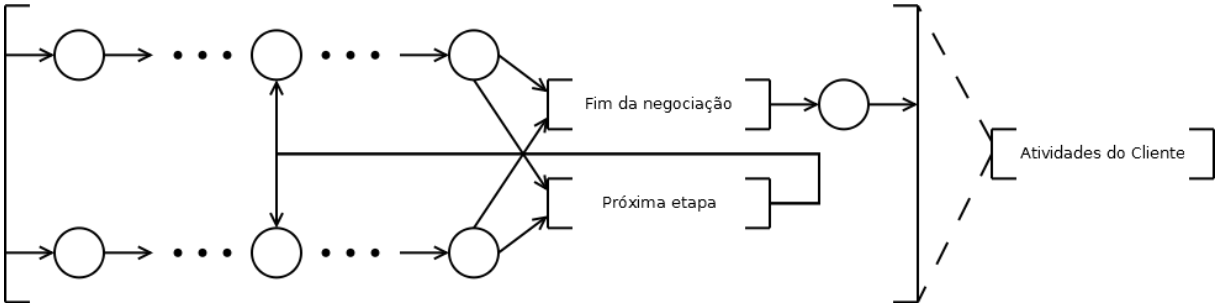


Figura 5.9: Refinamento dos distribuidores que finalizam as negociações ou iniciam uma nova negociação das etapas de atendimento ao pedido.

Os serviços podem ser definidos juntando as representações das Figuras 5.6, 5.7 e 5.8. Isolando as atividades que são refinadas da [*Atividades do Cliente*] na Figura 5.10.

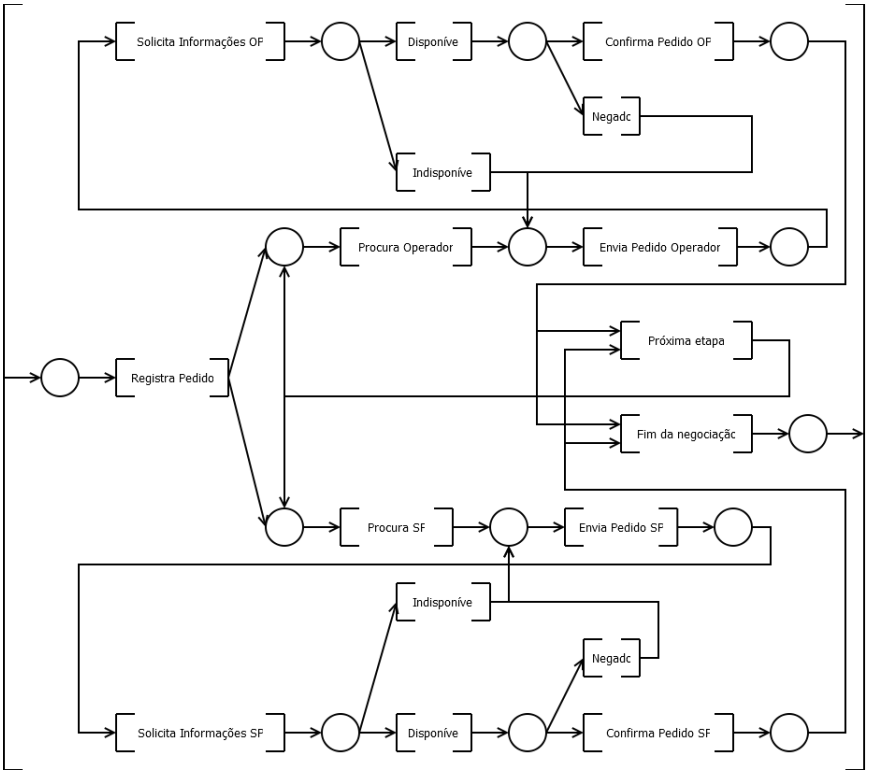


Figura 5.10: Modelo em PFS das [*Atividades do Cliente*]

Na representação de atividades da Figura 5.10, as atividades [*Envia Pedido Operador*], [*Solicita Informações OP*], *"Confirma Pedido"* e [*Monitora Operador*] podem ser agrupadas em um único serviço em que essas atividades são suas funcionalidades. Da mesma forma as

atividades [*Envia Pedido SP*], [*Solicita Informações SP*], *"Confirma Pedido"* e [*Monitora SP*] podem ser agrupadas em um único serviço em que essas atividades são suas funcionalidades.

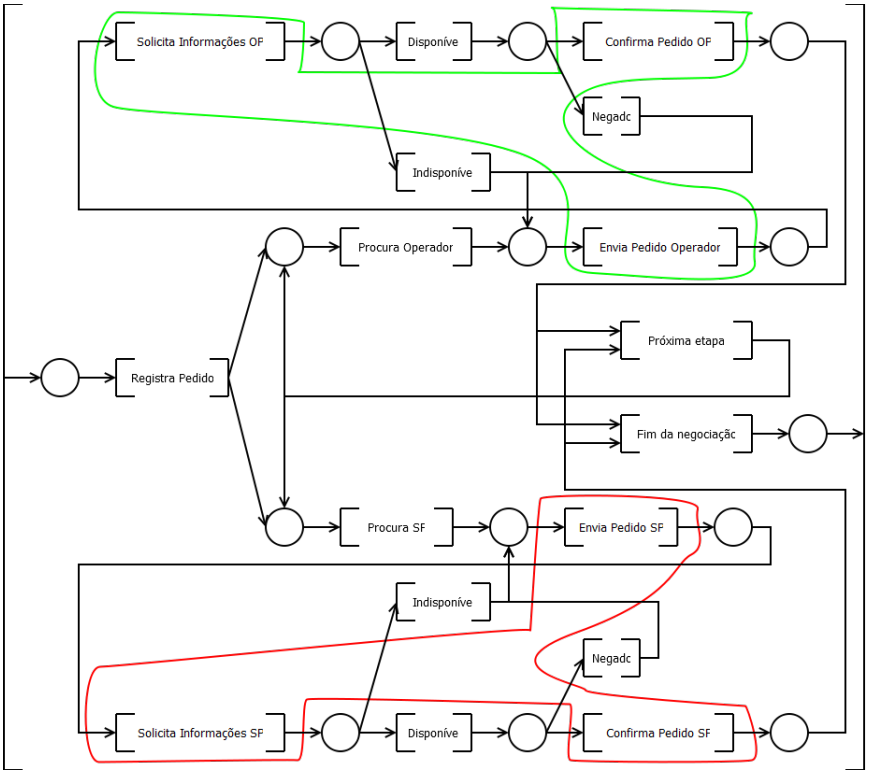


Figura 5.11: Modelo em PFS das [*Atividades do Cliente*] agrupadas em serviços

Na Figura 5.11 os serviços foram chamados de [*Solicita Operador*] (Fig 5.12) e [*Solicita SP*] (Fig 5.13).

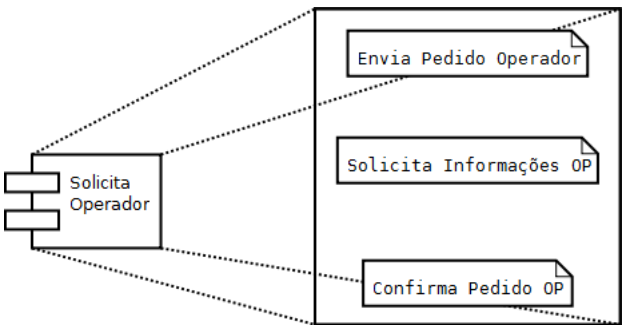


Figura 5.12: Atividades do serviço [*Solicita Operador*]

Os serviços podem ser definidos juntando as representações das Figuras 5.6, 5.7 e 5.8. Isolando as atividades que são refinadas da [*Atividades do Operador*] na Figura 5.14.

Da mesma forma, isolando as atividades que são refinadas da [*Atividades do SP*] na Figura

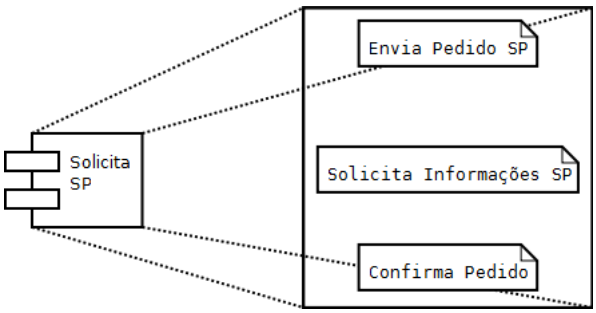


Figura 5.13: Atividades do serviço [*Solicita SP*]

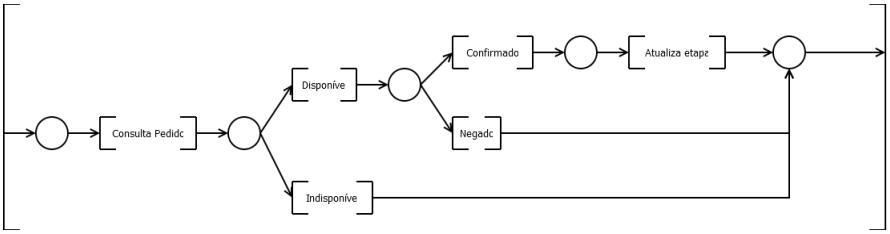


Figura 5.14: Modelo em PFS das [*Atividades do Operador*]

5.15.

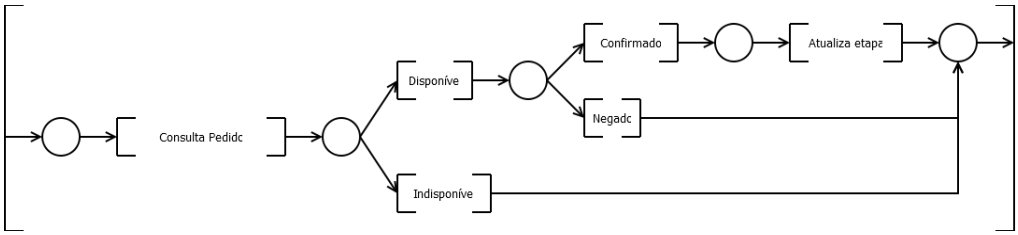


Figura 5.15: Modelo em PFS das [*Atividades do SP*]

A integração das partes representadas nas Figuras 5.11, 5.14, 5.15, pode ser vista na Figura 5.16

As atividades [*Consulta Pedido*] e [*Altera Pedido*] (Figuras 5.14 e 5.15) também podem ser agrupadas como um só serviço, chamado de *"Consulta/Altera Pedido"*, como na Figura 5.17.

De forma geral, a representação em serviços da arquitetura de controle pode ser vista na Figura 5.18 na região interna à curva de cor rosa.

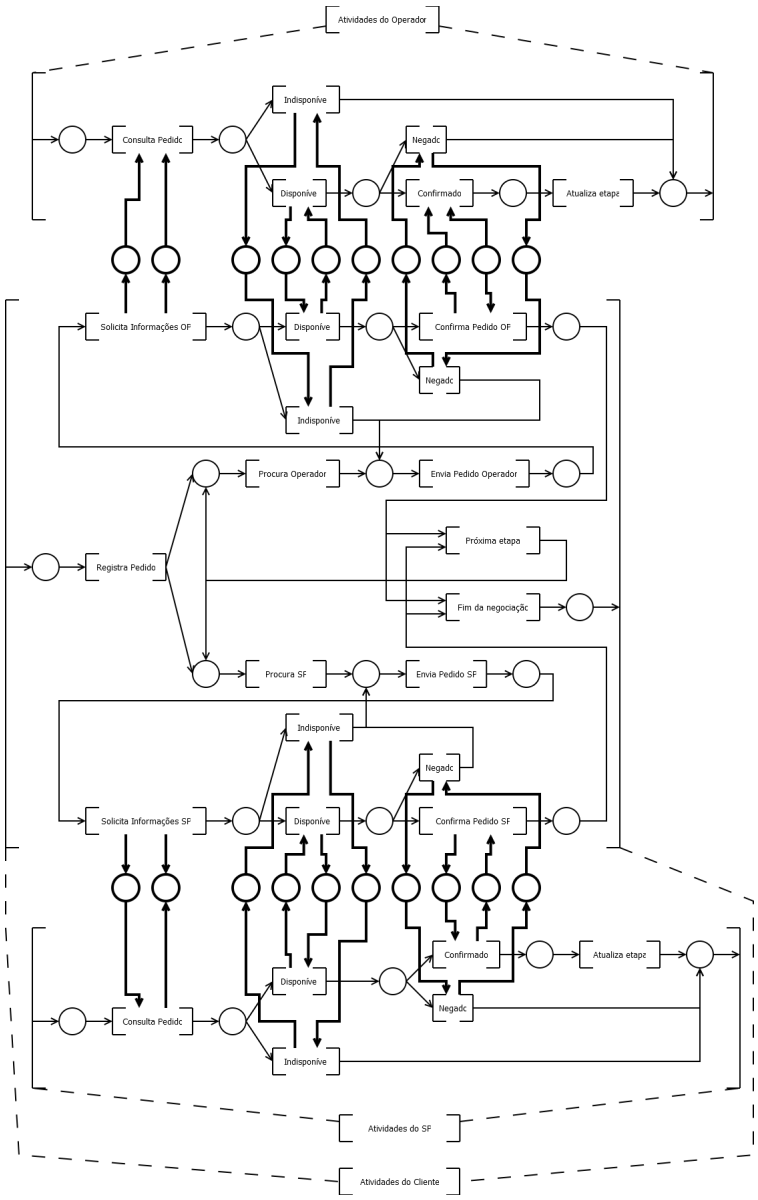


Figura 5.16: Modelo em PFS da integração das atividades dos usuários

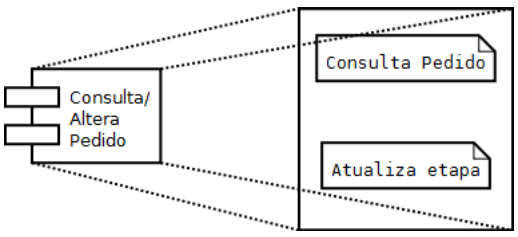


Figura 5.17: Atividades do serviço *"Consulta/Altera Pedido"*

5.2 Representação em RdP

Baseado nos modelos em PFS obtidos pela integração dos processos desenvolvidos pelos usuários do sistema, a RdP permite detalhar o comportamento dinâmico da arquitetura do

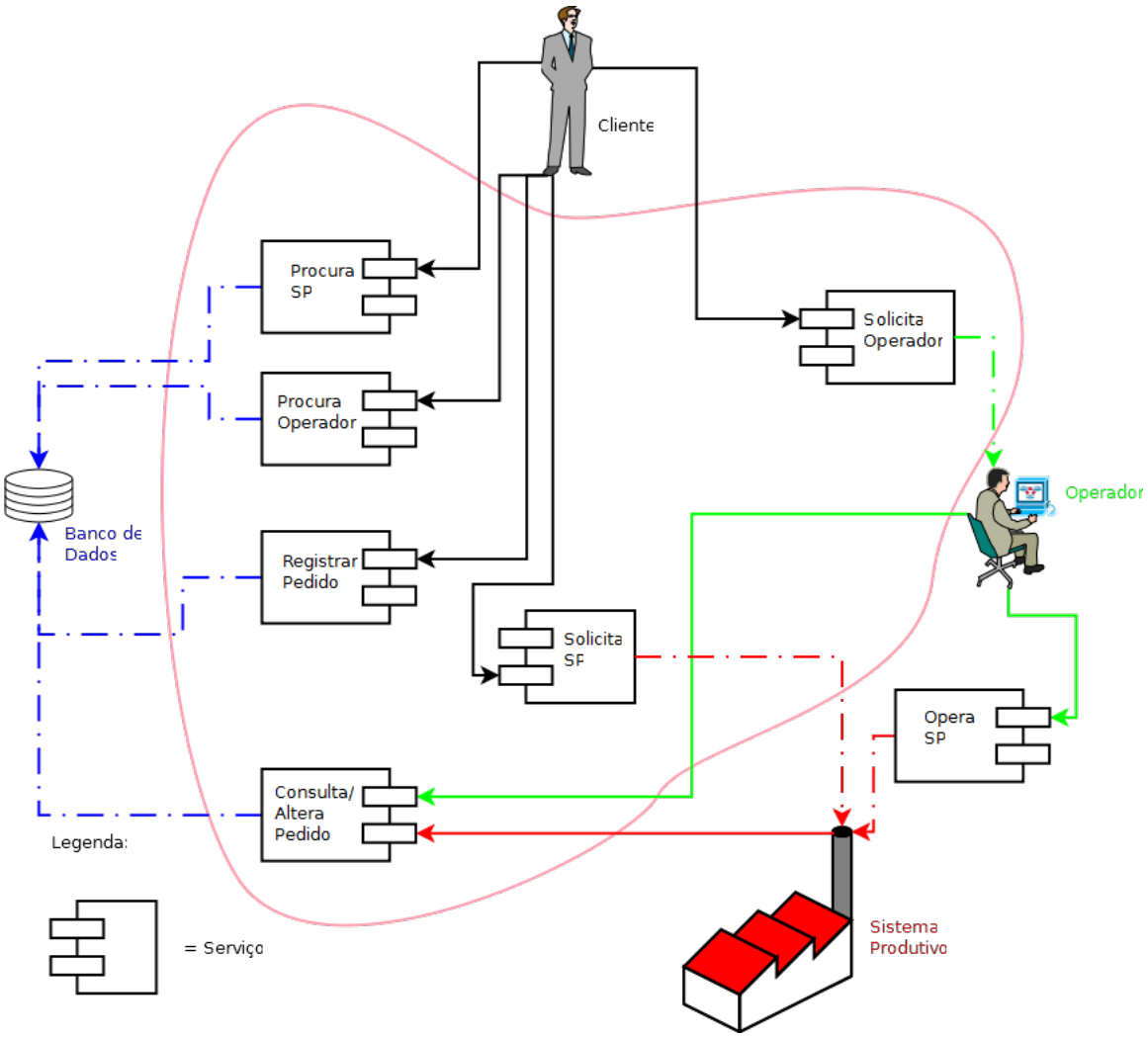


Figura 5.18: Representação em serviços da arquitetura de controle

sistema. Foram, assim, consideradas diversas situações, sendo que no presente texto destaca-se 3 casos fundamentais para avaliar a arquitetura de controle do SP disperso a ser implementado.

Em uma das situações o objetivo foi avaliar a negociação entre os usuários *"cliente"*/*"operador"* e *"cliente"*/*"sistema produtivo"* e a sincronização ao final da negociação. Para isso foram utilizados 1 *"cliente"*, 1 *"operador"* e 1 *"sistema produtivo"* (Figura 5.19).

Em outra das situações o objetivo foi avaliar a existência de conflito na escolha do *"sistema produtivo"* que irá atender o pedido do *"cliente"*. Para isso foram utilizados 1 *"cliente"* e 2 *"sistemas produtivos"* (Figura 5.20).

Por fim, uma das situações o objetivo foi avaliar o funcionamento geral da arquitetura com conflitos de escolha de *"operadores"* e de *"sistemas produtivos"* na negociação do atendimento ao pedido. Para isso foram utilizados 1 *"cliente"*, 4 *"operadores"* e 4 *"sistemas produtivos"* (na

Figura 5.21 os colchetes das atividades foram removidos para facilitar a visualização).

Para a análise do comportamento dinâmico de RdP deve-se estabelecer sua marcação inicial. No presente caso, considera-se uma marcação inicial que representa uma situação em que todos os usuários do sistema estão inicialmente disponíveis, Figura 5.21.

Segundo a proposta de Han et al. (2008) os WS podem ser modelados como uma transição da RdP, mas em algumas atividades o detalhamento pode variar. As atividades que possuem apenas um recurso de entrada e um recurso de saída são detalhadas como na Figura 5.22. As atividades que possuem apenas um recurso de entrada e dois recursos em paralelo de saída são detalhadas como na Figura 5.23. As atividades que possuem apenas um recurso de entrada e um recurso de saída, mas possuem fluxo secundário de informação, são detalhadas como na Figura 5.24 ou na Figura 5.25, i.e., a atividade é dividida em 2 partes, uma representando o evento de início da atividade e outra representando o evento de finalização da atividade. A estes eventos podem ser associados os arcos que representam o fluxo secundário.

As atividades do modelo da Figura 5.10 são convertidos em RdP conforme está na Tabela 5.1.

Para os modelos de atividades da Figura 5.14 e da Figura 5.15 a conversão segue o mesmo padrão da Tabela 5.1.

5.3 Análise dos modelos

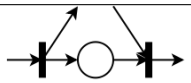
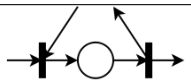
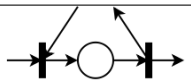
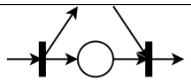
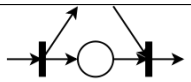
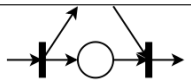
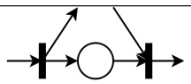
O modelo em Rede de Petri (RdP) da arquitetura de controle pode ser analisado utilizando uma ferramenta computacional. A ferramenta utilizada nessa etapa foi o HPSim, que é um *software* de edição e simulação para RdP.

A análise do modelo da Figura 5.19 (modelo com um *"cliente"*, um *"operador"* e um *"sistema produtivo"*), segundo as propriedades descritas na Seção 2.2, indica que este: (a) alcançou os estados desejados para o modelo; (b) o modelo, em RdP, é 1-limitado e por isso pode ser considerado seguro; (c) devido ao fluxo secundário de informações, o modelo, em RdP, não é conservativo; (d) o modelo, em RdP, não apresentou situações de *deadlock*; e (e) o modelo, em RdP, é reversível.

Com isso, conclui-se que o processo de negociação entre os usuários da do SP disperso deve ser implementado de acordo com o modelo da Figura 5.19.

A análise do modelo da Figura 5.20 (modelo com um *"cliente"* e dois *sistemas produtivos*), considerando um *operador*, segundo as propriedades descritas na Seção 2.2, indica que este:

Tabela 5.1: Modelo em RdP das atividades da Figura 5.11.

Atividades	**Modelo em RdP**
Registra Pedido	
Procura Operador	
Solicita Operador - Envia Pedido Operador	
Solicita Operador - Solicita Informações OP	
Disponível	
Indisponível	
Solicita Operador - Confirma Pedido	
Negado	
Procura SP	
Solicita SP - Envia Pedido SP	
Solicita SP - Solicita Informações SP	
Solicita SP - Confirma Pedido	

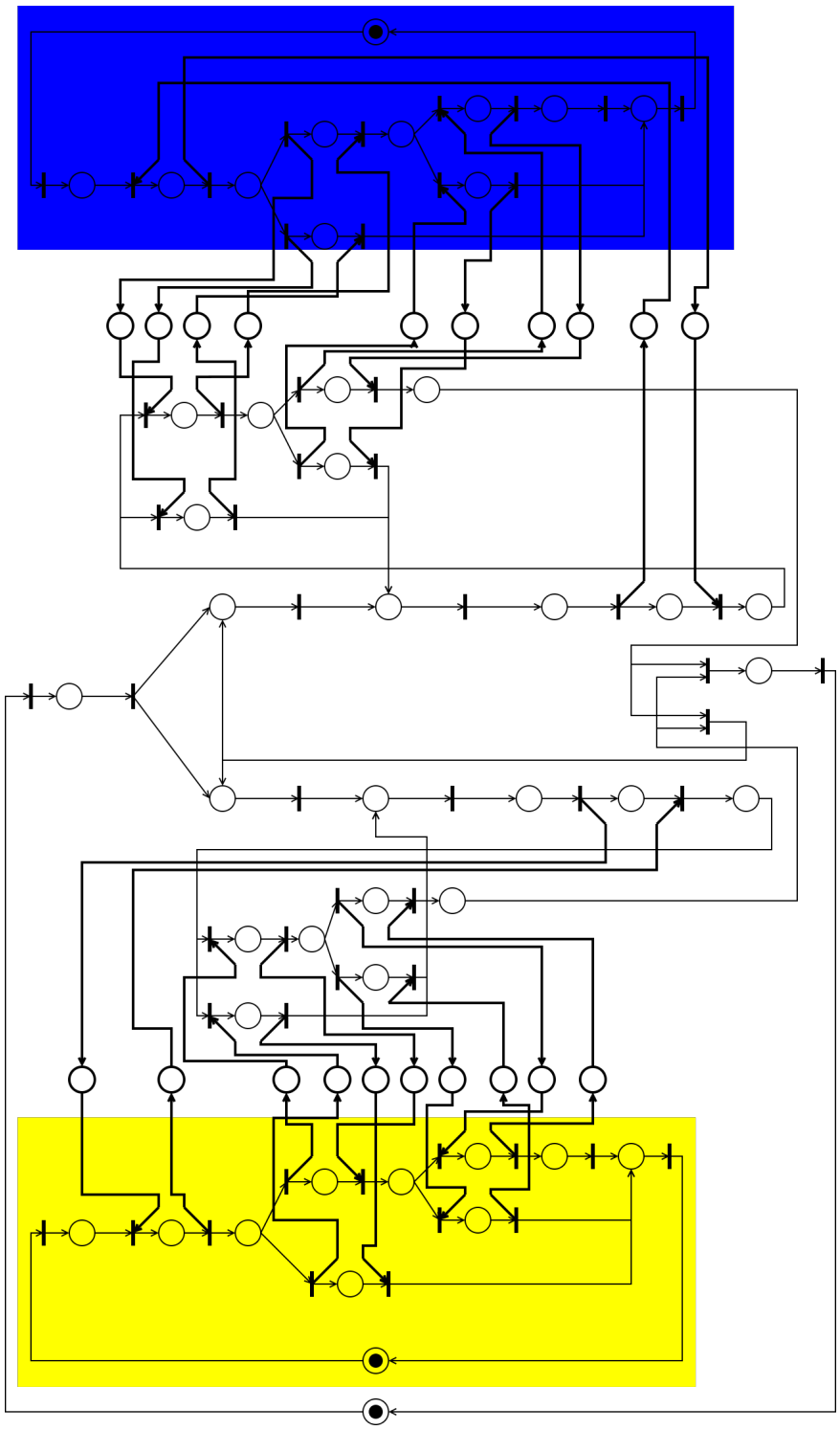


Figura 5.19: Representação da arquitetura em RdP para 1 *"cliente"*, 1 *"operador"* e 1 *"sistema produtivo"*

(a) alcançou os estados desejados para o modelo; (b) o modelo, em RdP, é 1-limitado e por isso pode ser considerado seguro; (c) devido ao fluxo secundário de informações, o modelo, em RdP, não é conservativo; (d) o modelo, em RdP, não apresentou situações de *deadlock*; e

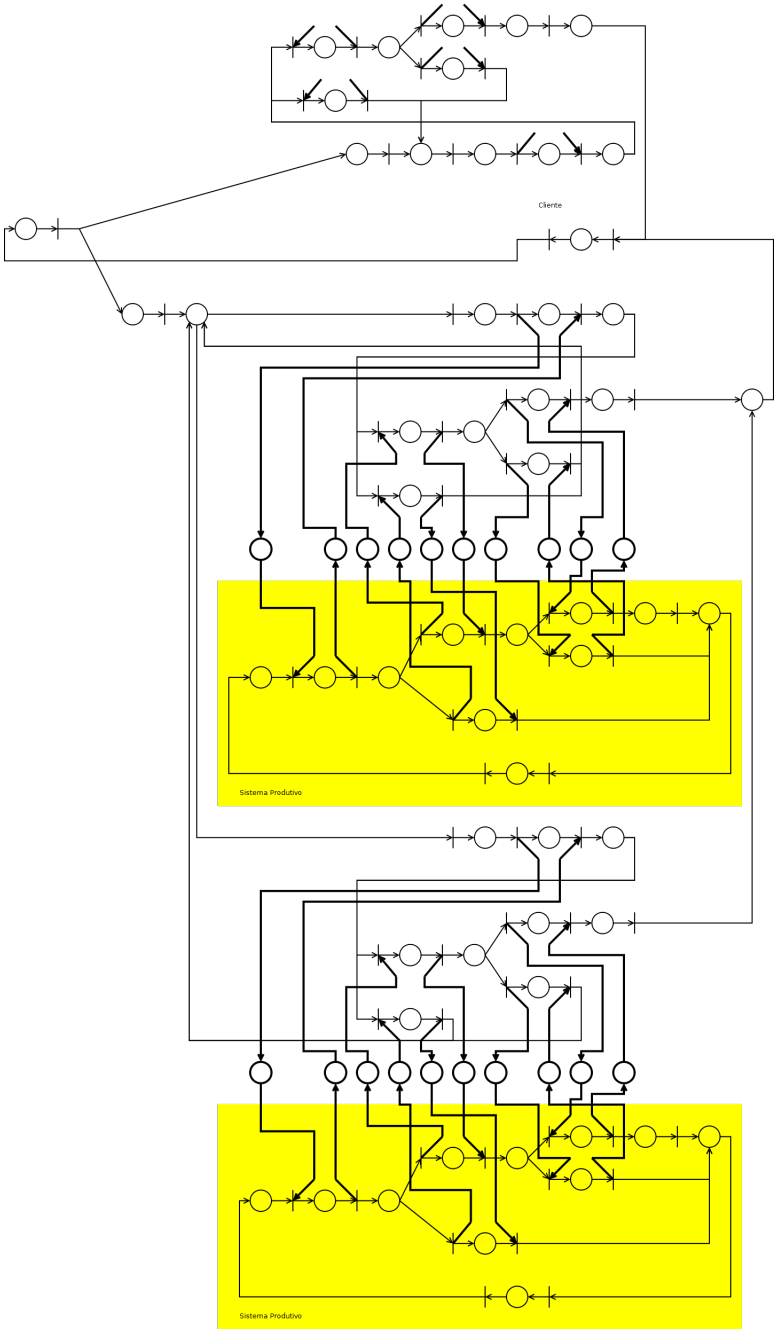


Figura 5.20: Representação da arquitetura em RdP para 1 *"cliente"* e 2 *"sistemas produtivos"*

(e) o modelo, em RdP, é reversível.

Com isso, conclui-se que o conflito na escolha do *"sistema produtivo"* que irá atender a etapa do pedido não gera estados indesejados pois o modelo especifica uma interface entre *"cliente"* e *"sistema produtivo"* para cada par *"cliente"*/*"sistema produtivo"*.

Por fim, a análise do modelo da Figura 5.21 (modelo com um *"cliente"*, quatro *"operadores"* e quatro *sistemas produtivos*), segundo as propriedades descritas na Seção 2.2, indica que este:

(a) alcançou os estados desejados para o modelo; (b) o modelo, em RdP, é 1-limitado e por isso pode ser considerado seguro; (c) devido ao fluxo secundário de informações, o modelo, em RdP, não é conservativo; (d) o modelo, em RdP, não apresentou situações de *deadlock*; e (e) o modelo, em RdP, é reversível.

Com isso, conclui-se que, de modo geral, a arquitetura de controle para SP disperso escolhida deve ser implementada de acordo com o modelo da Figura 5.21.

5.4 Síntese do capítulo

O procedimento de modelagem do sistema que usa inicialmente o PFS facilita a implementação das funcionalidades e a ordenação em que essas devem ser executadas. A análise do modelo refinado do modelo em PFS, que é um modelo em RdP, permite uma avaliação precisa das propriedades do sistema, baseada em um formalismo matemático.

Pela análise feita a partir dos modelos em RdP, caso a arquitetura de controle seja implementada seguindo os modelos, o sistema certamente apresentará apenas os estados dentro do universo especificado, isto é, o sistema nunca atinge nenhum estado indesejado ou imprevisto.

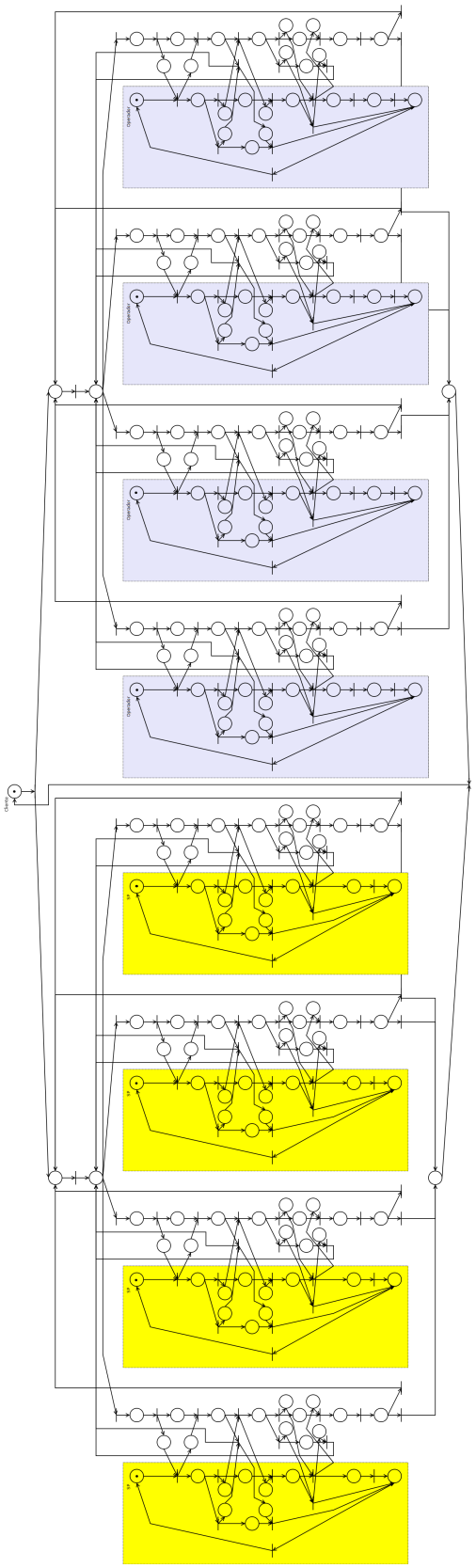


Figura 5.21: Representação da arquitetura em RdP para 1 *"cliente"*, 4 *"operadores"* e 4 *"sistemas produtivos"*

Figura 5.22: Modelo, em RdP, de uma transição com uma entrada e uma saída

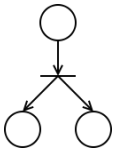


Figura 5.23: Modelo, em RdP, de uma transição com uma entrada e duas saídas

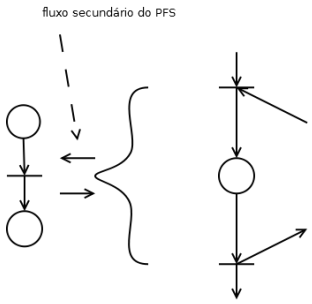


Figura 5.24: Modelo, em RdP, com uma atividade/transição que é decomposta em de duas transições a primeira com duas entradas e uma saída e a segunda com uma entrada e duas saídas

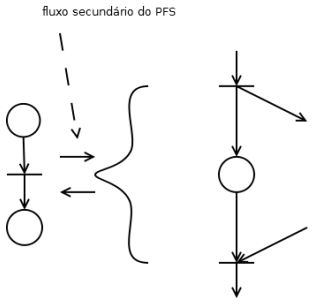


Figura 5.25: Modelo, em RdP, com uma atividade/transição que é decomposta em duas transições a primeira com uma entrada e duas saídas e a segunda com duas entradas e uma saída

6 Implementação e teste

Neste capítulo descreve-se a implementação da arquitetura de controle e os resultados dos testes realizados no sistema que emula um sistema produtivo (SP) disperso. Este capítulo considera o caso de pedido feito pelo *"cliente"* como um conjunto de etapas produtivas em que cada uma deve ser atendida por um *"operador"* e um *"sistema produtivo"*.

Baseado nos modelos obtidos no capítulo anterior, a primeira seção apresenta a estrutura das tabelas de dados que são utilizados em bancos de dados distribuídos e o relacionamento entre diferentes tabelas.

Em seguida, é mostrada a implementação dos WSs com descrição dos métodos (funcionalidades ou funções de apoio) e os exemplos de implementação de cada um dos métodos descritos.

Ao final são comentados os testes da aplicação dos WSs e tabelas de dados ao sistema que emula um SP disperso.

6.1 Repositórios de dados

Conforme os modelos desenvolvidos no capítulo anterior na arquitetura de controle considerada neste trabalho, foram definidas 4 tabelas de dados para os repositórios de dados. Essas 4 tabelas fazem parte de bancos de dados distribuídos que utilizam técnicas específicas de serviços para realizar suas consultas.

As tabelas são chamadas de *"pcs"*, *"ops"*, *"sps"* e *"eps"*. Cada uma dessas tabelas pode ter replicação em outro banco de dados e foram concebidas para assegurar a característica de bancos de dados distribuídos.

A tabela denominada *"pcs"* (Fig. 6.1) contém as informações dos pedidos dos *"clientes"*. Seu acesso é protegido para restringir os usuários que possuem acesso ao seu conteúdo.

Na Figura 6.1, os atributos indicados são *"num"*, *"nome"*, *"chaveL"*, *"chaveG"* e *"informa-*

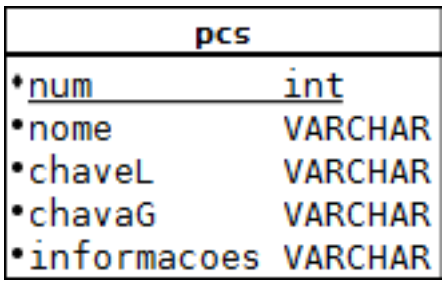


Figura 6.1: Representação da tabela de pedidos (*"pcs"*)

coes". O atributo *"num"* é a chave primária da tabela, usado para identificação nos relacionamentos entre tabelas. Este atributo é um inteiro e é fornecido automaticamente pelo banco de dados. O atributo *"nome"* é o nome do *"cliente"* que registrou o pedido. Este atributo é um conjunto de caracteres e é usado para identificar o pedido pelos *"operadores"* e *"sistemas produtivos"*. Este atributo é único na tabela e não pode ser nulo. O atributo *"chaveL"* é uma senha usada para permitir que usuários específicos (*"operador"* ou *"sistema produtivo"* que tenha negociado com o *"cliente"*) possam ler as *"informacoes"* do pedido. Este atributo é um conjunto de caracteres e junto com o atributo *"nome"* é usado para liberar o acesso aos dados. Este atributo não pode ser nulo. O atributo *"chaveG"* é uma senha usada para permitir que usuários específicos (*"operador"* ou *"sistema produtivo"* que tenha negociado com o *"cliente"*) possam ler e gravar informações no pedido do *"cliente"*. Este atributo é um conjunto de caracteres e junto com o atributo *"nome"* é usado para liberar acesso aos dados. Este atributo não pode ser nulo. O atributo *"informacoes"* é uma estrutura formatada de informações que indica as características do pedido do *"cliente"*. Este atributo é um conjunto de caracteres que deve ser mantido em sigilo para todos usuários que não possuem os atributos *"nome"* e *"chaveL"*. Este atributo não pode ser nulo.

A tabela denominada *"ops"* (Fig. 6.2) contém as informações das atribuições dos *"operadores"*. Seu acesso é público para que os *"clientes"* possam identificar os *"operadores"* que melhor atendem sua requisição.

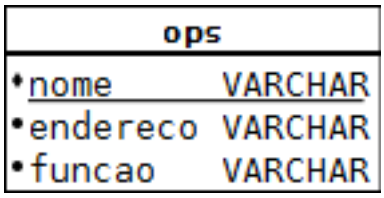


Figura 6.2: Representação da tabela de atribuições de *"operadores"* (*"ops"*)

Na Figura 6.2, os atributos indicados são *"nome"*, *"endereco"* e *"funcao"*. O atributo *"nome"*

é o nome do *"operador"* que está cadastrado no banco de dados. Este atributo é um conjunto de caracteres e é usado para identificar os *"operadores"*. Este atributo é único na tabela e não pode ser nulo, além disso, é a chave primária da tabela. O atributo *"endereco"* é o endereço virtual em que os *"clientes"* podem comunicar-se com o *"operador"*. Este atributo é um conjunto de caracteres e não pode ser nulo. O atributo *"funcao"* é uma estrutura formatada de informações que indica as atribuições que possui o *"operador"*. Este atributo é um conjunto de caracteres e sua estrutura é definida pelo banco de dados. Este atributo não pode ser nulo.

A tabela denominada *"sps"* (Fig. 6.3) contém as informações dos equipamentos, recursos materiais e outras entidades físicas que possuem os *"sistemas produtivos"*. Seu acesso é público para que os *"clientes"* possam identificar os *"sistemas produtivos"* que melhor atendem sua requisição.

Figura 6.3: Representação da tabela de atribuições de *"sistemas produtivos"* (*"sps"*)

Na Figura 6.3, os atributos indicados são *"nome"*, *"endereco"*, *"equipamentos"*, *"recursos"* e *"entidades"*. O atributo *"nome"* é o nome do *"sistema produtivo"* que está cadastrado no banco de dados. Este atributo é um conjunto de caracteres e é usado para identificar os *"sistemas produtivos"*. Este atributo é único na tabela e não pode ser nulo, além disso, é a chave primária da tabela. O atributo *"endereco"* é o endereço virtual em que os *"clientes"* podem comunicar-se com o *"sistema produtivo"*. Este atributo é um conjunto de caracteres e não pode ser nulo. O atributo *"equipamentos"* é uma estrutura formatada de informações que indica as máquinas e outros equipamentos que possui o *"sistema produtivo"*. Este atributo é um conjunto de caracteres e sua estrutura é definida pelo banco de dados. Este atributo não pode ser nulo. O atributo *"recursos"* é uma estrutura formatada de informações que indica os recursos materiais que possui o *"sistema produtivo"*. Este atributo é um conjunto de caracteres e sua estrutura é definida pelo banco de dados. Este atributo não pode ser nulo. O atributo *"entidades"* é uma estrutura formatada de informações que indica as outras entidades físicas que possui o *"sistema produtivo"*. Este atributo é um conjunto de caracteres e sua estrutura é definida pelo banco de dados. Este atributo não pode ser nulo.

A tabela denominada *"eps"* (Fig. 6.4) contém as informações dos *"operadores"* e *"sistemas produtivos"* que são responsáveis pela execução de cada etapa do atendimento ao pedido. Seu acesso é restrito e protegido pelo atributo *"chaveG"* da tabela *"pcs"*.

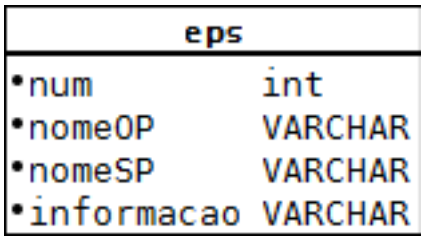


Figura 6.4: Representação da tabela de etapas de atendimento a pedidos (*"eps"*)

Na Figura 6.4, os atributos indicados são *"num"*, *"nomeOP"*, *"nomeSP"* e *"informacao"*. O atributo *"num"* é a chave primária da tabela, usado para identificação nos relacionamentos entre tabelas. Este atributo é um inteiro não nulo. O atributo *"nomeOP"* é o nome do *"operador"* que executará a etapa de atendimento ao pedido por teleoperação. Este atributo é um conjunto de caracteres e é usado para identificar o *"operador"*. Este atributo não pode ser nulo. O atributo *"nomeSP"* é o nome do *"sistema produtivo"* que fornecerá seus recursos para teleoperação da etapa de atendimento ao pedido. Este atributo é um conjunto de caracteres e é usado para identificar o *"sistema produtivo"*. Este atributo não pode ser nulo. O atributo *"informacao"* é uma estrutura formatada de informações que descreve completamente a etapa executada pelo *"operador"* e pelo *"sistema produtivo"*. Este atributo é um conjunto de caracteres e sua estrutura é definida pelo banco de dados. Este atributo não pode ser nulo.

As relações entre as tabelas *"pcs"*, *"ops"*, *"sps"* e *"eps"* são definidas na Figura 6.5.

O atributo *"num"* na tabela *"pcs"* é único, ou seja, cada elemento possui esse atributo diferente um do outro. Na tabela *"eps"* o atributo *"num"* não é único e possui relação com o de mesmo nome na tabela *"pcs"*. Essa relação é descrita como de $1 \rightarrow n$, na Figura 6.5, pois um pedido pode possuir mais de uma etapa.

O atributo *"nome"* na tabela *"ops"* é único, ou seja, cada elemento possui esse atributo diferente um do outro. Na tabela *"eps"* o atributo *"nomeOP"* não é único e possui relação com o atributo *"nome"* da tabela *"ops"*. Essa relação é descrita como de $1 \rightarrow n$, na Figura 6.5, pois um *"operador"* pode executar mais de uma etapa em vários pedidos.

O atributo *"nome"* na tabela *"sps"* é único, ou seja, cada elemento possui esse atributo diferente um do outro. Na tabela *"eps"* o atributo *"nomeSP"* não é único e possui relação com o atributo *"nome"* da tabela *"sps"*. Essa relação é descrita como de $1 \rightarrow n$, na Figura 6.5, pois um *"sistema produtivo"* pode executar mais de uma etapa em vários pedidos.

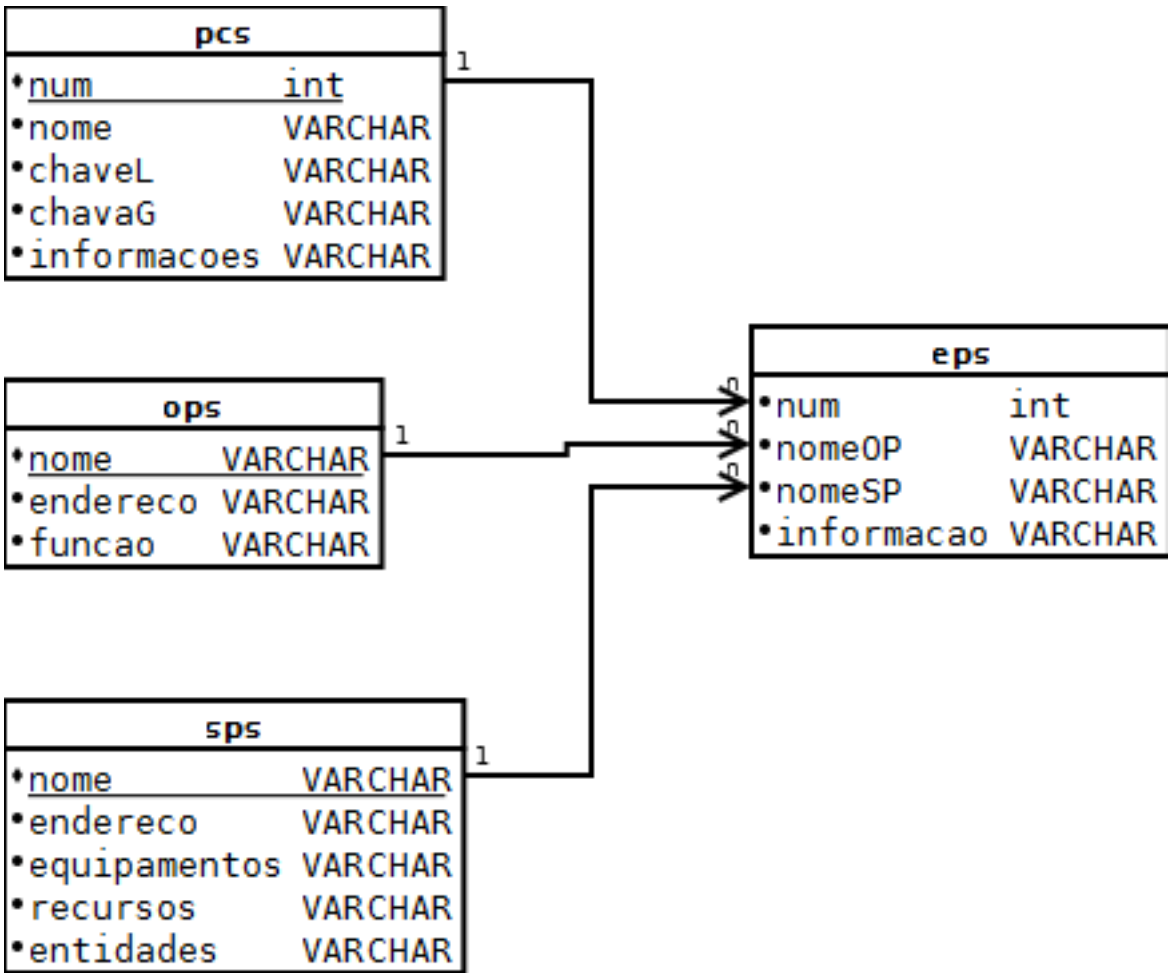


Figura 6.5: Representação das relações entre as tabelas

6.2 Web Services

Seguindo a descrição de serviços pela *Service oriented architecture Markup Language* (SoaML) em (OMG, 2008), os serviços são descritos como interfaces que se conectam a usuários, que prestam o serviço.

O serviço *"SolicitaOperador"* identificado no Capítulo 5, descrito na Figura 6.6, possui 5 métodos, 3 funcionalidades descritas na modelagem e 2 recuperadores de informações[1].

Na Figura 6.6, o método *"EnviaPedidoOP"*, que é uma das funcionalidades desse serviço, recebe como entrada 2 parâmetros do tipo *String*, o primeiro contendo uma chave de leitura e o segundo o nome do usuário, ambas registradas no *"banco de dados"*. O retorno desse método é 1 parâmetro do tipo *String* contendo o número da posição que o *"operador"* registrou a

[1]Um método que é utilizado para consulta de uma informação de um repositório de dados, sem realizar alterações de outras informações

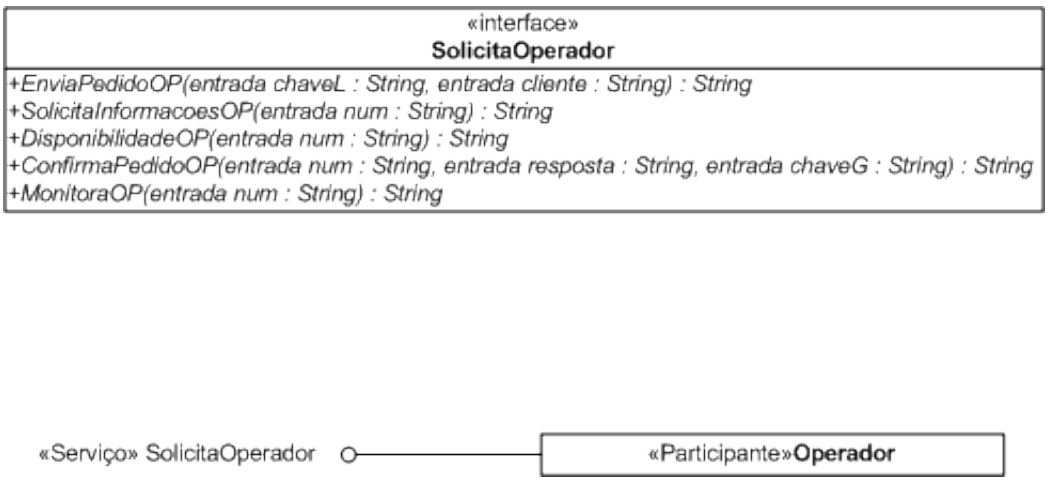


Figura 6.6: Representação do serviço *"SolicitaOperador"* em SoaML (OMG, 2008)

solicitação do *"cliente"*, ou *"FAIL"* caso não seja possível gravar a solicitação. Um exemplo de implementação desse método pode ser vista na Listagem 6.1.

Esse método da Listagem 6.1 utiliza uma função chamada *"conecta()"* que acessa um arquivo de configuração local com os dados necessários para conexão ao repositório de dados em que o *"operador"* gravou o pedido e realiza a conexão com esse repositório. Um exemplo desse código pode ser visto em Listagem 6.2.

Na Figura 6.6, o método *"SolicitaInformacoesOP"*, que é uma das funcionalidades desse serviço, recebe como entrada 1 parâmetro1 do tipo *String* que é o número da posição em que a solicitação do *"cliente"* foi registrada pelo *"operador"*. O retorno desse método é 1 parâmetro do tipo *String* contendo 2 possibilidades, *"SUCCESS"* caso o *"operador"* consiga alterar o *"Status"* da requisição do *"cliente"* de *"pendente"* para *"aguardando"* ou *"FAIL"* caso contrário. Esse método também utiliza a função *"conecta()"* da Listagem 6.2. Um exemplo de implementação desse método pode ser vista na Listagem 6.3.

Na Figura 6.6, o método *"DisponibilidadeOP"*, que é um recuperador de informações desse serviço, recebe como entrada 1 parâmetro do tipo *String* que é o número da posição em que a solicitação do *"cliente"* foi registrada pelo *"operador"*. O retorno desse método é 1 parâmetro de *Array* de *String* contendo o *"Status"* da solicitação e as informações (preço, prazo e outros) que o *"operador"* registrou no repositório de dados caso o *"Status"* da solicitação seja *"disponivel"*. Caso a consulta falhe, o retorno será um *Array* de *String* contendo *"FAIL"*. Esse método também utiliza a função *"conecta()"* da Listagem 6.2. Um exemplo de implementação desse método pode ser vista na Listagem 6.4.

Na Figura 6.6, o método *"ConfirmaPedidoOP"*, que é uma das funcionalidades desse

```
@WebMethod(operationName = "EnviaPedidoOP")
public String EnviaPedidoOP(@WebParam(name = "chaveL") String chaveL,
      @WebParam(name = "cliente") String cliente) {
   Statement homologacao = null;
   String resposta = "Inicial";
   String tabela = "fila";

   try {
      homologacao = conecta();
      String consulta = "INSERT INTO " + tabela + " (Nome,
            ChaveL)  VALUES ('" + cliente + "','" + chaveL + "')";
      if (homologacao.executeUpdate(consulta) == 0) {
         resposta = "FAIL"; //Caso a gravação dos dados falhe
      } else {
         consulta = "SELECT Num FROM " + tabela + " WHERE Nome =
               '" + cliente + "' AND ChaveL = '" + chaveL + "'";
         ResultSet res = homologacao.executeQuery(consulta);
         res.last();
         resposta = res.getString(1); //Caso contrário
      }
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.1: Exemplo de implementação do método *EnviaPedidoOP* em JAVA

serviço, recebe como entrada 3 parâmetros do tipo *String*, o primeiro contendo o número da posição em que a solicitação do *"cliente"* foi registrada pelo *"operador"*, o segundo contendo a resposta do *"cliente"* à proposta do *"operador"* e o terceiro contendo uma chave de gravação para as informações contidas no *"banco de dados"*. O retorno desse método é 1 parâmetro do tipo *String* contendo 2 possibilidades, *"SUCCESS"* caso o *"operador"* consiga alterar o *"Status"* da requisição do *"cliente"* de *"disponivel"* para *"confirmado"* ou *"negado"* (dependendo da resposta do *"cliente"*) e caso consiga gravar a informação da chave de gravação, ou *"FAIL"* caso contrário. Esse método também utiliza a função *"conecta()"* da Listagem 6.2. Um exemplo de implementação desse método pode ser vista na Listagem 6.5.

Por fim, na Figura 6.6, o método *"MonitoraOP"*, que é um recuperador de informações desse serviço, recebe como entrada 1 parâmetro do tipo *String* que é o número da posição em que a solicitação do *"cliente"* foi registrada pelo *"operador"*. O retorno desse método é 1 parâmetro de *String* contendo a porcentagem concluída do atendimento a etapa do pedido do *"cliente"* registrado em *"Concluido"* no repositório de dados do *"operador"*. Caso a consulta falhe, o retorno será *"FAIL"*. Esse método também utiliza a função *"conecta()"* da Listagem

6.2. Um exemplo de implementação desse método pode ser vista na Listagem 6.6.

De forma semelhante, o serviço *"SolicitaSP"* identificado no Capítulo 5, descrito na Figura 6.7, possui 5 métodos, 3 funcionalidades descritas na modelagem e 2 recuperadores de informações.

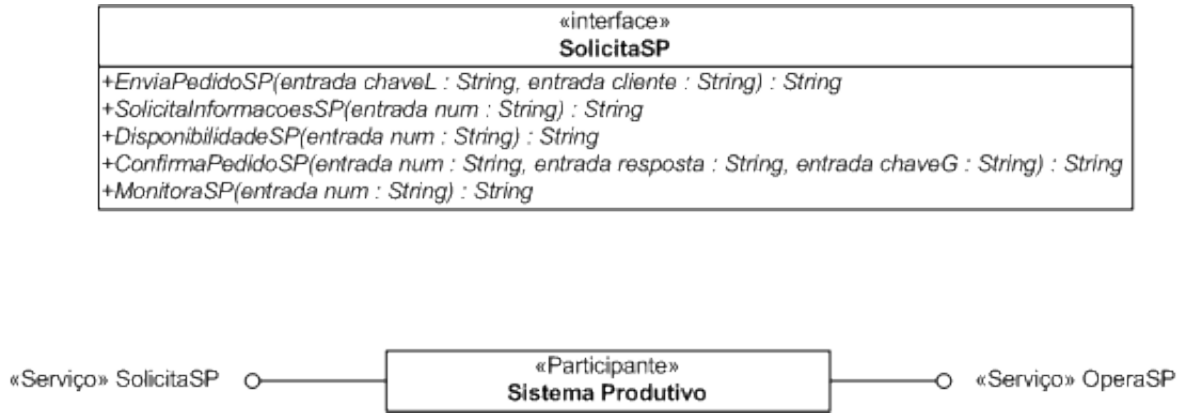


Figura 6.7: Representação do serviço *"SolicitaSP"* em SoaML (OMG, 2008)

Os métodos do serviço *"SolicitaSP"* (Figura 6.7) tem descrições similares as de *"SolicitaOperador"* (Figura 6.6). Os métodos *"EnviaPedidoSP"* (Listagem 6.7), *"SolicitaInformacoesSP"* (Listagem 6.8), *"DisponibilidadeSP"* (Listagem 6.9), *"ConfirmapedidoSP"* (Listagem 6.10) e *"MonitoraSP"* (Listagem 6.11) são similares aos métodos *"EnviaPedidoOP"* (Listagem 6.1), *"SolicitaInformacoesOP"* (Listagem 6.3), *"DisponibilidadeOP"* (Listagem 6.4), *"ConfirmapedidoOP"* (Listagem 6.5) e *"MonitoraOP"* (Listagem 6.6), respectivamente.

Outro serviço representado na Figura 6.7 é o *"OperaSP"* que possui uma implementação específica para cada tipo de equipamento ou recurso material presente no *"sistema produtivo"* que o provê. Um exemplo de método desse serviço pode ser visto na Listagem 6.12.

Os serviços de acesso aos *"bancos de dados"* seguem a idéia da SODA, ou seja, os repositórios de dados possuem estruturas específicas e o serviço acessa cada um desses *"bancos de dados"*, conhecendo a sua estrutura e faz a consulta que lhe foi solicitada. Caso se trate de uma consulta de busca de dados, após receber os dados de todos os *"bancos de dados"* o serviço filtra, trata e agrupa os dados e gera a apresentação dos mesmos para o solicitante. Caso se trate de uma consulta de inserção de dados, o serviço procura um *"banco de dados"* que aceite esses dados com a estrutura escolhida pelo próprio serviço. Caso se trate de uma consulta de alteração de dados, o serviço procura todos *"banco de dados"* que possuem o dado e necessitem ser alterados e faz em cada um a alteração solicitada de acordo com a estrutura do *"banco de dados"*.

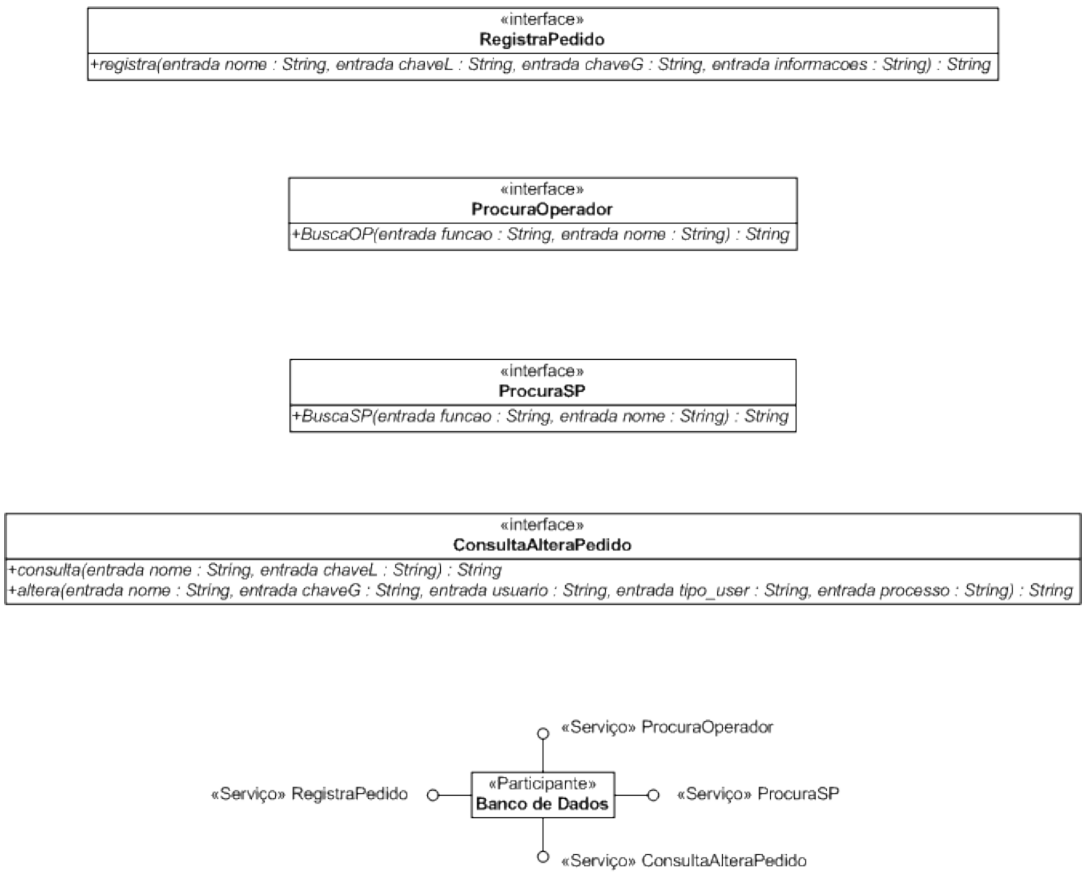


Figura 6.8: Representação dos serviços que utilizam SODA em SoaML (OMG, 2008)

Os serviços *"RegistraPedido"*, *"ProcuraOperador"*, *"ProcuraSP"* e *"ConsultaAlteraPedido"* identificados no Capítulo 5 e descritos na Figura 6.8, seguem a idéia acima.

O serviço *"RegistraPedido"* possui um único método com 4 parâmetros de entrada do tipo *String*, em que o primeiro parâmetro representa o nome que o *"cliente"* deseja registrar o pedido, o segundo parâmetro é uma chave que é usada por *"operadores"* e *"sistemas produtivos"* para consultar a leitura desse registro, o terceiro parâmetro é uma chave que é usada por *"operadores"* e *"sistemas produtivos"* para consultar a gravação de informações referentes a esse registro e o quarto parâmetro é uma informação estruturada que permite ao serviço consultar todas as informações do pedido do *"cliente"*. Um exemplo de implementação desse método pode ser visto em Listagem 6.13.

Esse método da Listagem 6.13 também utiliza uma função chamada *"conecta"*, mas com parâmetros de entrada diferentes da representada na Listagem 6.2. Essa função acessa um arquivo de configuração local com os dados necessários para conexão ao repositório de dados caso os parâmetros de entrada sejam vazios, caso contrário usa os parâmetros de entrada como dados, e realiza a conexão com esse repositório. Um exemplo desse código pode ser

visto em Listagem 6.14. Neste caso, os parâmetros de entrada podem ser dados obtidos a partir de uma tabela com informações de acesso aos *"bancos de dados"* externos.

O serviço *"ProcuraOperador"* possui um único método com 2 parâmetros de entrada do tipo *String*, em que o primeiro parâmetro representa o tipo de atividade que o *"operador"* cadastrado nos *"bancos de dados"* é capaz de exercer e o segundo parâmetro representa o nome de um *"operador"* cadastrado. Esse método busca em *"bancos de dados"* distribuídos os *"operadores"* cadastrados que possuem as funcionalidades ou o nome especificado pelo usuário do serviço (*"cliente"*) e retorna um parâmetro do tipo *String* contendo uma estrutura de dados com as informações dos *"operadores"* encontrados. Esse método também utiliza a função *"conecta"* (Listagem 6.14). Um exemplo de implementação desse método pode ser visto em Listagem 6.15, utilizando a estrutura de XML para apresentação dos dados e utilizando tabelas de dados com estruturas descritas na Figura 6.2.

O método descrito na Listagem 6.15 utiliza uma função chamada *"acrescenta"* que possui dois parâmetros de entrada do tipo *String*, o primeiro contendo o nome da etiqueta que será dada à informação em uma estrutura XML e o segundo contendo essa informação. A função retorna um parâmetro do tipo *String* com uma estrutura de XML. Um exemplo dessa função pode ser visto em Listagem 6.16.

De forma semelhante, o serviço *"ProcuraSP"*, da Figura 6.8, possui um único método, mas com com 4 parâmetros de entrada do tipo *String*, em que o primeiro parâmetro representa o tipo de equipamento que o *"sistema produtivo"* cadastrado nos *"bancos de dados"* possui em sua planta, o segundo parâmetro representa o tipo de recursos materiais que o *"sistema produtivo"* cadastrado nos *"bancos de dados"* possui em sua planta, o terceiro parâmetro representa o tipo de outras entidades físicas que o *"sistema produtivo"* cadastrado nos *"bancos de dados"* possui em sua planta e o quarto parâmetro representa o nome de um *"sistema produtivo"* cadastrado. Assim como o método *"BuscaOP"* que está na Listagem 6.15, o método *buscasp* retorna uma estrutura de dados com as informações dos *"sistemas produtivos"* encontrados. Um exemplo de implementação desse método pode ser visto em Listagem 6.17, que utiliza as funções *"conecta"* (Listagem 6.14) e *"acrescenta"* (Listagem 6.16), utilizando tabelas de dados com estruturas descritas na Figura 6.3.

O serviço *"ConsultaAlteraPedido"*, da Figura 6.8, possui 2 métodos que representam suas funcionalidades. O primeiro método é chamado de *"consulta"* e é usado pelos *"operadores"* e *"sistemas produtivos"* para procurar as informações do pedido de um dado *"cliente"*. Esse método possui 2 parâmetros de entrada do tipo *String*, o primeiro parâmetro é o nome com que o *"cliente"* registrou seu pedido nos *"bancos de dados"* e o segundo parâmetro é uma

chave de acesso a leitura que o *"cliente"* registra junto as demais informações do pedido para que mantenha as informações com acesso restrito. Esse método utiliza as funções *"conecta"* (Listagem 6.14) e *"acrescenta"* (Listagem 6.16). Um exemplo de implementação pode ser visto em Listagem 6.18 utilizando tabelas de dados com estruturas descritas na Figura 6.4.

O segundo método do serviço *"ConsultaAlteraPedido"* chama-se *"altera"* (Figura 6.8) e possui 5 parâmetros de entrada do tipo *String*. O primeiro parâmetro é o nome com que o *"cliente"* registrou seu pedido nos *"bancos de dados"*, o segundo parâmetro é uma chave de acesso a gravação que o *"cliente"* registra junto as demais informações do pedido para que mantenha as informações com acesso restrito, o terceiro parâmetro é o nome do usuário (*"operador"* ou *"sistema produtivo"*) que irá registrar o atendimento a uma das etapas do pedido, o quarto parâmetro é o tipo de usuário (*"operador"* ou *"sistema produtivo"*) que irá registrar a etapa do processo e o quinto parâmetro é uma descrição da etapa que será realizada por esse. Esse método retorna um parâmetro do tipo *String* com 3 possibilidades de resposta, caso seja possível gravar as informações e o outro tipo de usuário (*"operador"* ou *"sistema produtivo"*) já tenha sido registrado para o mesmo pedido, a resposta será o nome do outro tipo de usuário. Caso seja possível gravar as informações e nenhum outro tipo de usuário tenha sido registrado, a resposta será *"SUCCESS"*. Caso não seja possível gravar as informações, a resposta será *"FAIL"*. Esse método utiliza a função *"conecta"* (Listagem 6.14). Um exemplo de implementação pode ser visto em Listagem 6.19 utilizando tabelas de dados com estruturas descritas nas Figuras 6.4 e 6.1.

6.3 Testes

Os serviços descritos na seção anterior foram implementados em computadores do laboratório de Sistemas de Automação da Escola Politécnica da USP. Estes computadores estão conectados a uma rede interna que inclui os equipamentos do SP emulado.

Para o teste foram usados: quatro computadores para oferecer o serviço *"SolicitaOperador"* (descrito na seção anterior), com os usuários desses sendo *"operadores"*; quatro computadores para oferecer os serviços *"SolicitaSP"* e *"OperaSP"* (descritos na seção anterior), com usuários desses sendo *"sistemas produtivos"*; um computador para oferecer o serviço *"RegistraPedido"* (descrito na seção anterior); um computador para oferecer o serviço *"ProcuraOperador"* (descrito na seção anterior); um computador para oferecer o serviço *"ProcuraSP"* (descrito na seção anterior); um computador para oferecer o serviço *"ConsultaAlteraPedido"* (descrito na seção anterior); um computador para manter a tabela de *"banco de dados"* *"pcs"* (descrito na

seção 6.1); dois computadores para manter a tabela de *"banco de dados" "ops"* (descrito na seção 6.1); um computador para manter a tabela de *"banco de dados" "sps"* (descrito na seção 6.1); dois computadores para manter a tabela de *"banco de dados" "eps"* (descrito na seção 6.1); e dois computadores para interface com usuários *"clientes"*.

Os testes foram conduzidos para confirmar a execução dos processos especificados em diferentes cenários. Por exemplo, quando um dos *"clientes"* fez duas solicitações de pedidos: primeiro com 4 peças sendo 2 com *"corpo"* preto, 1 com *"corpo"* rosa e 1 com *"corpo"* prata; e depois com 2 peças com *"corpo"* prata. E um outro *"cliente"* fez uma solicitação de 3 peças sendo 1 com *"corpo"* preto, 1 com *"corpo"* prata e 1 com *"corpo"* rosa.

Para estes testes os *"sistemas produtivos"* estão associados (isto é, devidamente conectados) a apenas um dos subsistemas do SP disperso, que efetivamente realizam os processos físicos para a produção de produtos. Inicialmente, os *"operadores"* inicialmente foram cadastrados para teleoperar um subsistema. Depois, em outros testes, estes *"operadores"* foram cadastrados para operar até 2 subsistemas cada um.

Em todos os testes realizados os pedidos foram atendidos nos prazos combinados entre *"clientes"*, *"operadores"* e *"sistemas produtivos"* confirmando os resultados previstos na análise dos modelos.

6.4 Síntese do capítulo

A utilização de WSs para a negociação entre *"clientes"*, *"operadores"* e *"sistemas produtivos"* utiliza os *"bancos de dados"* para gravar e procurar informações sobre pedidos, etapas e cadastro dos usuários (*"operadores"* e *"sistemas produtivos"*).

Os WSs implementados com os métodos descritos anteriormente permitem que a negociação de cada etapa produtiva seja exclusivamente entre *"clientes"* e *"operadores"* ou *"sistemas produtivos"*.

Nos testes realizados, não foram encontrados problemas para o processamento de pedidos desde que as responsabilidades dos usuários, estipuladas no Capítulo 4, sejam cumpridas.

```
private Statement conecta() {
   Connection conn = null;
   Statement stat = null;
   BufferedReader file = null;
   String type, base, user, pass;
   String line;
   try {
      file = new BufferedReader(new FileReader(new File("solicita.conf")));
      while ((line = file.readLine()) != null) {
         if (line.startsWith("tipo"))
            type = line.substring(line.indexOf("=")+1, line.length()-1).trim();
         if (line.startsWith("usuario"))
            user = line.substring(line.indexOf("=")+1, line.length()-1).trim();
         if (line.startsWith("senha"))
            pass = line.substring(line.indexOf("=")+1, line.length()-1).trim();
         if (line.startsWith("endereco"))
            base = line.substring(line.indexOf("=")+1, line.length()-1).trim();
      }
   } catch (Exception e) {}
   try {
      if (type.equals("sqlserver")) {
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver")
               .newInstance();
         conn = DriverManager.getConnection("jdbc:sqlserver://" + base, user,
               pass);
         stat = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
               ResultSet.CONCUR_READ_ONLY);
      } else {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         conn = DriverManager.getConnection("jdbc:mysql://" + base, user, pass);
         stat = conexao.createStatement();
      }
   } catch (Exception ex) {}
   return stat;
}
```

Listagem 6.2: Exemplo de implementação da função *conecta* em JAVA

```
@WebMethod(operationName = "SolicitaInformacoesOP")
public String SolicitaInformacoesOP(@WebParam(name = "num") String num) {
   Statement homologacao = null;
   String resposta = "Inicial";
   String tabela = "fila";
   try {
      homologacao = conecta();
      String consulta = "UPDATE  " + tabela + " SET  Status =  'aguardando'
            WHERE  Num =  '" + num + "'";
      if (homologacao.executeUpdate(consulta) == 0) {
         resposta = "FAIL"; //Caso a gravação dos dados falhe
      } else {
         resposta = "SUCCESS"; //Caso contrário
      }
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.3: Exemplo de implementação do método *SolicitaInformacoesOP* em JAVA

```
@WebMethod(operationName = "DisponibilidadeOP")
public String DisponibilidadeOP(@WebParam(name = "num") String num) {
   Statement homologacao = null;
   String resposta = "";
   String tabela = "fila";
   String consulta;
   try {
      homologacao = conecta();
      consulta = "SELECT Status FROM " + tabela + " WHERE Num = '" + num + "'";
      ResultSet res = homologacao.executeQuery(consulta);
      res.last();
      String resp = res.getString(1);
      resposta = resposta + "\"" + resp + "\"";
      if (resp.equals("disponivel")) {
         consulta = "SELECT Info FROM processados WHERE Num = '" + num + "'";
         res = homologacao.executeQuery(consulta);
         while (res.next()) {
            resposta = resposta + ",\"" + res.getString(1) + "\"";
         }
      }
      resposta = resposta + ")";
   } catch (Exception ex) {
      resposta = "\"FAIL\")";
   }
   return "new Array(" + resposta;
}
```

Listagem 6.4: Exemplo de implementação do método *DisponibilidadeOP* em JAVA

```
@WebMethod(operationName = "ConfirmaPedidoOP")
public String ConfirmaPedidoOP(@WebParam(name = "num") String num,
      @WebParam(name = "resposta") String resposta,
      @WebParam(name = "chaveG") String chaveG) {
   Statement homologacao = null;
   String resp = "";
   String tabela = "fila";
   try {
      homologacao = conecta();
      String val = (resposta.equals("SIM")) ? "confirmado" : "negado";
      String consulta = "UPDATE  " + tabela + " SET  Status =  '" + val + "'
            WHERE  Num =  '" + num + "'";
      if (homologacao.executeUpdate(consulta) == 0) {
         resp = "FAIL";
      } else {
         if (resposta.equals("SIM")) {
            consulta = "UPDATE  " + tabela + " SET  ChaveG =  '" + chaveG + "'
                  WHERE  Num =  '" + num + "'";
            resp = (homologacao.executeUpdate(consulta) == 0) ? "FAIL" :
                  "SUCCESS";
         } else {
            resp = "SUCCESS";
         }
      }
   } catch (Exception ex) {
      resp = "FAIL";
   }
   return resp;
}
```

Listagem 6.5: Exemplo de implementação do método *ConfirmaPedidoOP* em JAVA

```
@WebMethod(operationName = "MonitoraOP")
public String MonitoraOP(@WebParam(name = "num") String num) {
   Statement homologacao = null;
   String resposta = "";
   try {
      homologacao = conecta();
      String consulta = "SELECT Concluido FROM emprocesso WHERE Num =
            '" + num + "'";
      ResultSet res = homologacao.executeQuery(consulta);
      res.last();
      resposta = res.getString(1);
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.6: Exemplo de implementação do método *MonitoraOP* em JAVA

```
@WebMethod(operationName = "EnviaPedidoSP")
public String EnviaPedidoSP(@WebParam(name = "chaveL") String chaveL,
      @WebParam(name = "cliente") String cliente) {
   Statement homologacao = null;
   String resposta = "Inicial";
   String tabela = "fila";

   try {
      homologacao = conecta();
      String consulta = "INSERT INTO " + tabela + " (Nome,
            ChaveL)  VALUES ('" + cliente + "','" + chaveL + "')";
      if (homologacao.executeUpdate(consulta) == 0) {
         resposta = "FAIL"; //Caso a gravação dos dados falhe
      } else {
         consulta = "SELECT Num FROM " + tabela + " WHERE Nome =
               '" + cliente + "' AND ChaveL = '" + chaveL + "'";
         ResultSet res = homologacao.executeQuery(consulta);
         res.last();
         resposta = res.getString(1); //Caso contrário
      }
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.7: Exemplo de implementação do método *EnviaPedidoSP* em JAVA

```
@WebMethod(operationName = "SolicitaInformacoesSP")
public String SolicitaInformacoesSP(@WebParam(name = "num") String num) {
   Statement homologacao = null;
   String resposta = "Inicial";
   String tabela = "fila";
   try {
      homologacao = conecta();
      String consulta = "UPDATE  " + tabela + " SET  Status =  'aguardando'
            WHERE  Num =  '" + num + "'";
      if (homologacao.executeUpdate(consulta) == 0) {
         resposta = "FAIL"; //Caso a gravação dos dados falhe
      } else {
         resposta = "SUCCESS"; //Caso contrário
      }
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.8: Exemplo de implementação do método *SolicitaInformacoesSP* em JAVA

```
@WebMethod(operationName = "DisponibilidadeSP")
public String DisponibilidadeSP(@WebParam(name = "num") String num) {
   Statement homologacao = null;
   String resposta = "";
   String tabela = "fila";
   String consulta;
   try {
      homologacao = conecta();
      consulta = "SELECT Status FROM " + tabela + " WHERE Num = '" + num + "'";
      ResultSet res = homologacao.executeQuery(consulta);
      res.last();
      String resp = res.getString(1);
      resposta = resposta + "\"" + resp + "\"";
      if (resp.equals("disponivel")) {
         consulta = "SELECT Info FROM processados WHERE Num = '" + num + "'";
         res = homologacao.executeQuery(consulta);
         while (res.next()) {
            resposta = resposta + ",\"" + res.getString(1) + "\"";
         }
      }
      resposta = resposta + ")";
   } catch (Exception ex) {
      resposta = "\"FAIL\")";
   }
   return "new Array(" + resposta;
}
```

Listagem 6.9: Exemplo de implementação do método *DisponibilidadeSP* em JAVA

```
@WebMethod(operationName = "ConfirmaPedidoSP")
public String ConfirmaPedidoSP(@WebParam(name = "num") String num,
      @WebParam(name = "resposta") String resposta,
      @WebParam(name = "chaveG") String chaveG) {
   Statement homologacao = null;
   String resp = "";
   String tabela = "fila";
   try {
      homologacao = conecta();
      String val = (resposta.equals("SIM")) ? "confirmado" : "negado";
      String consulta = "UPDATE  " + tabela + " SET  Status =  '" + val + "'
            WHERE  Num =  '" + num + "'";
      if (homologacao.executeUpdate(consulta) == 0) {
         resp = "FAIL";
      } else {
         if (resposta.equals("SIM")) {
            consulta = "UPDATE  " + tabela + " SET  ChaveG =  '" + chaveG + "'
                  WHERE  Num =  '" + num + "'";
            resp = (homologacao.executeUpdate(consulta) == 0) ? "FAIL" :
                  "SUCCESS";
         } else {
            resp = "SUCCESS";
         }
      }
   } catch (Exception ex) {
      resp = "FAIL";
   }
   return resp;
}
```

Listagem 6.10: Exemplo de implementação do método *ConfirmaPedidoSP* em JAVA

```
@WebMethod(operationName = "MonitoraSP")
public String MonitoraSP(@WebParam(name = "num") String num) {
   Statement homologacao = null;
   String resposta = "";
   try {
      homologacao = conecta();
      String consulta = "SELECT Concluido FROM emprocesso WHERE Num =
            '" + num + "'";
      ResultSet res = homologacao.executeQuery(consulta);
      res.last();
      resposta = res.getString(1);
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.11: Exemplo de implementação do método *MonitoraSP* em JAVA

```
[WebMethod]
public string mTelLiga_Ventosa() {
   string retorno = "OK";
   opcClient opc;
   opcClientGroup group;
   try {
      opc = (opcClient)Application["opcClient"];
      group = opc.GetGroupByPosition(0);
      group.GetItemById("W003").Write("1");
      Thread.Sleep(mySleep);
      group.GetItemById("W003").Write("0");
   }
   catch (Exception e) {
      retorno = "NOK";
   }
   finally {
      group = null;
      opc = null;
   }
   return retorno;
}
```

Listagem 6.12: Exemplo de implementação de função para controle de ventosa do SP1-Alimentação (Capítulo 3) do serviço *"OperaSP"* em C# via OPC

```
@WebMethod(operationName = "registra")
public String registra(@WebParam(name = "nome") String nome,
      @WebParam(name = "chaveL") String chaveL,
      @WebParam(name = "chaveG") String chaveG,
      @WebParam(name = "info") String info) {
   Statement homologacao = null;
   Statement homolog2 = null;
   String resposta = "FAIL";
   try {
      homologacao = conecta("", "", "", "");
      ResultSet res = homologacao.executeQuery("SELECT tipo, endereco, usuario,
            senha FROM bancos");
      String consulta;
      ResultSet result2;
      boolean gravado = false;
      while (res.next() && !gravado) {
         homolog2 = conecta(res.getString(1), res.getString(2), res.getString(3)
               , res.getString(4));
         consulta = "INSERT INTO pcs VALUES ('" + nome + "','" + chaveL + "'
               ,'" + chaveG + "','" + info + "')";
         try {
            if (homolog2.executeUpdate(consulta) != 0) {
               consulta = "SELECT num FROM pcs WHERE Nome = '" + nome + "'
                     AND chaveG = '" + chaveG + "'";
               result2 = homolog2.executeQuery(consulta);
               result2.last();
               resposta = result2.getString(1);
               gravado = true;
            }
         } catch (Exception ex) {}
      }
   } catch (Exception ex) {
      resposta = "FAIL";
   }
   return resposta;
}
```

Listagem 6.13: Exemplo de implementação do método *"registra"* do serviço *"RegistraPedido"* em JAVA

```
private Statement conecta(String type, String base, String user, String pass){
   Connection conn = null;
   Statement stat = null;
   BufferedReader file = null;
   String line;
   if (type.equals("")&& base.equals("")&& user.equals("")&& pass.equals("")){
      try {
         file = new BufferedReader(new FileReader(new File("banco.conf")));
         while ((line = file.readLine()) != null) {
            if (line.startsWith("tipo"))
               type=line.substring(line.indexOf("=")+1, line.length()-1).trim();
            if (linhe.startsWith("usuario"))
               user=line.substring(line.indexOf("=")+1, line.length()-1).trim();
            if (linha.startsWith("senha"))
               pass=line.substring(line.indexOf("=")+1, line.length()-1).trim();
            if (line.startsWith("endereco"))
               base=line.substring(line.indexOf("=")+1, line.length()-1).trim();
         }
      } catch (Exception e) {}
   }
   try {
      if (type.equals("sqlserver")) {
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver")
               .newInstance();
         conn = DriverManager.getConnection("jdbc:sqlserver://" + base, user,
               pass);
         stat = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
               ResultSet.CONCUR_READ_ONLY);
      } else {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         conn = DriverManager.getConnection("jdbc:mysql://" + base, user, pass);
         stat = conexao.createStatement();
      }
   } catch (Exception ex) {}
   return stat;
}
```

Listagem 6.14: Outro exemplo de implementação da função *"conecta"* em JAVA

```
@WebMethod(operationName = "BuscaOP")
public String BuscaOP(@WebParam(name = "funcao") String funcao,
      @WebParam(name = "nome") String nome) {
   String resp = "";
   Statement homologacao = null;
   Statement homolog2 = null;
   try {
      homologacao = conecta("", "", "", "");
      ResultSet res = homologacao.executeQuery("SELECT tipo, endereco, usuario
            , senha FROM bancos");
      String consulta;
      ResultSet res2;
      resp = "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<sql>\n";
      while (res.next()) {
         homolog2 = conecta(res.getString(1), res.getString(2), res.getString(3)
               , res.getString(4));
         consulta = "SELECT nome, endereco, funcao FROM ops ";
         if(tipo.equals("mysql")){
            consulta += "WHERE INSTR(funcao, '" + funcao + "') > 0";
            consulta += (!nome.equals("")) ? " AND
                  INSTR(nome, '" + nome + "') > 0" : "";
         }
         else{
            consulta += "WHERE CHARINDEX('" + funcao + "', funcao) > 0";
            consulta += (!nome.equals("")) ? " AND
                  CHARINDEX('" + nome + "', nome) > 0" : "";
         }
         res2 = homolog2.executeQuery(consulta);
         while (res2.next()) {
            resp += acrescenta("operador", acrescenta("nome", res2.getString(1))
                  + acrescenta("endereco", res2.getString(2)) +
                  acrescenta("funcao", res2.getString(3)));
         }
      }
      resp += "\n</sql>";
   } catch (Exception ex) {
      resp = "FAIL";
   }
   return resp;
}
```

Listagem 6.15: Exemplo de implementação do método *"BuscaOP"* do serviço *"ProcuraOperador"* em JAVA

```
private String acrescenta(String nome, String cont) {
   String resp = "<" + nome;
   if (cont.equals(""))
      resp = resp.concat(">\n</" + nome + ">");
   else
      resp = resp.concat(">\n" + cont + "\n</" + nome + ">");
   return resp;
}
```

Listagem 6.16: Exemplo de implementação da função *"acrescenta"* em JAVA

```
@WebMethod(operationName = "BuscaSP")
public String BuscaSP(@WebParam(name = "equip") String equip,
      @WebParam(name = "recurso") String recurso,
      @WebParam(name = "entid") String entid,
      @WebParam(name = "nome") String nome) {
   String resp = "";
   Statement homologacao = null;
   Statement homolog2 = null;
   try {
      homologacao = conecta("", "", "", "");
      ResultSet res = homologacao.executeQuery("SELECT tipo, endereco, usuario
            , senha FROM bancos");
      String consulta;
      ResultSet res2;
      resp = "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<sql>\n";
      while (res.next()) {
         homolog2 = conecta(res.getString(1), res.getString(2), res.getString(3)
               , res.getString(4));
         consulta = "SELECT * FROM sps ";
         if(tipo.equals("mysql")){
            consulta += "WHERE INSTR(equipamentos, '" + equip + "') > 0 AND
                  INSTR(recursos, '" + recurso + "') > 0 AND
                  INSTR(entidades, '" + entid + "') > 0";
            consulta += (!nome.equals("")) ? " AND
                  INSTR(nome, '" + nome + "') > 0" : "";
         }
         else{
            consulta += "WHERE CHARINDEX('" + equip + "', equipamentos) > 0 AND
                  CHARINDEX('" + recurso + "', recursos) > 0 AND
                  CHARINDEX('" + entid + "', entidades) > 0";
            consulta += (!nome.equals("")) ? " AND
                  CHARINDEX('" + nome + "', nome) > 0" : "";
         }
         res2 = homolog2.executeQuery(consulta);
         while (res2.next()) {
            resp += acrescenta("SP", acrescenta("nome", res2.getString(1))
                  + acrescenta("endereco", res2.getString(2)) +
                  acrescenta("equipamentos", res2.getString(3)) +
                  acrescenta("recursos", res2.getString(4))+
                  acrescenta("entidades", res2.getString(5)));
         }
      }
      resp += "\n</sql>";
   } catch (Exception ex) {
      resp = "FAIL";
   }
   return resp;
}
```

Listagem 6.17: Exemplo de implementação do método *"BuscaSP"* do serviço *"ProcuraSP"* em JAVA

```
@WebMethod(operationName = "consulta")
public String consulta(@WebParam(name = "equip") String equip,
      @WebParam(name = "chaveL") String chaveL) {
   String resp = "";
   Statement homologacao = null;
   Statement homolog2 = null;
   try {
      homologacao = conecta("", "", "", "");
      ResultSet res = homologacao.executeQuery("SELECT tipo, endereco, usuario
            , senha FROM bancos");
      String consulta;
      ResultSet res2;
      Bool localizado = false;
      while (res.next() && !localizado) {
         homolog2 = conecta(res.getString(1), res.getString(2), res.getString(3)
               , res.getString(4));
         consulta = "SELECT nome, informacoes FROM pcs WHERE Nome = '" +
               nome + "' AND ChaveL = '" + chaveL + "'";
         if((res2 = homolog2.executeQuery(consulta))){
            res2.last();
            resp = acrescenta("nome", res2.getString(1));
            resp += acrescenta("info", res2.getString(2));
            resp = "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n" +
                  acrescenta("cliente", resp) + "\n</sql>";
            localizado = true;
         }
      }
   } catch (Exception ex) {
      resp = "FAIL";
   }
   return resp;
}
```

Listagem 6.18: Exemplo de implementação do método *"consulta"* do serviço *"ConsultaAlteraPedido"* em JAVA

```
@WebMethod(operationName = "altera")
public String altera(@WebParam(name = "equip") String equip,
      @WebParam(name = "chaveG") String chaveG,
      @WebParam(name = "usuario") String usuario,
      @WebParam(name = "tipo_user") String tipo_user,
      @WebParam(name = "processo") String processo){
   String consulta, resp = "FAIL";
   Statement homologacao, homolog2;
   ResultSet res, res2;
   try {
      homologacao = conecta("", "", "", "");
      res = homologacao.executeQuery("SELECT tipo, endereco, usuario
            , senha FROM bancos");
      Bool localizado = false;
      while (res.next() && !localizado) {
         homolog2 = conecta(res.getString(1), res.getString(2), res.getString(3)
               , res.getString(4));
         consulta = "SELECT num FROM pcs WHERE Nome = '" + nome + "' AND
               ChaveG = '" + chaveG + "'";
         res2 = homolog2.executeQuery(consulta);
         if(res2.next() == false){
            res = homologacao.executeQuery("SELECT * FROM eps WHERE num =
                  '"+res2.getString(1)+"'");
            if(res.next())
                  if(tipo_user.equals("operador")){
                     homologacao.executeUpdate("UPDATE eps SET nomeOP = '"+
                           usuario+"' WHERE num='"+res2.getString(1)+"'");
                     resp = res.getString(3);
                  } else{
                     homologacao.executeUpdate("UPDATE eps SET nomeSP = '"+
                           usuario+"' WHERE num='"+res2.getString(1)+"'");
                     resp = res.getString(2);}
            else{
                  if(tipo_user.equals("operador"))
                     homologacao.executeUpdate("INSERT INTO eps (num, nomeOP,
                           informacao) VALUES ('"+res2.getString(1)+"',
                           '"+usuario+"', '"+processo+"')");
                  else
                     homologacao.executeUpdate("INSERT INTO eps (num, nomeSP,
                           informacao) VALUES ('"+res2.getString(1)+"',
                           '"+usuario+"', '"+processo+"')");
                  resp = "SUCCESS";
            } localizado = true;
   }}} catch (Exception ex) { resp = "FAIL";}
   return resp;}
```

Listagem 6.19: Exemplo de implementação do método *"altera"* do serviço *"ConsultaAlteraPedido"* em JAVA

7 Conclusão

O trabalho realizado procurou inicialmente identificar os conceitos e técnicas associadas a sistemas distribuídos e orientação a serviços inicialmente propostos para a área de sistemas de informação visando à implementação prática de uma arquitetura de controle dos processos de um sistema produtivo (SP) disperso. Considerando que um dos problemas críticos neste tipo de sistema é a negociação entre os diferentes tipos de usuários envolvidos, foi desenvolvida a modelagem dos processos relacionados com a comunicação entre esses usuários. Baseado na natureza desses processos, explorou-se aqui o formalismo da rede de Petri e uma de suas extensões (o PFS – *production flow schema*) para sistematizar a construção de modelos e realizar a análise estrutural e comportamental do sistema. Foi considerado aqui um SP disperso devidamente emulado por equipamentos disponíveis no Laboratório de Automação de Sistemas da Escola Politécnica da Universidade de São Paulo. Depois da comprovação pela análise dos modelos do comportamento esperado da arquitetura de controle dos processos do SP e, com base ainda numa das características da rede Petri, os modelos foram então vistos como a especificação do sistema para a sua implementação prática. Então foi realizada a implementação da arquitetura de controle e os testes confirmaram o atendimento às especificações básicas dos processos.

Os estudos, análises e testes realizados foram fundamentais para adquirir maior experiência prática relacionada à concepção, projeto, implementação e operação de arquiteturas de controle distríbuido aplicado a SPs dispersos.

Os testes confirmaram que em SPs dispersos com recursos limitados e um número de pedidos de *"clientes"* muito superior a capacidade de atendimento a esses, decisões equivocadas por parte de qualquer usuário pode prejudicar o atendimento de todos os pedidos. Por exemplo, se um *"cliente"* possui um pedido que deveria ser atendido em duas etapas sequenciais e negocia indevidamente primeiramente o atendimento da segunda etapa e posteriormente o atendimento da primeira etapa, esse *"cliente"* pode gerar uma condição de *deadlock* no processo global e impedir a continuação do atendimento aos pedidos de outros *"clientes"*.

Na arquitetura de controle aqui implementada, a maior liberdade de decisões atribuída aos *"clientes"* limita a participação de outros *"clientes"*, devido ao nível de conhecimento dos processos que neste caso todos os *"clientes"* deveriam possuir. Com isso, a arquitetura de controle implementada comprova que se for considerado um ambiente heterogêneo de *"clientes"*, necessariamente, deve-se introduzir um outro elemento com função de supervisor.

De qualquer modo, a implementação e os testes realizados permitem compará-la com outras arquiteturas de controle de modo a se ter uma escala para mensuração de dados e adquirir mais experiência na implementação prática de SPs baseados em serviços.

A metodologia de pesquisa aplicada permitiu o domínio sobre conceitos teóricos de sistemas distribuídos, SPs dispersos e de ferramentas que suportam sua implementação como a arquitetura orientada a serviços (SOA – *service oriented architecture*), a arquitetura de bancos de dados orientada a serviços e os *web services*.

Em conformidade com o ciclo de desenvolvimento previsto na metodologia de pesquisa, observa-se que os modelos gerados para a arquitetura de controle seguiram especificações, inicialmente sem base em experiências práticas do funcionamento de SPs dispersos e, os testes com a arquitetura implementada indicam possibilidades de aprimoramento das especificações para a modelagem da arquitetura.

Os modelos desenvolvidos consideram o mesmo sequenciamento de etapas produtivas, mas com outros tipos sequenciamentos os conflitos e situações de *deadlock* devem ser revistos e, com isso, novos modelos e análises são necessárias para garantir a devida realização dos processos produtivos.

O ambiente em que foram realizados os testes possuía limitação de vários recursos. Em computação distribuída, também se tem alguma limitação de recursos, mas de capacidade é, em geral, muito superior em relação ao ambiente aqui disponível. Assim, um ponto de estudo futuro envolve a implementação e teste de SPs dispersos com maior número de subsistemas e com subsistemas capazes de atender a mesma etapa produtiva dos pedidos.

Bibliografia

ADAM, N. R.; ATLURI, V.; HUANG, W.-K. Modeling and analysis of workflows using Petri nets. *Journal of Intelligent Information System*, Kluwer Academic Publisher, Boston, v. 10, n. 2, p. 131–158, mar 1998.

AUSFELDER, C.; CASTELAIN, E.; GENTINA, J.-C. A method for hierarchical modeling of the command of flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics*, v. 24, n. 4, p. 564–573, apr 1994. ISSN 0018-9472.

BAILEY, M. Tele-manufacturing: rapid prototyping on the internet. *Computer Graphics and Applications, IEEE*, v. 15, n. 6, p. 20 –26, nov 1995. ISSN 0272-1716.

BASIRI, M.; BISHOP, A. N.; JENSFELT, P. Distributed control of triangular formations with angle-only constraints. *Systems & Control Letters*, v. 59, p. 147–154, 2010.

BERAN, P.; HABEL, G.; SCHIKUTA, E. SODA a distributed data management framework for the internet of services. In: *Seventh International Conference on Grid and Cooperative Computing, 2008. GCC '08.* Shenzhen, China: [s.n.], 2008. p. 292–300.

BOEHM, B. W. A spiral model of software development and enhancement. *Computer*, v. 21, n. 5, p. 61–72, May 1988. ISSN 0018-9162.

BONDAVALLI, A. et al. A new approach and a related tool for dependability measurements on distributed systems. *IEEE Transactions on Instrumentation and Measurement*, v. 99, p. 1 –12, 2010. ISSN 0018-9456.

BRAUER, W.; REISIG, W. Carl Adam Petri and "Petri nets". *Informatik-Spektrum*, Springer Berlin / Heidelberg, v. 29, n. 5, p. 369–381, oct 2006.

BRUUN, P.; MEFFORD, R. N. Lean production and the internet. *International Journal of Production Economics*, v. 89, n. 3, p. 247 – 260, 2004. ISSN 0925-5273. Manufacturing Strategy.

CAI, K.; WONHAM, W. Supervisor localization: A top-down approach to distributed control of discrete-event systems. *Automatic Control, IEEE Transactions on*, v. 55, n. 3, p. 605 –618, march 2010. ISSN 0018-9286.

CALVEZ, J. P. *Embedded Real-Time Systems:*. New York, USA: John Wiley & Sons, 1993. ISBN 0 471 93563 8.

CAMPBELL, D. Service oriented database architecture: APP server-lite? In: *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2005. p. 857–862. ISBN 1-59593-060-4.

CHENG, J. C. et al. A service oriented framework for construction supply chain integration. *Automation in Construction*, v. 19, n. 2, p. 245 – 260, 2010. ISSN 0926-5805.

CHO, S. K. et al. Teleoperation of a mobile robot using a force-reflection joystick with sensing mechanism of rotating magnetic field. *IEEE/ASME Transactions on Mechatronics*, v. 15, n. 1, p. 17 –26, feb. 2010. ISSN 1083-4435.

CHOW, T. et al. Ontology-based information sharing in service-oriented database systems. In: *IEEE International Conference on Services Computing, SCC*. Bangalore, India: [s.n.], 2009. p. 276–283.

DAVID, R.; ALLA, H. Petri nets for modeling of dynamic systems - a survey. *Automatica*, v. 30, n. 2, p. 175–202, feb 1994.

FARAHBOD, R. *Extending and Refining an Abstract Operational Semantics of the Web Services Architecture for the Business Process Execution Language*. Dissertação (Mestrado) — Sharif University of Technology, 2001.

GORADIA, A.; XI, N.; ELHAJJ, I. Internet based robots: applications, impacts, challenges and future directions. In: *IEEE Workshop on Advanced Robotics and its Social Impacts*. Nagoya, Japão: [s.n.], 2005. p. 73 – 78.

GREFEN, P. et al. Dynamic business network process management in instant virtual enterprises. *Computers in Industry*, Elsevier, v. 60, p. 86–103, 2009.

GROBA, C. et al. A service-oriented approach for increasing flexibility in manufacturing. In: *IEEE International Workshop on Factory Communication Systems, 2008. WFCS 2008.* Dresden, Alemanha: [s.n.], 2008. p. 415 –422.

GUSTIN, G. D. B. *Aplicação de redes de Petri interpretadas na modelagem de sistemas de elevadores em edifícios inteligentes.* Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, jan 2000.

HAMADI, R.; BENATALLAH, B. A Petri net-based model for web service composition. In: *Proceedings of the 14th Australasian Database Conference (ADC'03)*. Adelaide, Australia: Australian Computer Society, 2003. p. 191–200.

HAN, R. et al. A Petri net theory-based method for modeling web service-based systems. In: *4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08.* Dalian, China: [s.n.], 2008. p. 1–7.

HASEGAWA, K. et al. On resource arc for Petri net modelling of complex resource sharing system. *Journal of Intelligent and Robotic Systems*, Springer Netherlands, v. 26, n. 3-4, p. 423–437, nov 1999.

JENSEN, K. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Berlin: SpringerVerlag, 1992.

JUNQUEIRA, F.; VILLANI, E.; MIYAGI, P. E. A platform for distributed modeling and simulation of productive systems based on petri nets and object-oriented paradigm. In: *10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*. Catania: [s.n.], 2005. v. 1, p. 907–914. ISBN 978-078039402-5.

KANESHIRO, P. J. et al. Modeling of collision resolution algorithm in Lonworks networks. In: *Proceedings of ASME International Mechanical Engineering Congress and Exposition*. Seattle: ASME, 2008. v. 9, p. 743–749. ISBN 0791843033; 978-079184303-1.

KELLER, T.; DIETRICH, M. Management of SOA in public administration: A case study. In: *International Conference on Management Science and Engineering, 2009. ICMSE 2009.* Moscow, Russia: [s.n.], 2009. p. 2016–2020.

KIEPUSZEWSKI, B.; HOFSTED, A. ter; AALST, W. van der. Fundamentals of control flow in workflows. *Acta Informatica*, v. 39, n. 3, p. 143–209, march 2003. ISSN 0001-5903.

KIM, W. et al. WSCPC: An architecture using semantic web services for collaborative product commerce. *Computer in Industry*, Elsevier, v. 57, p. 787–796, aug 2006.

LASA, I. S.; CASTRO, R. de; LABURU, C. O. Extent of the use of lean concepts proposed for a value stream mapping application. *Production Planning & Control: The Management of Operations*, Taylor & Francis, v. 20, n. 1, p. 82–98, 2009. ISSN 0953-7287.

LEE, J.-S.; ZHOU, M.-C.; HSU, P.-L. An application of Petri nets to supervisory control for human-computer interactive systems. *IEEE Transactions on Industrial Electronics*, v. 52, n. 5, p. 1220–1226, Oct. 2005. ISSN 0278-0046.

LEITAO, P. Agent-based distributed manufacturing control: A state-of-art survey. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 22, p. 979–991, 2009.

LI, Q. et al. Business processes oriented heterogeneous systems integration platform for networked enterprises. *Computers in Industry*, v. 61, n. 2, p. 127 – 144, 2010. ISSN 0166-3615.

LI, Z.; ZHOU, M. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, v. 34, n. 1, p. 38–51, Jan. 2004. ISSN 1083-4427.

LI, Z.; ZHOU, M. Control of elementary and dependent siphons in Petri nets and their application. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, v. 38, n. 1, p. 133–148, Jan. 2008. ISSN 1083-4427.

LUO, R.; TZOU, J.-H.; LAN, C.-C. The development of web-based e-business system for rapid prototyping manufacturing. *The 29th Annual Conference of the IEEE Industrial Electronics Society, 2003. IECON '03.*, v. 2, p. 1290 – 1295, nov. 2003.

MAYERL, C.; VOGEL, T.; ABECK, S. *SOA-based Integration of Service Management Applications SOA-based Integration of Service Management Applications Integrated IT Service Management*. 2008.

MEDINA, I. et al. Proposal of a methodology for implementing a service-oriented architecture in distributed manufacturing systems. In: SEGUI, V. J. (Ed.). *Third Manufacturing Engineering Society International Conference: MESIC-09*. Alcoy, Espanha: AIP, 2009. v. 1181, n. 1, p. 622–632.

MELO, J. I. G. et al. Framework for collaborative manufacturing systems based in services. In: ABCM. *Proceedings of the 20th International Congress of Mechanical Engineering*. Gramado, 2009.

MELO, J. I. G. et al. A procedure for modeling and analysis of service-oriented and distributed productive systems. In: *IEEE International Conference on Automation Science and Engineering, CASE*. Washington, EUA: [s.n.], 2008. p. 941–946.

MELO, J. I. G.; MORALES, R. A. G.; MIYAGI, P. E. Supervisory system for hybrid productive systems based on Bayesian networks and OO-DPT nets. In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*. Hamburg, Alemanha: [s.n.], 2008. p. 1108–1111.

MIYAGI, P. E. *Controle Programável - Fundamentos do controle de sistemas a eventos discretos*. São Paulo, SP: Editora Edgard Blücher LTDA, 1996. ISSN 85-212-0079-X.

MIYAGI, P. E. et al. Petri net approach for modeling system integration in intelligent buildings. *Jounal of the Brazilian Society of Mechanical Sciences*, v. 24, n. 4, p. 341–350, nov 2002. ISSN 0100-7386.

MORALES, R. A. G.; MELO, J. I. G.; MIYAGI, P. E. Diagnosis and treatment of faults in productive systems based on Bayesian networks and Petri net. In: *IEEE International Conference on Automation Science and Engineering, CASE*. Scottsdale, EUA: [s.n.], 2007. p. 357–362.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, v. 77, n. 4, p. 541–580, apr 1989.

NASSAR, M. G. V. et al. Modeling and analysing of the material entry flow system in a pickling line process using Petri nets. In: *ABCM Symposium Series in Mechatronics*. Brasília, Brasil: [s.n.], 2008. v. 3, p. 444–453.

OASIS. *Reference Model for Service Oriented Architecture 1.0*. [S.l.], aug 2006.

OMG. *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS)*. [S.l.], 2008.

PALOMERA-PEREZ, M.; BENITEZ-PEREZ, H. Scheduling coordinated task. In: *4th IEEE Conference on Industrial Electronics and Applications, 2009. ICIEA 2009.* Xi'an, China: [s.n.], 2009. p. 3944 –3949.

PETERSON, J. L. Petri nets. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 9, n. 3, p. 223–252, 1977. ISSN 0360-0300.

SHAH, N. Process industry supply chains: Advances and challenges. *Computers & Chemical Engineering*, v. 29, n. 6, p. 1225 – 1236, 2005. ISSN 0098-1354.

SHAN, J. et al. Research and implementation of code generator for information system based on SOA. In: *International Conference on Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09.* Dalian, China: [s.n.], 2009. p. 143–147.

SHIMIZU, Y.; ZHANG, Z.; BATRES, R. *Frontiers in Computing Technologies for Manufacturing Applications*. London, Inglaterra: Springer, 2007. ISSN 1860-5168. ISBN 978-1-84628-955-2.

THOMAS, A.; ARTIBA, A. Modelling and control of productive system: Concepts and application. *International Journal of Production Economics*, v. 121, n. 1, p. 1 – 3, 2009. ISSN 0925-5273.

TOK, W. H.; BRESSAN, S. DBNet: A service-oriented database architecture. In: *17th International Conference on Database and Expert Systems Applications, 2006. DEXA '06.* Bilbao, Espanha: [s.n.], 2006. p. 727–731. ISSN 1529-4188.

VILLANI, E.; MIYAGI, P. E.; VALETTE, R. *Modelling and Analysis of Hybrid Supervisory Systems: A Petri Net Approach*. London, Inglaterra: Springer, 2007.

WANG, D.; TSE, Q. C. K.; ZHOU, Y. A decentralized search engine for dynamic web communities. *Knowledge and Information Systems*, Springer London, dec 2009. ISSN 0219-3116.

WANG, H.; HU, Q. Research and application of an integration platform for e-commerce system based on SOA. In: *International Conference on Management of e-Commerce and e-Government, 2009. ICMECG '09.* Nanchang, China: [s.n.], 2009. p. 424–427.

WANG, J.; SONG, Y.-T.; XIONG, Y. Service evaluation in SOA: Toward business/IT alignment. In: *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009. SNPD '09.* Daegu, Coréia do Sul: [s.n.], 2009. p. 310–315.

WANG, Q.; TAN, D. Research on the control of mobile robots in network based teleoperation system. In: *International Workshop on Intelligent Systems and Applications, ISA.* Wuhan, China: [s.n.], 2009. p. 1 –4.

WOLFERT, J. et al. Organizing information integration in agri-food - a method based on a service-oriented architecture and living lab approach. *Computers and Eletronics in Agriculture*, Elsevier, v. 70, n. 2, p. 389–405, mar 2010.

XU, X. et al. A novel modeling design method for automated storage and retrieval system based on Petri nets. In: *Proceedings of the IEEE International Conference on Automation and Logistics*. Jinan, China: IEEE, 2007. p. 2046–2051.

YAN, J. et al. Ontology of collaborative manufacturing: Alignment of service-oriented framework with service-dominant logic. *Expert Systems with Applications*, v. 37, n. 3, p. 2222 – 2231, 2010. ISSN 0957-4174.

YAU, S. S.; AN, H. G. Adaptive resource allocation for service-based systems. In: *Internetware '09: Proceedings of the First Asia-Pacific Symposium on Internetware*. New York, NY, USA: ACM, 2009. p. 1–7. ISBN 978-1-60558-872-8.

YOO, T.; JEONG, B.; CHO, H. A Petri nets based functional validation for services composition. *Expert Systems with Applications*, v. 37, p. 3768–3776, 2010.

ZHU, W.; MELO, W. Refactoring J2EE application for JBI-based ESB: A case study. In: *IEEE International Enterprise Distributed Object Computing Conference, EDOC*. Auckland, Nova Zelândia: [s.n.], 2009. p. 213–217. ISSN 1541-7719.

ZURAWSKI, R.; ZHOU, M. Petri nets and industrial applications: A tutorial. *IEEE Transactions on Industrial Eletronics*, v. 41, n. 6, p. 567–583, dec 1994.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo