UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DETECÇÃO E CORREÇÃO DE SITUAÇÕES DE DEADLOCK EM WORFLOW NETS INTERORGANIZACIONAIS

LUCIANE DE FÁTIMA SILVA

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUCIANE DE FÁTIMA SILVA

DETECÇÃO E CORREÇÃO DE SITUAÇÕES DE DEADLOCK EM WORFLOW NETS INTERORGANIZACIONAIS

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Stéphane Julia

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Detecção e Correção de situações de Deadlock em WorkFlow nets Interorganizacionais**" por **Luciane de Fátima Silva** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 03 de fevereiro de 2014

Orientador: _____

Prof. Dr. Stéphane Julia
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Carlos Eduardo Trabuco Dórea
Universidade Federal do Rio Grande do Norte

_____

Prof. Dr. Carlos Roberto Lopes
Universidade Federal de Uberlândia

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Data: 03 de Fevereiro de 2014

Autor:	**Luciane de Fátima Silva**
Título:	**Detecção e Correção de situações de Deadlock em WorkFlow nets Interorganizacionais**
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

Autor

"Nunca deixe que lhe digam que não vale a pena acreditar no sonho que se tem ou que seus planos nunca vão dar certo ou que você nunca vai ser alguém."

Renato Russo

Dedicatória

Dedico este trabalho aos meus pais Geraldo e Maria Luisa e ao meu querido irmão Eder.

Agradecimentos

Agradeço primeiramente a Deus, pela vida e pelas oportunidades que colocou em meu caminho, por me amparar nos momentos de dificuldades e dar forças para superá-los.

Aos meus pais e meu irmão, pela paciência e compreensão, por ouvir minha angústias e me suportar em todo momento que precisei. Agradeço todo o amor, carinho, amizade, cuidado e apoio, que são fatores essenciais para que eu consiga alcançar todos os meus objetivos com sucesso.

Ao meu orientador, Prof. Stéphane Julia, por seu profissionalismo, pela oportunidade e confiança depositadas em mim, por sua paciência e constante orientação para que eu pudesse concluir este trabalho.

A todos os professores que contribuíram pelo meu aprendizado. Em especial, ao Prof. Michel dos Santos Soares, pelo incentivo para que eu ingressasse no curso de Mestrado e por todo o conhecimento compartilhado.

Aos meus verdadeiros amigos, que não mediram esforços para me ajudar em todos os momentos. Em especial, agradeço a Geycy Dyany, pelo companheirismo em todos os finais de semana nos laboratórios da UFU para a finalização da escrita deste trabalho e aos amigos, Rafael e Taffarel, que me auxiliaram na revisão do texto.

A todos, o meu muito obrigada!

Resumo

Neste trabalho é proposta uma abordagem baseada na prevenção de *deadlocks* em *WorkFlow nets* Interorganizacionais para lidar com situações dessa natureza. Processos de negócio interorganizacionais são modelados por workflows interorganizacionais. Situações de *deadlock* nos processos de negócio interorganizacionais geralmente estão relacionadas a perdas durante trocas de mensagens entre vários processos de negócio. Dentro da teoria das redes de Petri, uma situação de *deadlock* é caracterizada pela presença de um sifão que pode ficar vazio. Depois de detectar e controlar as estruturas de sifão que levam às situações de *deadlock* nas WorkFlow nets Interorganizacionais, é proposta uma arquitetura distribuída para modelar as *WorkFlow nets* Interorganizacionais livre de *deadlock*. Em particular, o princípio básico consiste em definir novas *WorkFlow nets* compartilhadas entre os workflows originais que permitem remover os cenários responsáveis pelos *deadlocks*.

Palavras chave: Processos de negócio, redes de Petri, *WorkFlow net* Interorganizacional, Sifão, Deadlock.

Abstract

In this work, an approach based on Deadlock avoidance of Interorganizational WorkFlow nets is proposed to deal with these situations. Interorganizational business processes are modeled by Interorganizational WorkFlow nets. Deadlock situations in interorganizational business processes come generally related to losses during message exchanges between several business processes. Within the Petri net theory, a Deadlock situation is characterized by the presence of a siphon that can be empty. After detecting and controlling the Siphon structures that lead to Deadlock situations in Interorganizational WorkFlow nets, a method for the design of Interorganizational WorkFlow nets free of Deadlock is proposed. In particular, the basic principle is to define new WorkFlow nets shared among the original workflow processes that allow one to remove the scenarios responsible for the Deadlocks.

***Keywords*:** *Business process, Petri nets, Interorganizational WorkFlow net, Siphon, Deadlock.*

Sumário

Lista de Figuras

Capítulo 1

Introdução

Todo produto ou serviço realizado pelas organizações modernas é necessariamente o resultado de um processo, isto é, de um conjunto de atividades integradas e consistentes [Weske 2012]. Embora os estudos sobre processos e sua constante melhoria sejam feitos há vários anos, como, por exemplo, nos trabalhos de Fredick Winslow Taylor (*Taylorismo*) e Henry Ford (*Fordismo*), realizados no início do século XX, a compreensão dos processos de negócio tem se mostrado como tendência dentro das organizações modernas.

Conforme foi mencionado em [Maranhão e Macieira 2004], não exatamente abandonando a estrutura de funções na empresa, mas reduzindo sua importância, as empresas contemporâneas têm passado de maneira gradual a se organizarem orientadas aos processos que as permeiam. A maior vantagem desta orientação por processos é a visão que se tem na organização, revelando situações que podem gerar problemas, gargalos e insuficiências que algumas vezes podem permanecer invisíveis se a organização funcionar aparentemente de maneira correta.

Em [Pádua et al. 2004] é definido um processo de negócio como um conjunto de atividades que podem, ou não, serem executadas simultaneamente, com alguma especificação de controle e fluxo de dados entre as atividades. Para [Aalst e Hee 2002] o conceito de processo é fundamental para o entendimento de como o negócio opera e quais oportunidades existem para coordenar as atividades que o constituem.

Grandes estruturas organizacionais, instituições de ensino e governamentais, multinacionais, entre outras, estão envolvidas com uma grande quantidade de pessoas que realizam um amplo número de processos, os quais muitas vezes têm dependências e relações com outros processos dentro ou fora das organizações correspondentes. Isso

implica em processos cada vez maiores e mais complexos que requerem um controle mais rígido.

Na literatura o termo *workflow* é aplicado para representar a utilização de elementos computacionais para auxiliar a execução de processos [Hollingsworth 1994]. Segundo o Modelo de Referência de *Workflow* da *Worfkflow Management Coalition* [WfMC 1996], um *workflow* é a automação de processos ou fluxos de trabalho, por inteiro ou por partes, nos quais tramitam documentos, informações ou atividades entre os participantes do processo para que estes desenvolvam ações, respeitando um conjunto de regras ou procedimentos bem definidos. *Workflow* pode ser visto então como um conjunto de atividades relacionadas que coletivamente alcançam um objetivo comum.

Os principais objetivos do *workflow* são facilitar as comunicações, tornando-as mais simples e seguras, melhorar a criação colaborativa de produtos gerados dentro do fluxo de atividades (documentos, especificações, projetos, etc.), realizar uma divisão do trabalho mais eficaz de forma a distribuir os recursos de acordo com as necessidades do processo, alertar os elementos do processo quanto à ocorrência de importantes eventos e mudanças e melhorar o processo de tomada de decisão [WfMC 1996]. Para que estes objetivos sejam alcançados, os fluxos e a distribuição do trabalho devem ser controlados pelos Sistemas de Gerenciamento de *Workflow* (*WfMS*). Após a definição do processo, os *WfMS* verificam se as atividades são executadas na sequência correta e notificam o responsável sobre a execução das tarefas. Em [Aalst 1998], a meta fundamental do gerenciamento de *workflow* é ter a certeza de que atividades corretas são executadas pelas pessoas certas e no tempo certo.

Como as organizações modernas têm cada vez mais que lidar com processos administrativos complexos, os Sistemas de Gerenciamento de *Workflow* têm que atuar sobre processos compartilhados entre várias organizações. As organizações atuais estão cada vez mais inseridas em interligações e dependências com processos que extrapolam suas fronteiras organizacionais. Portanto, passamos de um ambiente intraorganizacional, em que cada empresa tem o completo controle do seu processo, para contextos interorganizacionais, com processos distribuídos entre várias organizações. Em [Xu et al. 2009] os autores apresentam os *workflows* interorganizacionais, que são *workflows* que abrangem processos distribuídos entre organizações diferentes. Os autores destacam também que gerenciar este tipo de *workflow* se torna mais difícil, por algumas vezes os processos internos de cada empresa, que fazem parte do *workflow* global, terem o controle do próprio *workflow* interno. Além disso, é preciso ter um controle eficiente sobre as comunicações entre os *workflows* locais e sobre os recursos compartilhados entre todos

os processos. Assim, *workflows* interorganizacionais exigem uma cooperação importante entre diversos processos locais para que todas as empresas envolvidas alcancem os objetivos definidos.

De acordo com [Murata 1989] as redes de Petri são apropriadas para modelar Sistemas de Tempo Real, uma vez que elas permitem a representação de situações de conflito, compartilhamento de recursos, representação de comunicações entre processos e restrições de precedência. Em [Aalst e Hee 2002] os autores consideram a teoria das redes de Petri como uma ferramenta eficiente para modelagem e análise de processos de negócio. Em [Aalst 1998] o autor afirma que há várias razões para usar redes de Petri na modelagem de processos de negócio: são fornecidas de forma independente, possuem semântica formal, natureza gráfica, expressividade, propriedades e técnicas de análise. Ele ainda destaca que tal formalismo previne ambiguidades e contradições que podem acontecer com a maioria das técnicas de diagramação informais. As *WorkFlow nets*, definidas por Aalst em [Aalst 1998], são redes de Petri que modelam *workflows*. Os *workflows* interorganizacionais são então modelados pelas redes de Petri denominadas *WorkFlow nets* interorganizacionais [Aalst et al. 2000].

Uma das propriedades das *WorkFlow nets* é a *Soundness*, que é um critério importante a ser satisfeito quando se trata de processos de negócio. De fato, boas propriedades de modelos formais bem definidos, tais como as *WorkFlow nets*, podem ser facilmente provadas quando os processos de negócio seguem uma estrutura rígida que não permite desvios relacionados à descrição do processo durante à execução em tempo real.

Num processo interorganizacional, cada parceiro de negócio tem de definir seus *workflows* privados que estão ligados a outros *workflows* pertencentes aos outros parceiros. As *WorkFlow nets* interorganizacionais correspondem a um conjunto finito de *WorkFlow nets* que estão fracamente acopladas por meio de mecanismos de comunicação geralmente do tipo assíncrono. Como a sincronização de processos paralelos pode levar a uma fonte potencial de situações de travamento [Aalst et al. 2000], é difícil estabelecer a propriedade *Soundness* para *workflows* interorganizacionais complexos. Por isso, mesmo provando que um processo pertencente a uma *WorkFlow net* é localmente *sound*, não existe a garantia de que todo o processo global seja *sound*. Desta maneira, mesmo se cada um dos processos pertencentes a *WorkFlow net* interorganizacional respeitem individualmente a propriedade *Soundness*, não existe a garantia para que o processo global também satisfaça a propriedade, como foi mostrado em [Aalst 1998].

As situações de total travamento são indesejadas quando se trata de *workflow*. Estes travamentos totais são conhecidos como *deadlock* que descrevem uma situação desfavorável quando a execução de um processo é bloqueada e não pode ser devidamente encerrada

ou concluída. De um modo geral, tais situações surgem devido as exigências que não podem ser satisfeitas [Holt 1972]. Uma situação de *deadlock* em um *workflow* se dá caso a execução de um processo seja interrompida e não possa ser continuada. Existem várias razões pelas quais um *workflow* pode ficar preso em um *deadlock*. A maioria delas são razões estruturais que impedem um processo de ser concluído [Maruta et al. 1998].

Detectar situações de *deadlock* em processos interorganizacionais em tempo de execução é difícil e usualmente requer um custo computacional alto [Shim et al. 2002]. Por isso é desejável detectar tais situações e removê-las antes da execução real do modelo. Ainda de acordo com [Shim et al. 2002], geralmente os *deadlock* estão relacionados em processos interorganizacionais com erros ou pelo fato de não terem sido consideradas algumas situações que podem ser encontradas durante a execução. E destaca também que, como geralmente os *workflows* interorganizacionais são maiores e mais difíceis de compreender, não se deve colocar toda a responsabilidade pela detecção de *deadlocks* nos responsáveis por modelá-los. Por isso, é indicado um processo de automação para localização de situações que levam ao travamento em tempo de execução.

Para encontrar e resolver problemas de *deadlocks* em *WorkFlow nets* interorganizacionais é preciso identificar uma estrutura especial nas redes de Petri nomeada Sifão. Em [Li e Zhou 2008] o sifão é definido como um conjunto de subredes locais, tais que o seu conjunto de transições de entrada está contido no conjunto de transições de saída. Como parte de sua característica, pelo fato de conter mais transições de saída que entrada, um sifão pode se esvaziar, ficando permanentemente livre de fichas. Se um sifão está vazio, as suas transições de saída tornam-se permanentemente desativadas, causando um *deadlock* parcial ou total. Vários autores têm desenvolvido algoritmos para detectar tais estruturas e métodos eficientes foram propostos por [Barkaoui e Abdallah 1995b], [Chu e Xie 1997] e [Iordache et al. 2002] para a síntese de supervisores que criam restrições para que a marcação do sifão nunca se torne completamente vazia, garantindo assim o não esvaziamento das fichas nos sifões.

A proposta deste trabalho é mostrar que as situações de *deadlock* nas *WorkFlow nets* interorganizacionais estão relacionadas às estruturas chamadas Sifões que se esvaziam. Pretende-se, em particular, apresentar um algoritmo conhecido para detecção dos sifões que se esvaziam e, em seguida, apresentar um procedimento de controle supervisório para garantir um número estritamente positivo de fichas nos sifões, de forma a remover as situações indesejadas dos modelos de *WorkFlow nets* interorganizacionais. Os resultados de análise e projeto obtidos foram utilizados para propor uma arquitetura distribuída e livre de *deadlocks* na qual o controle supervisório das situações foi feito sem modificar a estrutura de controle das *WorkFlow nets* locais.

1.1 Contribuições

Este trabalho apresenta quatro contribuições principais:

- mostrar que as situações de *deadlock* nas *WorkFlow nets* interorganizacionais estão relacionadas às estruturas nomeadas sifões que se esvaziam antes de completar a execução dos processos de negócio envolvidos;
- apresentar um procedimento sistemático para detecção destas situações e das estruturas relacionadas a elas para sistemas modelados com *WorkFlow nets* interorganizacionais;
- apresentar um procedimento sistemático para controle supervisório do sifão, criando restrições que impedem que o modelo de negócio interorganizacional alcance um estado de travamento;
- definir uma arquitetura distribuída livre de *deadlocks* baseada nos resultados obtidos da análise do modelo controlado.

1.2 Organização da dissertação

O texto desta dissertação encontra-se organizado da seguinte maneira.

No capítulo 2, as fundamentações teóricas necessárias para o entendimento deste trabalho são apresentadas. Na Seção 2.1 faz-se uma introdução sobre as definições das redes de Petri, bem como seus elementos e propriedades. Na Seção 2.2 são apresentadas as *WorkFlow nets* e a propriedade *Soundness*. As Seções 2.3, apresentam as *WorkFlow nets* interorganizacionais e a propriedade *Soundness* aplicadas a elas. A Seção 2.4 apresenta a ocorrência de situações de *deadlock* nas redes de Petri e a Seção 2.5 apresenta o controle supervisório destas situações. Por fim, a Seção 2.6 fornece conceitos sobre Arquitetura de Software.

No capítulo 3 é apresentada a revisão bibliográfica. Nele, as principais contribuições existentes na literatura abordando o tema de "situações de *deadlock* em *WorkFlow nets* Interorganizacionais" são descritas juntamente com as limitações existentes em cada proposta apresentada.

No capítulo 4, um procedimento sistemático é apresentado para detectar e corrigir situações de *deadlock* nas *WorkFlow nets* interorganizacionais. As seções 4.1 descrevem

a caracterização do problema proposto a ser solucionado neste trabalho. Nas seções 4.2 e 4.3 são realizadas respectivamente a análise do modelo e a especificação do controle supervisório. Por fim, na seção 4.4 é apresentada a proposta de arquitetura que modela uma *WorkFlow net* controlada e livre de *deadlock*.

O capítulo 5 apresenta um estudo de caso no qual se aplica todo o procedimento sistemático apresentado no capítulo 4 a uma *WorkFlow net* interorganizacional composta por três processos locais.

Finalmente, o Capítulo 6 apresenta a conclusão deste trabalho e as perspectivas de trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

Subdividido em 6 seções, este capítulo apresenta o aparato teórico sobre o qual esta dissertação se baseia. Deste modo, a Seção 2.1 fornece uma visão geral sobre as redes de Petri e suas propriedades, que são utilizadas para modelar e analisar formalmente os *workflows* interorganizacionais. Na Seção 2.2 são apresentadas as *WorkFlow nets* que modelam os processos de negócio e a propriedade *Soundness*, utilizada na verificação da corretude do modelo. As Seções 2.3, apresentam as *WorkFlow nets* interorganizacionais e a propriedade *Soundness* aplicada a elas. A Seção 2.4 apresenta a ocorrência de situações de *deadlock* nas redes de Petri e a Seção 2.5 apresenta o controle supervisório destas situações. Por fim, a Seção 2.6 fornece conceitos básicos essenciais para o entendimento sobre Arquitetura de Software.

2.1 Redes de Petri

A teoria das redes de Petri originou-se na tese de *Carl Adam Petri*, apresentada em 1962 à Universidade de *Darmstadt* em [Petri 1962]. Nos anos seguintes, impulsionado por um grupo de pesquisadores do MIT (*Massachussetts Institute of Tecnology*) e interessado pelo trabalho inicial de *Carl Adam Petri*, *Anatol W. Holt* lançou entre 1968 e 1976 as bases teóricas das redes de Petri [Cardoso e Valette 1997].

Segundo [Cardoso e Valette 1997], uma rede de Petri é uma ferramenta gráfica e matemática que se adapta a inúmeras aplicações em que as noções de eventos e de evoluções simultâneas são importantes. As redes de Petri são um tipo de grafo, bipartido e orientado, capaz de captar a dinâmica de Sistemas a Eventos Discretos [Murata 1989].

Representam, portanto, sistemas dinâmicos em que há relação de concorrência, paralelismo e sincronização de informações.

A aplicabilidade das redes de Petri como ferramenta para estudo de sistemas é considerada importante em [Peterson 1981] por permitir representação matemática, análise dos modelos e fornecer informações úteis sobre a estrutura e o comportamento dinâmico dos sistemas modelados. Como admitem a introdução de alterações na definição dos elementos de modelagem, as aplicações das redes de Petri podem ocorrer em áreas diferentes, tais como: redes de computadores e protocolos de comunicação, sistemas operacionais, programação paralela, banco de dados distribuídos, sistemas flexíveis de manufatura e na modelagem de processos de negócio.

Referente a suas características, o formalismo matemático das redes de Petri é um dos pontos mais importantes. Por serem formais, é possível realizar uma análise precisa do modelo para verificar propriedades tais como a precedência entre eventos, sincronização e a existência de situações que levam ao bloqueio da execução da rede. Além disso, permitem a visualização dos processos e a comunicação entre eles.

Caracterizadas por uma representação gráfica compacta que dispensa a enumeração explícita de todos os estados do sistema, as redes de Petri são eficientes para sistemas de grande porte. Caso seja utilizada a modelagem com autômatos, esta enumeração explícita de estados torna a modelagem mais custosa e quando se trata de sistemas de grande porte, é considerável o aumento no custo computacional associado à análise do modelo.

As redes de Petri apresentam inúmeras vantagens se comparadas a outros modelos de representação de sistemas [Rillo 1988], e têm-se mostrado bastante úteis para a modelagem conceitual, análise, simulação e controle de sistemas complexos. Elas apresentam as seguintes características:

- possuem capacidade para modelagem hierárquica, com fundamentação matemática bem desenvolvida, que pode ser utilizada para análise;
- o formalismo matemático inerente às redes de Petri permite melhor compreensão sobre o comportamento do sistema;
- englobam especificação, modelagem, análise, avaliação de desempenho e de implementação em uma única ferramenta;
- descrevem de maneira precisa e formal as sincronizações, possibilitando a segurança de funcionamento;

- diferentes propriedades de sistemas concorrentes, tais como conflito, concorrência, *deadlock*, sincronismo e exclusão mútua podem ser verificadas formalmente;
- a representação gráfica pode ser utilizada como documentação e os modelos podem ser simulados, de modo que não só a estrutura do sistema, mas também o comportamento dinâmico da especificação podem ser observados;
- representam o sistema em diferentes níveis de abstração e os processos individuais e inter-relacionados em sistemas distribuídos.

Segundo [Murata 1989] [Cardoso e Valette 1997], os elementos básicos de uma rede de Petri são: os lugares, as transições, as fichas e os arcos. Uma representação de uma rede de Petri, composta pelos seus elementos básicos pode ser vista na Figura 2.1(a). O lugar, representado por um círculo, pode ser interpretado como uma condição, um estado parcial, um estado de espera, algum procedimento, um conjunto de recursos, etc. É considerado o elemento passivo do modelo. A transição é representada por uma barra ou retângulo e é associada a um evento que ocorre. É o elemento ativo do modelo. A ficha, também conhecida como marca é representada por um ponto num lugar e é um indicador de que a condição associada ao lugar é verdadeira. Os arcos são representados por setas que ligam as transições aos lugares e os lugares às transições.

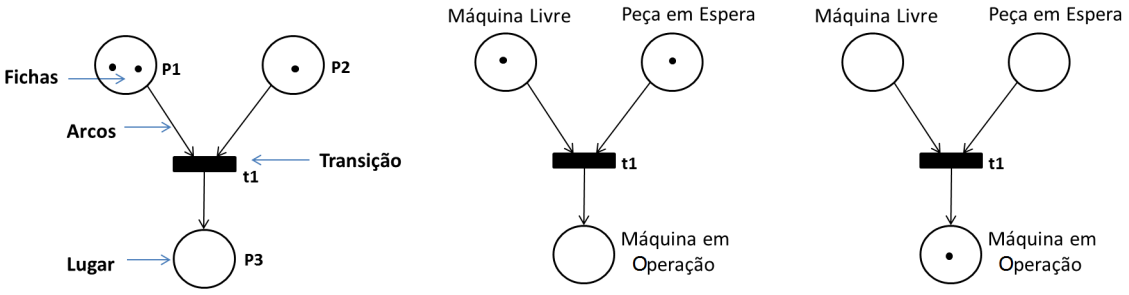

FIGURA 2.1: Exemplos de sensibilização e disparo de transição em uma rede de Petri.

O estado do sistema é dado pela distribuição das fichas nos lugares da rede de Petri. Um lugar marcado representa então um estado parcial do sistema. Portanto, a quantidade de fichas, em cada lugar e num determinado momento, define a marcação da rede, ou seja, o estado da rede naquele instante. Mudar de um estado para outro em uma rede de Petri corresponde à evolução dessa marcação, e é consequência direta do disparo de transições. Uma condição necessária para o disparo de uma transição é que ela esteja sensibilizada, isto é, cada lugar de entrada da transição deve possuir ao menos uma

ficha. Portanto, o disparo consiste em retirar uma marca de cada lugar de entrada da transição e colocar uma marca em cada lugar de saída da transição.

A definição de uma rede de Petri clássica conforme [Murata 1989] [Cardoso e Valette 1997] é a que se segue:

Definição 2.1. ("Rede de Petri Clássica") Uma rede de Petri clássica é uma quádrupla (P, T, Pré, Pós) onde:

- P é um conjunto finito de lugares;
- T é um conjunto finito de transições;
- Pré é uma relação que define os arcos que ligam os lugares às transições;
- Pós é uma relação que define os arcos que ligam as transições aos lugares.

Um lugar p é chamado de lugar de entrada de uma transição t se, e somente se, existe um arco direcionado de p para t. Por exemplo, considerando a rede de Petri da Figura 2.1(a), os lugares *P1* e *P2* são lugares de entrada da transição *t1*.

Um lugar p é chamado de lugar de saída de uma transição t se, e somente se, existe um arco direcionado de t para p. Por exemplo, considerando a rede de Petri da Figura 2.1(a), o lugar *P3* é um lugar de saída da transição *t1*.

Definição 2.2. ("Rede de Petri Marcada") Uma rede de Petri Marcada é uma dupla (R,M) onde:

- R é uma rede de Petri clássica;
- M é a marcação inicial dada pela aplicação M: P $\rightarrow$ N, onde N é o conjunto dos inteiros incluindo o zero;

M(p) é o número de fichas contidas no lugar p. A marcação M é a distribuição das fichas nos lugares, sendo representada por um vetor coluna cuja dimensão é o número de lugares. Para o exemplo da Figura 2.1(a) a marcação inicial é dada por M^T = [2 1 0] (M^T representa o vetor M transposto).

Um lugar p, quando se consideram as redes de Petri clássicas marcadas, contém em certo momento, zero ou mais fichas (*tokens*) representadas por pontos pretos (•). O número de fichas pode mudar durante a execução da rede.

As transições são componentes ativos em uma rede de Petri clássica, pois elas mudam a marcação da rede de acordo com as seguintes regras de disparo:

- uma transição t é dita sensibilizada se, e somente se, cada lugar de entrada p de t contém pelo menos uma ficha. Por exemplo, a transição *t1* da rede de Petri da Figura 2.1(c) não está sensibilizada. Já a transição *t1* da rede de Petri da Figura 2.1(b) está sensibilizada, pois há uma ficha em cada lugar de entrada desta transição;

- uma transição sensibilizada pode ser disparada. Se a transição t disparar, então t consome uma ficha de cada lugar de entrada p de t e produz uma ficha em cada lugar de saída p de t. As redes de Petri das Figuras 2.1(b) e 2.1(c) mostram um exemplo de disparo de transição, isto é, a marcação inicial é representada pela Figura 2.1(b) e a marcação final, obtida após o disparo da transição *t1*, é representada pela Figura 2.1(c). Neste caso, *t1* consome uma ficha de cada lugar de entrada (*Máquina Livre* e *Peça em Espera*) e produz uma ficha no lugar de saída (*Máquina em Operação*).

Uma rede de Petri pode ser representada algebricamente por uma matriz de incidência, a qual representa a relação entre os lugares e as transições, com dimensão $n \times m$: n é o número de lugares e m é número de transições. Esta matriz é formada pela subtração da matriz de incidência posterior *Pós* com a matriz de incidência anterior *Pré* ($I = Pós - Pré$).

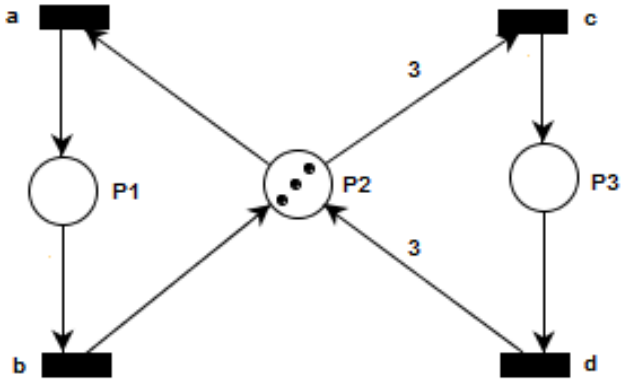

FIGURA 2.2: Elementos de Modelagem de uma *WorkFlow net*.

A notação matricial da rede de Petri da Figura 2.2 é dada por:

$$Pré = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad Pós = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \text{e} \quad I = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

A matriz de incidência fornece a evolução das fichas na rede quando ocorre o disparo das transições. Cada coluna corresponde à modificação por meio do disparo da transição associada. Por exemplo, a primeira coluna da matriz de incidência I indica que o disparo da transição *a* consiste na remoção de uma ficha do lugar *P2* e adição de uma ficha no lugar *P1*.

Uma sequência de disparo S pode ser representada por um vetor chamado *vetor característico da sequência*. Sua dimensão é igual ao número de transições da rede de Petri e cada componente pertencente à s representa o número de ocorrências da transição t numa sequência de disparo S. Considerando a Figura 2.2, o disparo da sequência S = *aac* a partir da marcação M_0, em que M_0^T = [0 3 0], leva a rede a uma marcação M_1, em que M_1^T = [1 2 0], e o vetor s^T (transposto do vetor característico S) seria s^T = [2 1 0 0]. Com base no vetor característico é possível saber que a transição *a* foi disparada duas vezes (s(1) = 2), *b* disparada uma vez e as demais, *c* e *d*, não foram disparadas na sequência S. O vetor característico não é o único, pois pode haver várias outras situações de disparo possíveis na rede.

Se o disparo de uma sequência S é tal que a partir de uma marcação inicial M_i obtemos uma marcação final M_f, então a **Equação Fundamental** da "rede de Petri clássica" é dada por:

$$M_f = M_i + I \cdot S \tag{2.1}$$

o que significa que se pode obter uma determinada marcação final[1] M_f, a partir de uma marcação inicial[2] M_i, mediante a adição da marcação inicial ao produto da matriz de incidência pelo vetor característico.

A equação fundamental descreve o comportamento da "rede de Petri clássica" possibilitando a sua análise estrutural e comportamental. Para exemplificar o funcionamento de tal equação, a Figura 2.3(a) e a sequência de disparo S^T = [1 1 0] serão consideradas.

A marcação final M_f obtida é apresentada abaixo (a qual pode ser observada na Figura 2.3(b)):

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

[1] A marcação final representa o estado final do sistema correspondente
[2] A marcação inicial representa o estado inicial do sistema correspondente

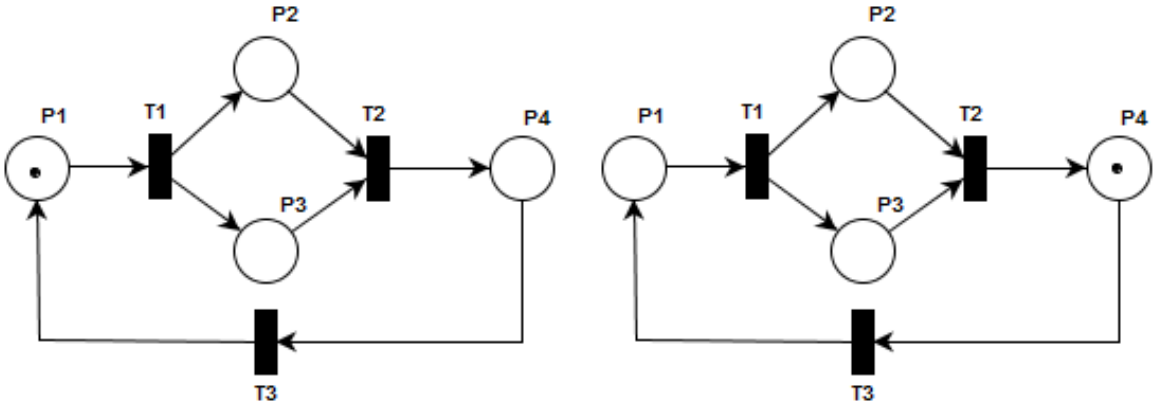

FIGURA 2.3: Sequência de disparo de transições.

O vetor característico S nem sempre corresponde a uma sequência de disparo válida e nem informa a ordem da execução das transições, entretanto, constitui uma condição necessária para verificação de alcançabilidade [Cardoso e Valette 1997].

2.1.1 Propriedades das redes de Petri

Em [Murata 1989] são definidas as propriedades relativas à rede de Petri marcada: alcançabilidade, limitabilidade, vivacidade e reiniciabilidade, reagrupadas sob o nome genérico de boas propriedades. Suas definições implicam considerações sobre o conjunto de marcações acessíveis a partir da marcação inicial. As propriedades relativas às redes de Petri não marcadas (que independem, portanto, da marcação inicial) permitem derivar métodos de cálculo, diretamente das definições, por meio da resolução de um sistema de equações lineares.

As boas propriedades das redes de Petri são:

- **alcançabilidade:** é a base fundamental para o estudo das propriedades dinâmicas de qualquer sistema. De acordo com [Girault e Valk 2001] podemos chamar de alcançabilidade a propriedade comportamental da rede de Petri, utilizada para o estudo de propriedades dinâmicas. Esta propriedade indica a possibilidade de atingirmos determinada marcação M_1, a partir de uma marcação inicial. Assim, uma marcação M_n é dita alcançável pela marcação M_0, se existe uma sequência de disparo de transições que, aplicada à rede a partir de M_0, possibilita alcançar a marcação M_n. Essa propriedade garante que certos estados ou operações sempre serão alcançados ou realizados. A verificação da alcançabilidade para um

modelo, sob o ponto de vista de um processo de negócio, é um meio de verificar a possibilidade de execução de um caso (cenário).

- **limitabilidade:** Uma rede de Petri marcada N é k-limitada se, e somente se, todos os seus lugares são k-limitados. Isto significa que o número de fichas em cada lugar não excede um número finito k para qualquer marcação alcançável a partir da marcação inicial. Em especial, uma rede marcada N é chamada de binária se, e somente se, todos os seus lugares são binários. Uma rede binária permite no máximo uma ficha em cada lugar e todos os arcos possuem valor unitário.

- **vivacidade:** Uma rede de Petri é considerada viva se qualquer transição t da rede é disparável, não importando a sequência de disparos seguida para disparar t. Isto significa que, para ser viva, a transição t deve poder ser sensibilizada a partir de qualquer marcação. Essa propriedade garante que o sistema é livre de *deadlock*, isto é, todas as transições da rede que o modela são disparáveis.

- **reiniciabilidade:** Uma rede é reiniciável se, qualquer que seja a sequência de disparo de transições seguida, é sempre possível voltar à marcação inicial M_0. Quando um sistema modelado tem seu funcionamento repetitivo, essa propriedade é de vital importância. Uma rede de Petri é repetitiva se existir uma sequência de disparos, associada a uma dada marcação, na qual todas as ações são executadas um número de vezes infinito. Se a sequência existir, mas apenas algumas ações forem disparadas ilimitadamente, a rede é dita parcialmente repetitiva.

Podemos agrupar os métodos formais para a análise das redes de Petri em duas categorias: a dos métodos que consideram principalmente a visão comportamental da rede, utilizando seu grafo de marcações e que dependem da marcação inicial, e os métodos que privilegiam a visão estrutural da rede, envolvendo a equação fundamental e que independem da marcação inicial da rede .

Um grafo de alcançabilidade (ou marcações acessíveis) representa o conjunto de estados alcançáveis e pode ser usado para verificar uma variedade de propriedades, por exemplo, se a rede é livre de *deadlock* [Murata 1989]. Este grafo de alcançabilidade é fundamental para o estudo das propriedades dinâmicas dos sistemas e é por intermédio dele que se deduz o total de casos atingíveis e verifica-se a quantidade de erros típicos que podem ocorrer na definição do processo.

O grafo de alcançabilidade de uma rede de Petri é um grafo direcionado e rotulado. Cada um de seus vértices representa uma marcação alcançável da rede e cada uma de

suas arestas representa o disparo de uma transição. O grafo das marcações acessíveis é dado por G = (V,A), em que V é o conjunto de vértices distintos (rotulados com a indicação da marcação correspondente) e A é o conjunto de todas as arestas rotuladas com o nome da transição correspondente.

Uma vez construído o grafo é possível determinar certas propriedades das redes de Petri. As marcações da rede são alteradas mediante o disparo das transições, ou seja, uma sequência de disparos resulta em uma sequência de marcações.

As propriedades estruturais são definidas por meio dos componentes conservativos de lugar e componentes repetitivos estacionários. A partir destes elementos estruturais, pode-se utilizar também a informação sobre a marcação, definindo-se, assim, os invariantes lineares de lugar e de transição, que fornecem algumas informações adicionais sobre o comportamento dinâmico da rede de Petri. Portanto, existem nas redes conjuntos de lugares e transições cujo comportamento não se altera durante o seu funcionamento. A identificação e interpretação de cada um destes conjuntos são essenciais, pois refletem certas propriedades da rede que podem ser interessantes para a análise do sistema modelado.

Os componentes conservativos de uma rede de Petri são representados por seus invariantes de lugar, ou seja, são conjuntos de lugares da rede nos quais a soma das fichas é constante durante todo o seu funcionamento.

Um invariante linear de lugar é uma função linear de marcação dos lugares cujo valor é uma constante que depende apenas da marcação inicial da rede. A equação que permite determinar a evolução da rede é a equação fundamental $M' = M + I_s$ onde s é uma sequência $s = [t1, t2, ..., tn]$. Para se obter uma soma ponderada de marcações, multiplica-se inicialmente a equação fundamental por um vetor f^T:

$$f^T \cdot M' = f^T \cdot M + f^T \cdot I_s. \tag{2.2}$$

A única maneira de tornar a equação independente das sequências de disparo s é tornando o produto $f^T \cdot I$ nulo. Um componente conservativo de uma rede de Petri é o conjunto de lugares $p_1 \in P$ correspondentes aos elementos não nulos f_1 do vetor coluna f solução da equação

$$f^T \cdot I = 0 \tag{2.3}$$

com $f \geq 0$ [Cardoso e Valette 1997].

Os componentes repetitivos são representados em seus invariantes de transição, isto é, são conjuntos de transições da rede que, ao serem disparados em determinada sequência, retornam à marcação inicial.

Seja M a marcação inicial e I a matriz de incidência, para obter-se $M' = M$, a sequência de disparo s deve ser tal que o vetor s' verifique

$$I' = 0. \tag{2.4}$$

Toda solução s' resultante desta equação é chamada componente repetitivo estacionário [Cardoso e Valette 1997].

Por meio dos invariantes lineares de lugar e transição, podemos por exemplo, mostrar que uma rede é limitada em vez de construir uma árvore de cobertura de crescimento exponencial.

2.1.2 Tipos de redes de Petri

As redes de Petri são resultado do trabalho de vários grupos de pesquisa em diversos países. A aplicação dessas teorias na modelagem de diversos sistemas pode ser constatadas em inúmeros trabalhos: [Murata 1989], [Peterson 1981], [Reisig 1985], [Rillo 1988], [Jensen 1994], [Iordache 2003].

2.1.2.1 Rede de Petri Clássica

As redes de Petri clássicas (lugar-transição), por serem as mais divulgadas, são normalmente referenciadas apenas como redes de Petri e foram definidas na Seção 2.1 deste capítulo.

2.1.2.2 Rede de Petri Temporal ou Temporizada

As redes de Petri temporizadas são uma evolução da teoria de redes de Petri introduzidas em [Ramchandani 1974], na qual é possível representar o tempo no modelo. Esse tempo pode ser associado ao lugar ou à transição.

Esses modelos são empregados em sistemas onde o tempo é um fator importante. De um modo geral, as redes de Petri temporais são adequadas para serem utilizadas em

aplicações de simulação, diagnóstico e supervisão, análise de desempenho, mas também podem ser usadas para resolver o problema do escalonamento de sistemas de produção [Julia et al. 2008].

Nas redes de Petri p-temporizadas o tempo é representado por durações associadas aos lugares do modelo [Sifakis 1977], enquanto nas redes de Petri t-temporizadas o tempo é representado por durações (números racionais positivos ou nulos) associadas às transições [Ramchandani 1974]. Nas redes de Petri t-temporais [Merlin 1974] o tempo é representado por um intervalo [Θmin,Θmax] associado a cada transição e a duração da sensibilização deve ser maior que o Θmin e menor que Θmax, ou seja, a transição só pode ser disparada dentro deste intervalo de tempo. Já nas redes de Petri p-temporais, o tempo é representado por um intervalo de tempo [Θmin,Θmax] associado a um lugar p. O valor Θmin representa o tempo mínimo que uma ficha no lugar p torna-se disponível para o disparo de uma transição. O valor Θmax representa o tempo máximo que a ficha em p deve ser utilizada para o disparo de um transição. Depois de Θmax a ficha se torna "morta", o que caracteriza uma violação de restrição e um funcionamento anormal do sistema. Quando temos o tempo associado à transição, determina-se uma duração de tempo para o disparo, ou seja, o tempo no qual as marcações não são disparáveis ou visíveis, devido ao disparo não ser instantâneo.

2.1.2.3 Rede de Petri Estocástica

As redes de Petri Temporizadas foram o passo inicial para a criação do formalismo das redes de Petri Estocásticas. A rede de Petri Estocástica é uma ferramenta para modelagem e avaliação de desempenho de sistemas envolvendo concorrência, não-determinismo e sincronização [Balbo et al. 1994]. A possibilidade de unir a habilidade das redes de Petri para descrever sincronização e concorrência com um modelo estocástico é o principal atrativo para obter-se uma avaliação quantitativa de sistemas computacionais complexos.

Os primeiros estudos das redes de Petri Estocásticas foram desenvolvidos por [Symons 1978], [Florin e Natkin 1985] e [Molloy 1981]. As Redes de Petri Estocásticas associam uma distribuição exponencial ao tempo de disparo de cada transição habilitada da rede. Dessa maneira, a rede passa a ser probabilística, sendo descrita por um processo estocástico.

Uma rede de Petri Estocástica é similar a uma Cadeia de Markov [Norris 1998] finita e de tempo contínuo, na qual as marcações são os estados da cadeia e as taxas de transição de estado da cadeia são retiradas das médias de tempo de disparo das transições da rede.

O conjunto de alcance da rede de Petri Estocástica é o mesmo de sua correspondente ordinária (rede de Petri que resulta ao se retirarem os tempos associados às transições), o que facilita a sua análise.

Portanto, nas redes de Petri estocásticas um tempo aleatório é associado ao disparo de uma transição. Esse modelo é usado para sistemas cujos eventos não podem ser bem definidos, como, por exemplo, o tempo entre a falha de uma máquina e outra. Para esse modelo, geralmente, o tempo é definido por meio de uma distribuição que segue uma lei exponencial, o que permite associar a rede a um processo *Markoviano* equivalente.

2.1.2.4 Rede de Petri Colorida

Para modelar sistemas constituídos de muitos componentes idênticos que interagem entre si, o modelo das redes de Petri clássicas pode exibir grande redundância. Isto se deve ao fato de que para diferenciar dois componentes idênticos é preciso especificar uma estrutura idêntica de sub-rede para cada um dos componentes.

Para solucionar este problema [Jensen 1991] criou as redes de Petri coloridas, que são uma extensão das redes de Petri clássicas, nas quais as fichas podem ter diferentes tipos de informação associada. Dessa forma, pode-se modelar o componente comum apenas uma vez e associar diferentes fichas a esse componente, tornando possível representar diferentes estruturas por meio das fichas. Para isso, a cada ficha é associado um valor denominado cor da ficha que pode representar tipos arbitrários de dados complexos como, por exemplo: inteiros, reais, registros, etc.

Uma rede de Petri clássica pode ser transformada em uma rede de Petri colorida mediante a substituição de conjuntos de lugares idênticos por um só lugar, contendo o tipo da cor associado.

2.1.2.5 Rede de Petri Predicado-Transição

As redes de Petri Predicado-Transição foram propostas por [Genrich 1987]. Nelas introduz-se o conceito de variáveis associadas às fichas, o que não existia nas redes de Petri clássicas, possibilitando, assim, enriquecer as informações referentes às fichas.

Nas redes de Petri Predicado-Transição as transições descrevem um conjunto de eventos e não somente um como nas redes clássicas [Cardoso e Valette 1997]. Elas incorporam o conceito de individualidade na marcação de uma rede de Petri. Os lugares da rede são

chamados de predicados e as fichas que neles se encontram representam as condições válidas do predicado. Ou seja, as fichas em um predicado definem sua extensão atual.

Para realizar a escolha das fichas e das transições a disparar, são associadas condições suplementares de disparos, que podem ser escritas como fórmulas lógicas utilizando as variáveis dos arcos de entrada, bem como operadores e predicados a elas associados [Villani 2000]. Assim, uma transição t_i só pode ser disparada quando a condição associada a ela for satisfeita. Para as variáveis associadas às fichas são atribuídos vetores associados com os arcos de entrada de t_i. De modo similar, a ação associada com a transição t_i define as variáveis que serão associadas aos vetores dos arcos de saída. Se nenhuma ação é definida, os valores das variáveis não são alterados [Tomiyama 2007].

2.1.2.6 Rede de Petri Orientada a Objetos

As redes de Petri Orientadas a Objetos [Sibertin-Blanc 1985] baseiam-se na integração das redes de Petri Predicado-Transição e do paradigma de orientação a objetos. Nesse tipo de rede, as fichas são consideradas como *n-uplas* de instâncias de classes de objetos e transportam verdadeiras estruturas de dados definidas como conjuntos de atributos de classes específicas. Nas transições, por sua vez, são associadas pré-condições e ações que, respectivamente, atuam sobre os atributos das fichas e modificam seus valores [Cardoso e Valette 1997].

Neste tipo de rede de Petri os lugares estão associados às classes; as transições são associadas às operações que atuam sobre os atributos localizados nos lugares de entrada; e os objetos correspondem às fichas da rede. Portanto, a operação de uma determinada transição t_i só poderá ser executada por um objeto se este estiver num lugar de entrada de t_i [Cardoso e Valette 1997]. A definição detalhada que fixa as regras de sensibilização e disparo das transições das redes de Petri a Objetos encontram-se em [Sibertin-Blanc 1985].

Em [Sanders 1998], as redes de Petri Orientadas a objetos foram usadas para a modelagem de sistemas de produção complexos com problemas baseados em restrições. A vantagem dessa modelagem é a possibilidade de se aproveitar a capacidade das redes de Petri para representar concorrência, controle de fluxo e restrições; beneficiando-se ao mesmo tempo da modularidade da orientação a objetos.

2.2 *WorkFlow nets*

Uma rede de Petri que modela um processo de *workflow* é uma *WorkFlow net* [Aalst 1998] [Aalst e Hee 2002]. As *WorkFlow nets* foram apresentadas em [Aalst 1998] com o objetivo de possibilitar a verificação de propriedades qualitativas de processos de negócio por meio da aplicação de redes de Petri.

Uma WorkFlow net (WF-net) se baseia na definição de meta-transições para representar as tarefas (tasks) e anotações que são associadas às tarefas para indicar o acionamento (gatilho) que dispara a sua execução.

Uma WorkFlow net satisfaz as seguintes propriedades [Aalst 1998]:

- tem apenas um lugar de início (denominado *Start*) e apenas um lugar de término (denominado *End*), sendo estes dois tratados como lugares especiais;
- o lugar *Start* tem apenas arcos de saída e o lugar *End* tem apenas arcos de entrada;
- uma ficha em *Start* representa um caso que precisa ser tratado e uma ficha em *End* representa um caso que já foi tratado.
- toda tarefa t (transição) e condição p (lugar) deve estar em um caminho que se encontra entre o lugar *Start* e o lugar *End*.

A WF-net pode ser utilizada para a verificação de corretude de processo de negócio mediante a análise de seu grafo de alcançabilidade. Por meio da análise do modelo, alguns erros de modelagem dos processos podem ser identificados, tais como:

- tarefas sem condição de entrada: quando não está especificada a condição necessária para a execução da tarefa;
- tarefas sem condição de saída: quando não se tem influência no processamento do caso;
- *Deadlock*: quando o processamento do caso é bloqueado, não sendo possível ocorrer uma condição que permita a continuação da execução deste caso;
- *Livelock*: quando o sistema entra em um ciclo no qual a condição de saída nunca irá ocorrer;

- atividades ainda a serem executadas após o caso ter alcançado o estado final;
- fichas que permanecem no modelo (sistema) após a conclusão do processamento do caso. Por exemplo, a geração de uma informação duplicada.

2.2.1 Processos

Um processo define quais tarefas precisam ser executadas e em qual ordem a execução deve ocorrer. De acordo com [Aalst 1998], modelar um processo de negócio em termos de uma *WF-net* é bem direto: transições são componentes ativos e modelam as tarefas, lugares são componentes passivos e modelam as condições (pré e pós) e as fichas modelam os casos.

Para ilustrar o mapeamento de processos em *WF-nets*, considera-se o processo de tratamento de reclamações apresentado em [Aalst e Hee 2002]: *"Uma reclamação é inicialmente gravada. Então, o cliente que efetuou a reclamação e o departamento responsável pela reclamação são contactados. O cliente é questionado para maiores informações. O departamento é informado sobre a reclamação. Estas duas tarefas podem ser executadas em paralelo, isto é, simultaneamente ou em qualquer ordem. Depois disso, os dados são recolhidos e uma decisão é tomada. Dependendo da decisão, ou um pagamento de compensação é efetuado, ou uma carta é enviada. Finalmente, a reclamação é armazenada"*. A Figura 2.4 mostra a *WF-net* que representa este processo.

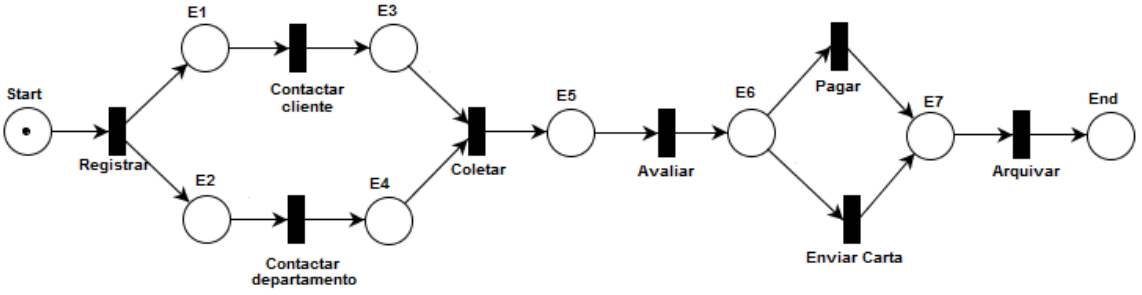

FIGURA 2.4: *WorkFlow net* para o processo de tratamento de reclamações e os seus acionamentos.

2.2.2 Roteamentos

De acordo com [Aalst 1998], pela rota de um caso, ao longo de uma série de tarefas, pode-se determinar quais tarefas precisam ser executadas (e em que ordem). Quatro construções básicas para o roteamento de tarefas são consideradas:

- sequencial: a forma mais simples de execução de tarefas, na qual uma tarefa é executada após a outra, havendo, claramente, dependência entre elas;
- paralela: mais de uma tarefa pode ser executada simultaneamente, ou em qualquer ordem. Neste caso, as tarefas podem ser executadas sem que o resultado de uma interfira no resultado das outras;
- condicional (ou rota seletiva): quando há uma escolha entre duas ou mais tarefas;
- iterativa: quando é necessário executar uma mesma tarefa múltiplas vezes.

Considerando o processo de tratamento de reclamações, mostrado na Figura 2.4, as tarefas *"Contatar cliente"* e *"Contatar departamento"* são um exemplo de roteamento paralelo, as tarefas *"Coletar"* e *"Avaliar"* são exemplo de roteamento sequencial e as tarefas *"Pagar"* e *"Enviar Carta"* são exemplo de roteamento condicional.

2.2.3 Acionamentos

Um acionamento é uma condição externa que guia a execução de uma tarefa sensibilizada [Aalst 1998]. Há quatro tipos distintos de tarefas:

- Usuário: uma tarefa é acionada por um recurso humano;
- Mensagem: um evento externo aciona uma tarefa sensibilizada;
- Tempo: uma tarefa sensibilizada é acionada por um relógio, isto é, a tarefa é executada em um tempo pré-definido.
- Automática: uma tarefa é acionada no momento em que é sensibilizada e não requer interação humana.

A Figura 2.5 mostra como cada um dos quatro tipos distintos de tarefas é associado às transições.

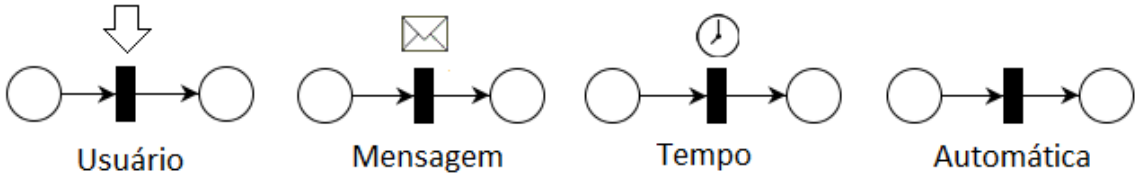

FIGURA 2.5: Tipos de tarefas nos acionamentos.

Nota-se que quando uma tarefa do tipo "Usuário" é considerada, essa tarefa é acionada por um recurso humano, isto é, há uma alocação de recurso associada a esta tarefa. Nos demais tipos de tarefa não há alocação de recursos associada.

O processo de tratamento de reclamações mostrado na figura 2.4 consiste em oito tarefas, das quais três são acionadas automaticamente (*Registrar*, *Coletar* e *Arquivar*) e cinco são acionadas por recursos humanos (*Contatar cliente*, *Contatar departamento*, *Avaliar*, *Pagar*, *Enviar Carta*).

2.2.4 *Soundness*

Soundness é o principal critério de corretude definido para *Workflow nets*. Em [Aalst e Hee 2002], são apresentadas as condições mínimas necessárias para uma *Workflow net* estar correta por meio da propriedade *Soundness*, definidas a seguir.

Definição 2.3. ("Soundness") Um procedimento modelado por uma *WorkFlow net* é *sound* se, e somente se:

- Para cada lugar alcançável M a partir de i, existe uma sequência de disparo que conduz do lugar M para o lugar o. Formalmente:

$$\forall_M (i \rightarrow M) \Rightarrow (M \rightarrow o). \tag{2.5}$$

- O lugar o é o único lugar alcançável a partir do lugar i com pelo menos uma ficha quando a execução terminar. Formalmente:

$$\forall_M (i \rightarrow M \wedge M \geq o) \Rightarrow (M = o). \tag{2.6}$$

- Não há nenhuma transição morta em (PN, i). Formalmente:

$$\forall_{t \in T} \exists_{M,M'} i \rightarrow M \rightarrow M'). \tag{2.7}$$

Em outras palavras, um processo é considerado logicamente correto (*sound*) se ele não contém nenhuma tarefa desnecessária e se todo caso iniciado pelo processo é totalmente concluído em algum momento, não restando nenhuma referência a ele (nenhuma ficha remanescente) no sistema. Portanto, uma *WF-net* é *sound* se, e somente, se os três requisitos abaixo são satisfeitos:

- para cada ficha colocada no lugar *i*, uma (e apenas uma) ficha aparecerá no lugar *o*;
- quando uma ficha aparece no lugar *o*, todos os outros lugares estão vazios, considerando o caso em questão;
- considerando uma tarefa associada à uma transição, é possível evoluir da marcação inicial até uma marcação que sensibiliza tal transição, ou seja, não deve haver nenhuma transição morta na *WF-net*.

A propriedade *Soundness* é um critério importante a ser satisfeito quando se trata de processos de negócio e a prova desse critério está relacionada com a análise qualitativa, no contexto das *WF-nets*.

2.3 *WorkFlow nets* interorganizacionais

Um fluxo de trabalho interorganizacional é um fluxo de trabalho que contém atividades que são executadas por várias organizações [Aalst 1998].

Os *workflows* interorganizacionais são *workflows* baseados em um tipo especial de interoperabilidade entre os processos: o *workflow* global consiste em dois ou mais *workflows* locais que operam de forma independente, mas precisam se comunicar e sincronizarem suas atividades em certos pontos, a fim de realizar corretamente o trabalho do processo global. Há duas maneiras básicas de interação entre os processos: a comunicação assíncrona (correspondente à troca de mensagens) e comunicação síncrona. Assim, o *workflow* interorganizacional resume-se em *workflows* locais independentes e uma estrutura de comunicação.

Para modelar *workflows* interorganizacionais, utilizaremos as *WorkFlow nets* interorganizacionais (*IOWF-nets*) [Aalst 1998] que são redes de Petri complementadas com dois conjuntos especiais utilizados para descrever a estrutura de comunicação entre os workflows locais: *CA* e *CS*. *CA* representa a comunicação assíncrona: se $(t, t') \in CA$,

então, a transição t deve ser executada antes da transição t'. CS representa o conjunto de elementos de comunicação síncrona: se $x \in CS$, então, todas as transições a partir de x têm que ser executadas ao mesmo tempo.

A Figura 2.6 apresenta uma *WorkFlow net* interorganizacional composta por duas *WorkFlow nets* locais: *LWF1* e *LWF2*. Estes dois *workflows* locais comunicam-se por meio de uma comunicação síncrona (*sc1*) e três elementos de comunicação assíncrona (*ac1*, *ac2* e *ac3*). Os elementos de comunicação assíncrona são conhecidos também como lugares de comunicação [Aalst 1998]. Estes lugares de comunicação são utilizados para modelar dependências casuais, como por exemplo, a tarefa *t11* em *LWF2* ter de esperar a tarefa *t1* em *LWF1* completar sua execução para que ela possa ser executada. Já o elemento de comunicação síncrono *sc1* força as transições *t7* e *t14* a serem executadas ao mesmo tempo.

Definição 2.4. ("IOWF-net") Uma *WorkFlow net* Interorganizacional (*IOWF-net*) é uma t-upla *IOWF-net* = PN_1, PN_2, ..., PN_n, PN_{AC}, AC, onde:

- $n \in \mathbb{N}$ é o número de *LWF-nets* (*WorkFlow nets* locais);
- Para cada $k \in \{1, ..., n\}$: PN_k é uma *WF-net* com lugar de entrada i_k e lugar de saída o_k;
- Para todo k, $l \in \{1, ..., n\}$: se $k \neq l$, então $(P_k \cup T_k) \cap (P_l \cup T_l) = \emptyset$;
- $T^* = \bigcup_{k \in \{1,...,n\}} T_k$, $P^* = \bigcup_{k \in \{1,...,n\}} P_k$, $F^* = \bigcup_{k \in \{1,...,n\}} F_k$ (relação entre os elementos das *LWF-nets*);
- P_{AC} é o conjunto de elementos de comunicação assíncrona (lugares de comunicação);
- $AC \subseteq P_{AC} \times \mathbb{P}(T^*) \times \mathbb{P}(T^*)$ é relação da comunicação assíncrona[3].

Cada elemento de comunicação assíncrona corresponde a um lugar em P_{AC}. A relação AC especifica o conjunto de transições de entrada e o conjunto de transições de saída para cada elemento de comunicação assíncrona [Aalst 1998].

[3] $\mathbb{P}(T^*)$ é o conjunto de todos os subconjuntos não vazios de T^*

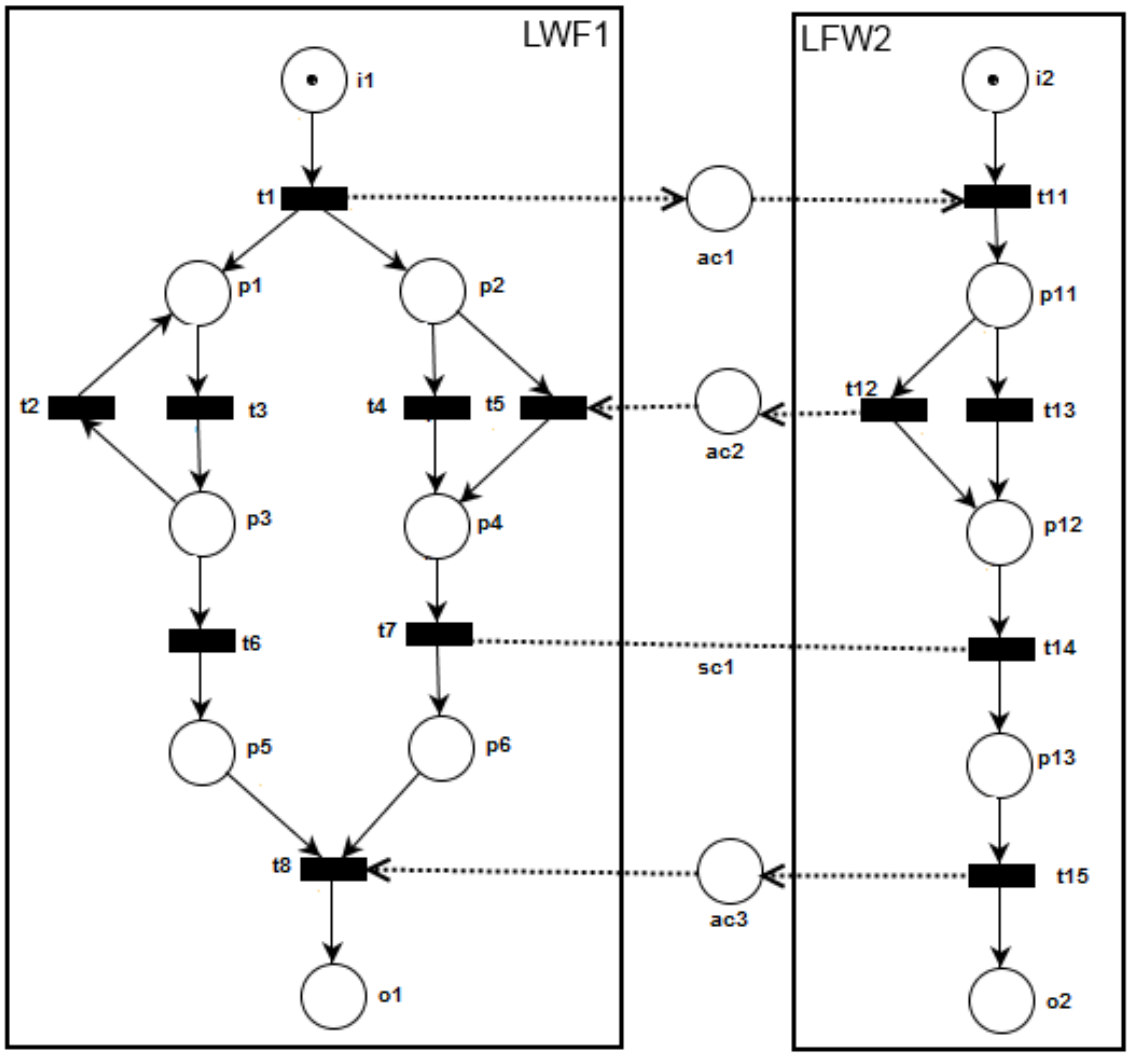

FIGURA 2.6: *Workflow net* interorganizacional composta por dois *workflows* locais e lugares de comunicação [Aalst 1998]

2.3.1 *Soundness* para *WorfkFlow nets* Interorganizacionais

A noção de *Soundness* já foi definida para as *WorkFlow nets*, expressando as condições mínimas que um processo deve satisfazer: um processo de negócio deve sempre ser capaz de completar um caso iniciado, qualquer caso deve terminar corretamente e cada tarefa deve contribuir ao menos para uma possível execução do processo de negócio [Aalst 1998]. Numa *WorkFlow net*, a conclusão de um caso é identificada por uma ficha no lugar final, então, podemos dizer que para executar o seu processo de forma correta, é necessário que para cada ficha (caso a ser tratado) no lugar de início, seja produzida uma ficha no lugar final e qualquer outro lugar da rede deve estar vazio.

De acordo com [Aalst 1998], em um *workflow* interorganizacional, mesmo se os *workflows* locais forem *sound*, podem aparecer erros de sincronização e é possível que os elementos de comunicação introduzam uma situação de travamento. Para que uma *Workflow net* interorganizacional seja *sound* é necessário que ela seja localmente e globalmente *sound*.

Para verificar se uma *WorkFlow net* interorganizacional é *sound* é necessário transformá-la de forma que se torne uma *WorkFlow net* com um único lugar de entrada e um único lugar de saída. Dado um *workflow* interorganizacional é possível construir uma rede estendida mediante a adição de um novo lugar (um lugar de entrada global) conectado a uma transição que, ao ser disparada, colocará uma ficha em cada lugar de entrada de cada *workflow* net local, e um novo lugar final (lugar de saída global) conectado a uma transição que consumirá uma ficha de cada lugar de saída dos *workflows* locais. Uma *WorkFlow ne*t interorganizacional será globalmente *sound* se esta rede estendida for *sound*.

A transformação da *WorkFlow net* interorganizacional apresentada na Figura 2.6 em uma *Workflow net* pode ser vista na figura 2.7, na qual foi adicionada o lugar de início único *i*, a transição *ti*, o lugar de fim *o* e a transição *to*.

2.4 *Deadlock* nas redes de Petri

O problema de *deadlock* é um assunto exaustivamente estudado em diversas áreas que envolvem comunicação, compartilhamento de informações e recursos [Tanenbaum 2010] [Silbertschatz et al. 2004]. O *deadlock* é caracterizado quando o fluxo dos processos é permanentemente impedido devido à falta de materiais, recursos e/ou informações [Coffman et al. 1971].

No contexto das redes de Petri, o *deadlock* pode ocorrer em virtude da competição entre os processos concorrentes pela utilização de recursos compartilhados ou por falhas de sincronização de comunicação entre esses processos concorrentes.

Em [Coffman et al. 1971] é apresentada uma formalização para o problema de *deadlock*, cuja evolução para ocorrência de tal situação é verificada se houver a ocorrência de quatro condições concomitantemente [Tanenbaum 2010] [Silbertschatz et al. 2004]:

- mútua exclusão: o processo utiliza o recurso alocado de modo exclusivo;

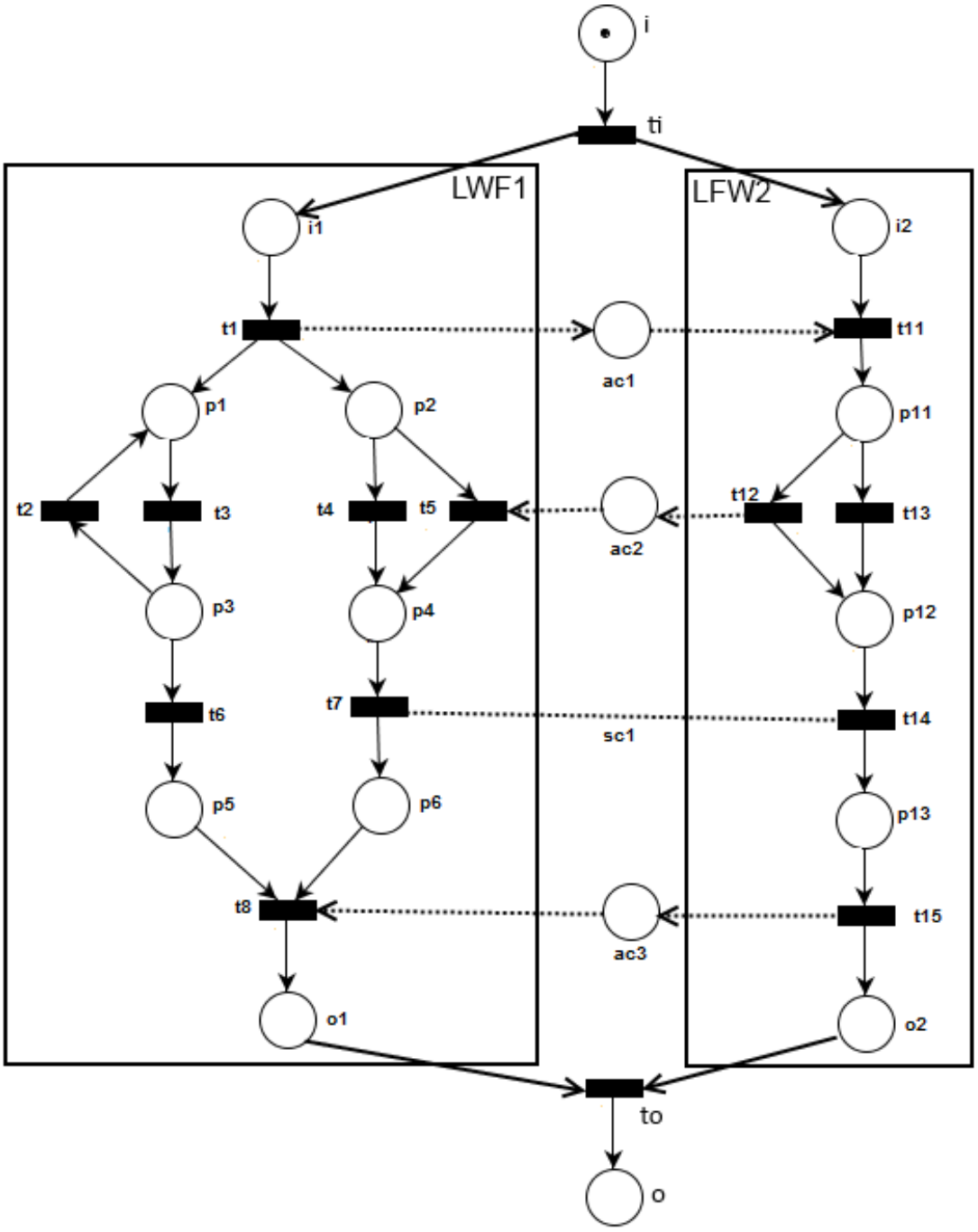

FIGURA 2.7: *Workflow net* interorganizacional modificada para transformá-la em uma única *WorkFlow net*.

- não há preempção: um recurso poderá ser liberado somente pelo processo que o alocou;
- retenção enquanto aguarda: o processo não libera o recurso alocado enquanto aguarda a alocação do próximo recurso;

- espera circular: é uma formação cíclica fechada de processos que aguardam a liberação de recursos alocados por outros processos pertencentes à mesma cadeia cíclica.

No trabalho [Aalst et al. 2000] é realizada a verificação para o problema de *deadlock* na perspectiva do *workflow*, na qual apenas a rota de casos é tratada. De acordo com os autores, potenciais *deadlocks* podem existir quando:

- múltiplas tarefas tentam alocar múltiplos recursos ao mesmo tempo;
- existem tarefas impondo restrições em um nível de exigência que nenhum recurso é qualificado para satisfazer

O primeiro tipo de *deadlock* pode acontecer frequentemente em sistemas de manufatura flexíveis onde ferramentas e espaço são necessários para completar operações ocasionando problemas. O segundo tipo ocorre quando não existe nenhum recurso adequado para realizar a tarefa.

Por definição, dada uma rede de Petri e uma marcação M' e seja T' um subconjunto de transições da rede, podemos dizer que a rede está em situação de *deadlock* na marcação M' em relação às transições de T', portanto, para qualquer que seja t_j pertencente a T', t_j está morta. Sendo T o conjunto de transições da rede, se $T = T$' então a situação da rede está em total impasse e nenhuma transição poderá ser disparada.

Uma rede de Petri é livre de *deadlock* se, para qualquer M' pertencente ao conjunto de marcações acessíveis da rede, existe uma transição t_j viva.

2.4.1 Detecção de *Deadlock*

Para analisar a vivacidade para sistemas modelados em rede de Petri, é importante lembrar uma definição de [Murata 1989]:

Seja N = {P,T} uma rede de Petri. Um sifão estrutural é um conjunto de lugares $S \subseteq P$ tal que $\bullet S \subseteq S\bullet$.

Em outras palavras, um sifão, também conhecido como *deadlock* estrutural, é um conjunto de lugares P, tal que o conjunto de transições de entrada de P esteja contido no conjunto de transições de saída de P. Como existem mais saídas de fichas que entradas,

esse conjunto de lugares pode ficar livre de fichas, com todos os lugares totalmente desmarcados (vazios), podendo provocar a situação de *deadlock*.

O sifão possui uma propriedade comportamental na qual caso ele não contenha nenhuma ficha em uma determinada marcação, ele permanecerá sem fichas para qualquer marcação subsequente [Tricas e Martinez 1995].

Em [Chu e Xie 1997] são destacadas algumas propriedades do Sifão:

- um sifão vazio (livre de marcações) permanece livre de marcações quando ocorre disparo de transições e, além do mais, suas transições de entrada e de saída não estão vivas;
- quando um *deadlock* é alcançado, o conjunto de lugares sem marcações formam um sifão.

Em [Tricas e Martinez 1995] é mostrado que uma rede de Petri em que exista um sifão insuficientemente marcado está em *deadlock*.

Existem vários algoritmos para encontrar sifões em redes de Petri. Alguns são baseados em Matriz de Incidência [Boer e Murata 1994], desigualdades [Ezpeleta et al. 1993], equações lógicas [Kinuyama e Murata 1986], Programação Matemática [Chu e Xie 1997] ou simples procedimento de pesquisa de primeira ordem aplicados em diferentes combinações de lugares [Jeng e Peng 1996].

O algoritmo apresentado a seguir para exemplificar a detecção de sifão em redes de Petri é o da Matriz de Sinais e Incidência proposto por [Boer e Murata 1994]. Para um melhor entendimento do algoritmo algumas definições básicas necessárias serão inicialmente apresentadas.

Para uma rede de Petri com n transições e m lugares, a matriz de incidência $I = [I_{ij}]$ é uma matriz n x m, cujo valor inicial é um dos seguintes:

- $I_{ij} = +$ se o lugar j é um lugar de saída da transição i;
- $I_{ij} = -$ se é um lugar de entrada da transição i;
- $I_{ij} = \pm$ se é lugar de entrada e de saída da transição i e
- $I_{ij} = 0$ em outro caso.

A adição denotada por $\oplus$ é uma operação binária comutativa no conjunto de quatro elementos, E = {+, −, 0, ±} definida como:

- $+ \oplus - = \pm$
- $c \oplus c = c, \forall c \in E$
- $\pm \oplus c = \pm, \forall c \in E$
- $0 \oplus c = c, \forall c \in E$

Um subconjunto de k lugares, S = p_1, p_2, ..., p_k, em uma rede de Petri N, é um sifão se, e somente se, a adição da k-ésima coluna do vetor da matriz de incidência de N, $I_1 \oplus I_2 \oplus ... \oplus I_k$, não contém nenhum valor +, onde I_j representa o vetor coluna correspondente ao lugar p_j, j = 1, 2, ..., k.

Assim, podemos concluir que todos os possíveis sifões em uma rede de Petri N podem ser gerados encontrando-se todas as possíveis combinações de vetores colunas da matriz de incidência I de N cuja soma não contém nenhum valor +. De outra forma, um conjunto de lugares dado é um sifão se a adição de suas respectivas colunas em I não contém nenhum valor +.

Para a compreensão do algoritmo, alguns termos são definidos:

1. Operação de Neutralização de + é a operação usada para se obter ±. Um valor + é dito neutralizado pela adição de − ou ±.
2. Um Lugar-sifão mínimo com relação ao lugar p é um sifão contendo um dado lugar p e um conjunto mínimo de lugares. Um lugar-sifão mínimo não é necessariamente um sifão.
3. $SEED_j$ é o lugar p_j correspondente à coluna I_j escolhida no passo 1 do algoritmo.
4. $PLANT_j$ é o conjunto de sifões que são encontrados pelo *SPROUT* a partir de $SEED_j$. O conjunto completo de sifões é obtido em $PLANT_j$ após o passo 3.
5. *LEAF* é um elemento de *PLANT* que é candidato a um lugar-sifão mínimo.

O algoritmo utiliza o vetor V e dois conjuntos S e SB. V = $I_1 \oplus I_2 \oplus ... \oplus I_r$ é a adição de r vetores colunas correspondentes aos r lugares em S = $p_1, p_2, ..., p_r$, e *SB* contém a coleção de sifões básicos que foram gerados. Índices e expoentes apropriados serão colocados em V e S durante a geração dos sifões.

As três etapas que constituem o algoritmo são: a identificação dos sifões *(LEAFs)*, a verificação se são sifões mínimos e a verificação de duplicatas. Serão utilizados como elementos de entrada a matriz de incidência I, o número de transições n e o de lugares m. A primeira etapa é usada para identificar os sifões modelo. A etapa 2 assegura que o algoritmo resulta num conjunto de lugares sifão mínimo que são iguais a sifões básicos [Boer e Murata 1994], e a etapa 3 precisa ser chamada pois *LEAF* idênticos podem ter sido gerados a partir de diferentes *SEEDs*.

Etapa 1:

Passo 1: Para a coluna I_j, j = 1, 2, ..., m na matriz de incidência I cujo lugar correspondente é chamado de $SEED_j$, faça o nível de recursão $r = 1$, faça $V_{jr} = I_j$ e vá para o passo 2.

Passo 2: Se V_{jr} tem um valor + na c-ésima coluna, procure uma coluna I_c que contenha um valor − ou ± na c-ésima coluna. Se encontrar I_c, adicione-o ao vetor V_{jr} para obter o vetor $V_{j(r+1)} = V_{jr} \oplus I_c$ contendo um valor ± na c-ésima coluna (um valor + neutralizado para ±), e repita para todas as neutralizações da coluna I_c possíveis. Serão produzidos novos conjuntos de $V_{j(r+1)}$.

Passo 3: Incremente r em 1. Repita o passo 2 até que não haja mais nenhum valor + em cada $V_{jr} = I_1 \oplus I_2 \oplus ... \oplus I_r$, ou nenhuma coluna para neutralização possa ser encontrada. Qualquer V_{jr} sem nenhum valor + representa um sifão $p_1, p_2, ..., p_r$. Esses novos sifões (*LEAFs* de $PLANT_j$) serão entrada para a etapa 2 do algoritmo.

Etapa 2:

Quando, no passo 1 (*SPROUT*), um novo candidato a sifão mínimo (novo *LEAF*) é encontrado, duas condições são verificadas: se um conjunto anterior de lugares (*LEAF*) contendo o novo *LEAF* já existe no atual conjunto de sifões encontrados ($PLANT_j$) a partir de $SEED_j$, então o conjunto de lugares anterior é sobreposto pelo novo conjunto de lugares; e se num subconjunto já existe, o novo conjunto não é adicionado a $PLANT_j$. Uma vez que todos m $PLANT_S$ ($PLANT_j$, j = 1, 2, ..., m) são gerados, o passo 3 é executado.

Etapa 3:

SB é inicializado com os sifões de $PLANT_1$ e para cada *LEAF* no próximo $PLANT_j$, j = 1, 2, 3, ..., m é adicionado a *SB* somente quando ele já não estiver contido em *SB*.

O exemplo a seguir apresentado em [Boer e Murata 1994] ilustra a utilização do algoritmo.

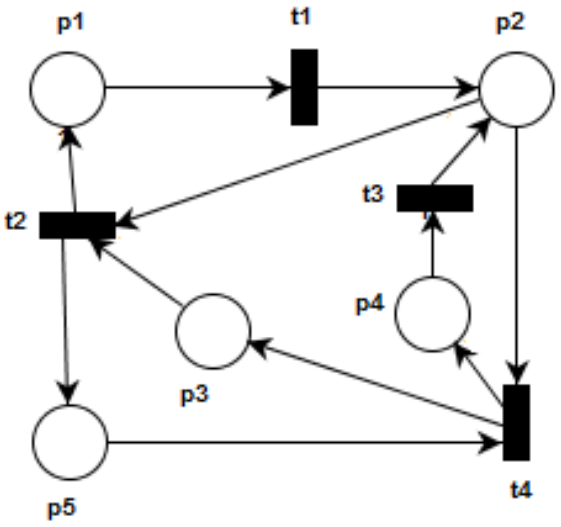

FIGURA 2.8: Rede de Petri utilizada para encontrar sifões.

Tomemos o exemplo da Figura 2.8 para ilustrar o algoritmo. A matriz de incidência de sinais I da rede é dada por:

$$I = \begin{bmatrix} - & + & 0 & 0 & 0 \\ + & - & - & 0 & + \\ 0 & + & 0 & - & 0 \\ 0 & - & + & + & - \end{bmatrix}$$

Passo 1: Tomemos a coluna 1:

$$I_1 = \begin{bmatrix} - \\ + \\ 0 \\ 0 \end{bmatrix} e \quad V_{11}^1 = \begin{bmatrix} - \\ + \\ 0 \\ 0 \end{bmatrix}, \quad S_{11}^1 = \{p1\}$$

onde o primeiro índice indica o *SEED* (lugar *p1*) e o segundo índice o nível de recursão (primeiro) e o expoente indica diferentes vetores V ou conjuntos S com índices idênticos.

Passo 2: V_{11}^1 tem um valor + na segunda linha (c = 2). As colunas 2 e 3 podem neutralizar este valor + que resulta em:

$$V_{12}^1 = \begin{bmatrix} - \\ + \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} + \\ - \\ + \\ - \end{bmatrix} = \begin{bmatrix} \pm \\ \pm \\ + \\ - \end{bmatrix}$$

$$S_{12}^{1} = \{p1, p2\}$$

$$V_{12}^{2} = \begin{bmatrix} - \\ + \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ - \\ 0 \\ + \end{bmatrix} = \begin{bmatrix} - \\ \pm \\ 0 \\ + \end{bmatrix}$$

$$S_{12}^{2} = \{p1, p3\}$$

Passo 3: Uma vez que V_{12}^{1} e V_{12}^{2} tem cada um valor +, o passo 2 é executado para ambos.

Passo 2: Para V_{12}^{1}, c = 3 que pode ser neutralizado por I_4 resulta em:

$$V_{13}^{1} = V_{12}^{1} \oplus I_4 = \begin{bmatrix} \pm \\ \pm \\ + \\ - \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ - \\ + \end{bmatrix} = \begin{bmatrix} \pm \\ \pm \\ \pm \\ \pm \end{bmatrix}$$

$$S_{13}^{2} = \{p1, p3, p4\}$$

Para V_{12}^{2}, c = 4 que pode ser neutralizado por I_2 ou I_5 resulta em:

$$V_{13}^{2} = V_{12}^{2} \oplus I_2 = \begin{bmatrix} - \\ \pm \\ 0 \\ + \end{bmatrix} \oplus \begin{bmatrix} + \\ - \\ + \\ - \end{bmatrix} = \begin{bmatrix} \pm \\ \pm \\ + \\ \pm \end{bmatrix}$$

$$S_{13}^{2} = \{p1, p3, p2\}$$

$$V_{13}^{3} = V_{12}^{2} \oplus I_5 = \begin{bmatrix} - \\ \pm \\ 0 \\ + \end{bmatrix} \oplus \begin{bmatrix} 0 \\ + \\ 0 \\ - \end{bmatrix} = \begin{bmatrix} - \\ \pm \\ 0 \\ \pm \end{bmatrix}$$

$$S_{13}^{3} = \{p1, p3, p5\}$$

Os vetores V_{13}^1 e V_{13}^3 correspondem, respectivamente, aos sifões {p1, p3, p4} e {p1, p3, p5}. Uma vez que eles não são subconjuntos um do outro, a etapa 2 não tem nenhum efeito e ambos, S_{13}^1 e S_{13}^3 são adicionados como *LEAF* em $PLANT_1$. Por enquanto, ambos são candidatos a lugar-sifão mínimo como relação ao lugar *p1*. É possível observar que o conjunto final de lugar-sifão mínimo não está disponível até que a recursão do passo 1 e a etapa 2 seja executada.

Como o vetor V_{13}^2 contém, ainda, um valor +, o passo 2 é executado para c = 3. A coluna I_4 pode neutralizar esse valor + que resulta em:

$$V_{14}^1 = V_{13}^2 \oplus I_4 = \begin{bmatrix} \pm \\ \pm \\ + \\ \pm \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ - \\ + \end{bmatrix} = \begin{bmatrix} \pm \\ \pm \\ \pm \\ \pm \end{bmatrix}$$

$$S_{14}^1 = \{p1, p3, p2, p4\}$$

Uma vez que não existe mais nenhum valor + em V_{14}^1 a etapa 2 é executada. Como $S_{13}^1 = \{p1, p2, p4\}$ é um subconjunto de $S_{14}^1 = \{p1, p3, p2, p4\}$, S_{14}^1 não é um lugar-sifão mínimo com relação a *p1*, e não é adicionado a $PLANT_i$. Posteriormente, {p1, p2, p4} tornará a ser selecionado com lugar-sifão mínimo com relação a *p3* e será adicionado como *LEAF* naquele ponto. Isto mostra a diferença entre sifão mínimo e lugar-sifão mínimo. Um lugar-sifão mínimo {p1, p2, p4} com relação a *p3* não é um sifão mínimo, pois {p1, p2, p4} é um dos seus subconjuntos e é um sifão mínimo. Este processo é mostrado na Figura 2.9.

Os demais lugares-sifão mínimos com relação aos outros quatro lugares *p2, p3, p4* e *p5* podem ser gerados de maneira similar e seus processos são ilustrados pela Figura 2.10. Observe que {*p4, p5, p2, p1*} na Figura 2.10(c) não é adicionada a $PLANT_4$, pois o subconjunto *LEAF* {*p4, p2, p1*} existe no mesmo *PLANT*.

A etapa 3 do algoritmo assegura que {*p2, p1, p4*} na Figura 2.10(a), {*p4, p2, p1*} na Figura 2.10(c) e p5, p3 na Figura 2.10(d) não são adicionados a *SB* por possuírem várias ocorrências. Assim, o conjunto final *SB* de sifões bases na rede de Petri na Figura 2.8 consiste em seis sifões básicos: {*p1, p2, p4*}, {*p1, p3, p5*}, {*p3, p2, p1, p4*}, {*p5, p3*}, {*p4, p5, p3*} e {*p5, p2, p1, p4*}.

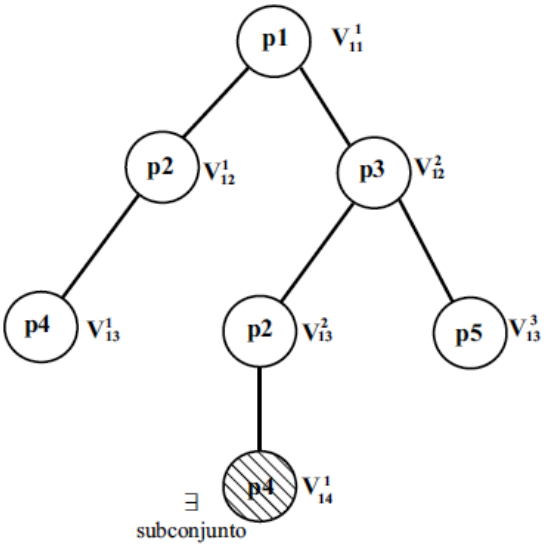

FIGURA 2.9: Geração de lugar-sifão mínimo com relação ao lugar p1.

2.5 Controle Supervisório de situações de *deadlock*

A teoria do Controle Supervisório para Sistemas de Eventos Discretos foi desenvolvida por [Ramadge e Wonham 1987]. De acordo com [Ramadge e Wonham 1987], uma planta possui todo comportamento possível do sistema a ser controlado, incluindo algumas situações indesejadas, tais como *deadlocks*. Uma planta também é vista como um sistema que gera eventos e que possui entradas de controle, por meio das quais alguns eventos podem ser desabilitados em determinados estados. O supervisor é um agente externo com habilidade de observar os eventos gerados pela planta e influenciar no seu comportamento por meio da entrada de controle.

Para este trabalho será utilizada a síntese do lugar de controle supervisório por meio do método dos invariantes de lugar [Moody et al. 1994] [Iordache e Antsaklis 2006]. Os invariantes de lugar correspondem aos conjuntos de lugares aos quais a soma de fichas permanece constante para todas as marcações alcançáveis pela rede. Um invariante de lugar é definido como qualquer vetor inteiro $x \in N^n$ que satisfaça:

$$x^T M = x^T M_0. \tag{2.8}$$

A equação 2.8 indica que a soma das fichas nos lugares do invariante permanece constante para qualquer marcação alcançável pela rede. Os invariantes de lugar podem ser obtidos mediante a obtenção de soluções inteiras para [Murata 1989]:

$$x^T I = 0 \tag{2.9}$$

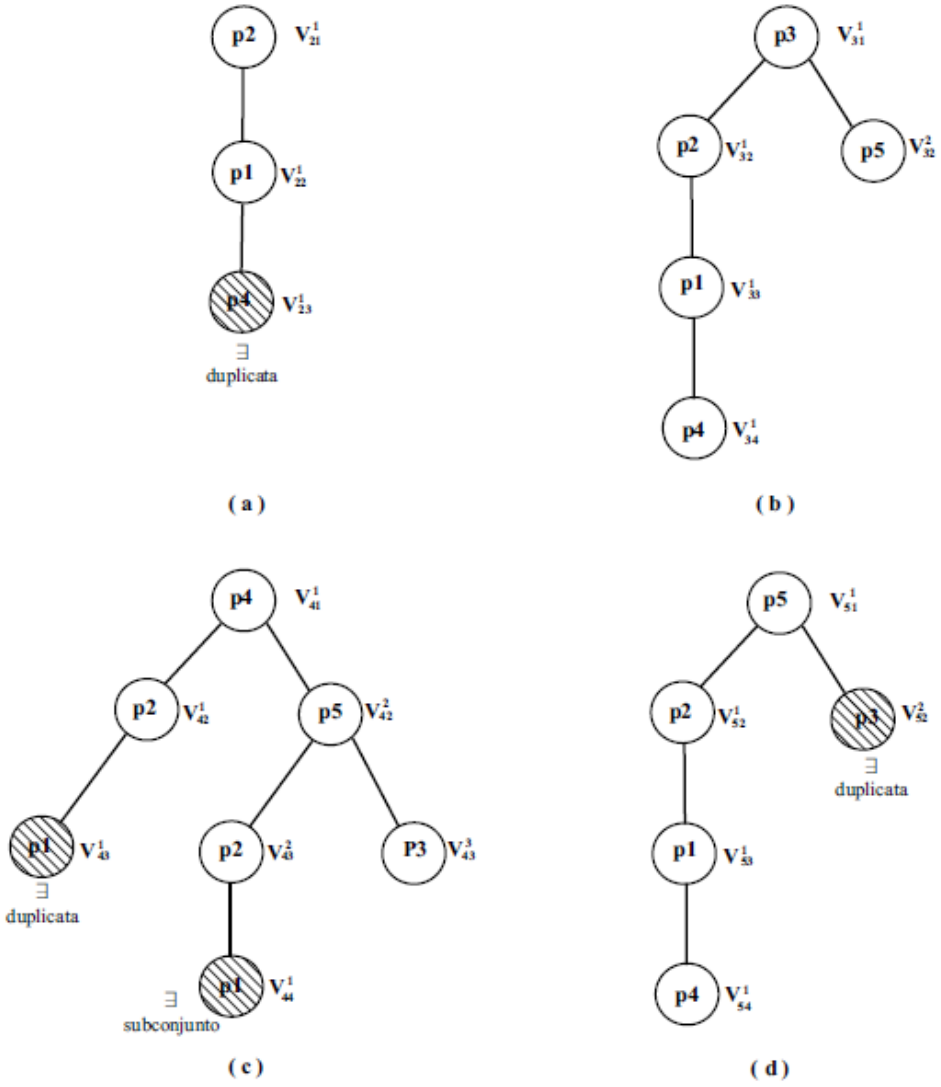

FIGURA 2.10: Relação de lugar-sifão mínimo com relação: (a)p2, (b)p3, (c)p4 e (d)p5

em que I é a matriz de incidência da rede. Dada uma rede de Petri com n lugares e m transições, o objetivo do controle é forçar os processos a obedecerem a restrições da seguinte forma:

$$l_1 M(p_i) + l_2 M(p_j) \leq b_1 \tag{2.10}$$

l_1, l_2 e b_1 são constantes que pertencem ao conjunto de inteiros. A inequação 2.10 pode ser transformada numa igualdade por meio da adição de uma variável de folga:

$$l_1 M(p_i) + l_2 M(p_j) + M_s = b_1. \tag{2.11}$$

A adição do lugar de controle p_s, cuja marcação irá satisfazer a condição de igualdade, é representada por meio da variável de folga. É importante destacar que os lugares

de controle impostos pelo método dos invariantes de lugar são lugares idênticos aos lugares da rede de Petri clássica, e não lugares de controle como os das redes de Petri controladas.

Com a adição do lugar de controle, a matriz de incidência original, de ordem n x m, será acrescida de uma linha, devido à adição da variável de folga. Esta matriz modificada, que corresponde à matriz do controlador, é denominada I_C. Esta matriz contém os arcos que conectam o lugar de controle às transições dos processos controlados da rede. Usualmente, na representação de I_C, omitem-se as linhas da matriz de incidência original.

Em [Moody e Antsaklis 1998] é demonstrado que a síntese de controladores pode ser realizada por meio das seguintes equações:

$$I_C = -L.I \tag{2.12}$$

$$M_{C0} = \alpha - 1 \tag{2.13}$$

onde L é uma matriz inteira n_c x n e α é a quantidade de fichas na marcação inicial do sifão. O elemento (i,j) da matriz L é igual a 1, se o lugar for submetido a uma restrição de controle, e igual a 0 caso contrário. A quantidade de fichas do controlador é dada em função da quantidade de fichas da marcação inicial do sifão menos um.

Para ilustrar o método apresentado, considere a Figura 2.11 :

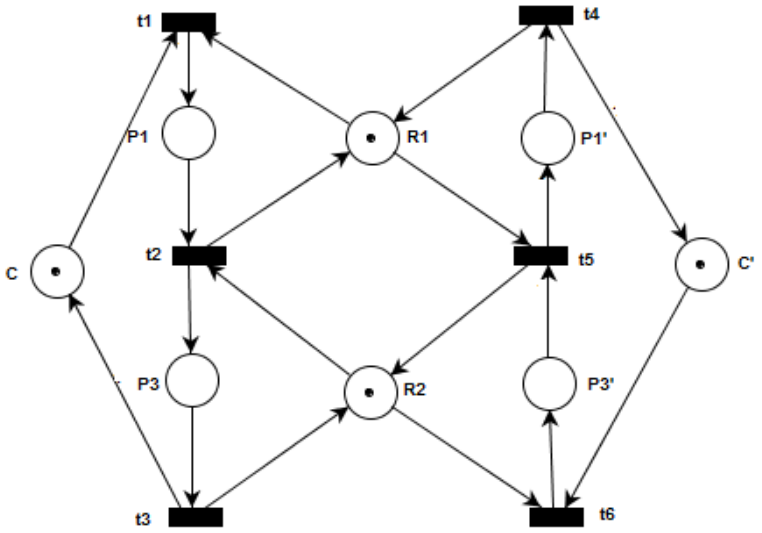

FIGURA 2.11: Exemplo de aplicação do método dos invariantes de lugar.

Executando a rede apresentada na Figura 2.11 temos uma situação de *deadlock* caracterizada pelo disparo sequencial das transições: *t1* e *t6*. Esta situação pode ser vista

na Figura 2.12. Observa-se no momento após a execução desta sequência de disparo, que não há nenhuma outra transição habilitada.

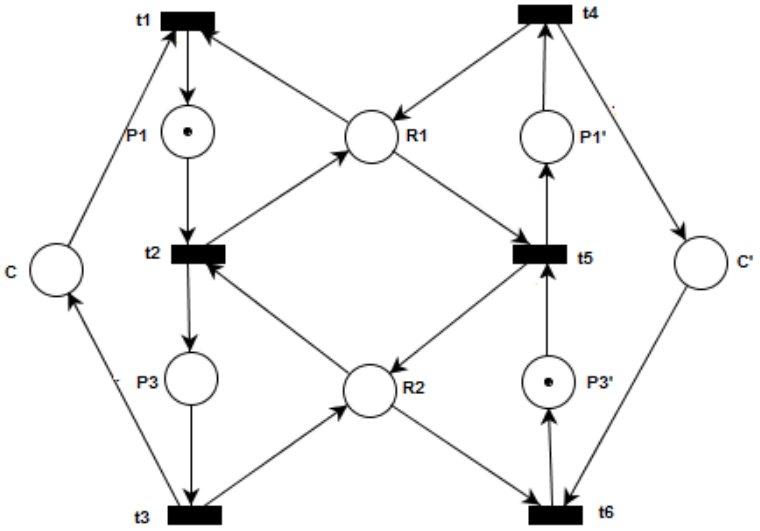

FIGURA 2.12: Rede da Figura 2.11 após o disparo de *t1* e *t6*.

Aplicando o método apresentado na Subseção 2.4.1, os seguintes sifões foram encontrados: *S1* = {*P3, R2, P3'*}, *S2* = {*P1, R1, P1'*}, *S3* = {*P1', P3', C'*} e *S4* = {*P3, R1, R2, P1'*}. O sifão *S4* = {*P3, R1, R2, P1'*}, destacado na Figura 2.13, pode ser esvaziado caso sejam disparadas as transições *t1* e *t6* em sequência.

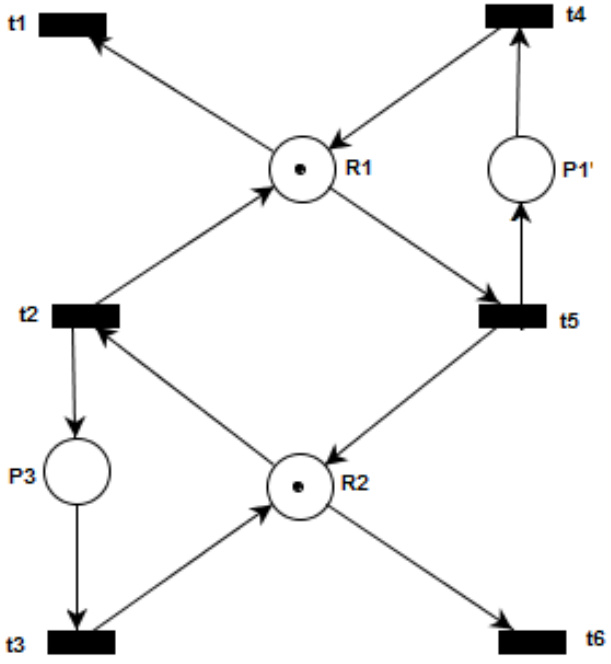

FIGURA 2.13: Sifão S4.

Sabe-se que a situação de *deadlock* acontece devido ao disparo sequencial das transições *t1* e *t6* e, portanto, é necessário impor ao sistema que os lugares *P1* e *P3'* não podem

conter fichas simultaneamente. Neste caso, para utilizar o método de invariante de lugar para estabelecer as restrições de controle é necessário fazer uma transformação das transições que devem ser controladas. Esta transformação de uma transição foi proposta em [Moody et al. 1994] e consiste em transformar uma transição t_j em dois elementos: um lugar p'_j e uma transição t'_j. A Figura 2.14 ilustra esta transformação.

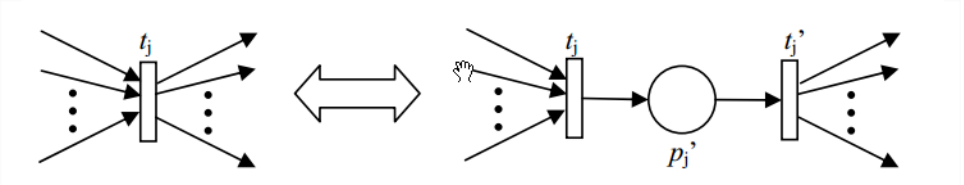

FIGURA 2.14: Transformação de uma transição.

A Figura 2.13 transformada é apresentada na Figura 2.15.

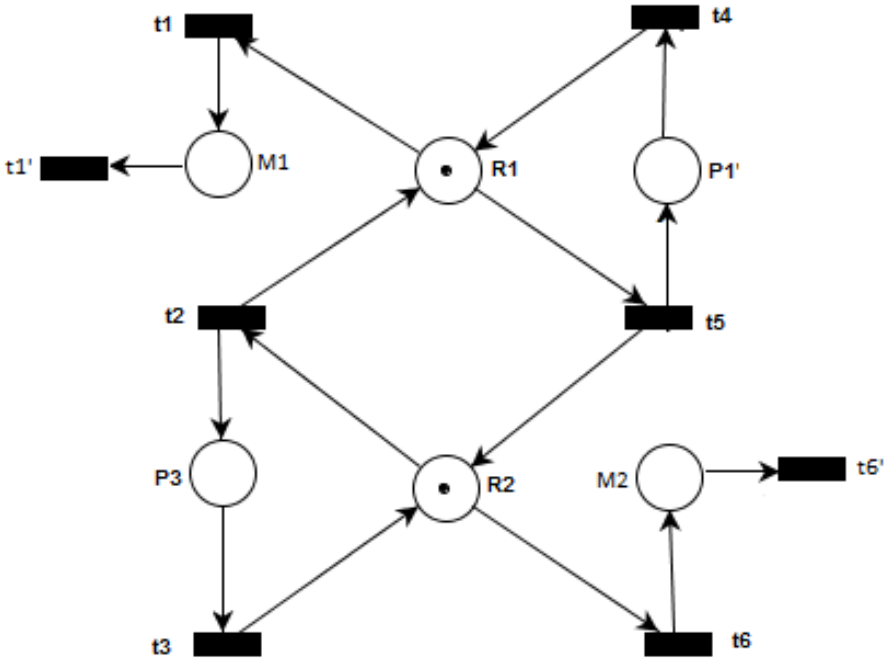

FIGURA 2.15: Sifão modificado.

Da Figura 2.15 pode-se estabelecer a seguinte especificação de controle: $M(M1) + M(M2) \leq 1$.

Dada a especificação acima, para formamos a matriz L, temos que os lugares *M1* e *M2* serão submetidos às restrições de controle, adquirindo valor igual a 1 e os demais lugares terão valor igual a 0. Tendo como sequência de lugares: P3, R1, R2, P1', M1 e M2, podemos especificar a matriz

$$\mathrm{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

A matriz de incidência da Figura 2.15 é dada por:

$$I = \begin{bmatrix} 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Aplicando as equações 2.12 e 2.13:

$$I_C = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix} . \begin{bmatrix} 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$I_C = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

$$e \quad M_{C0} = 2 - 1 = 1$$

Sabe-se que na matriz I_C os lugares com valor igual a 1 são transições de onde irão sair os arcos de entrada do lugar de controle e os lugares com valores iguais -1 são as transições que terão os arcos que sairão do lugar de controle. Isto significa que, com a matriz $I_C = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$, um lugar de controle *CP* será acrescentado ao Sifão *S4* não controlado e o mesmo será lugar de entrada das transições *t1 e t6*, e lugar de saída das transições *t1' e t6'*. A marcação inicial desse lugar conterá uma ficha. O lugar de controle *CP* é ilustrado na Figura 2.16.

Com a inclusão do lugar de controle *CP* percebe-se que não é possível disparar sequencialmente as transições *t1* e *t6*. Segundo [Moody et al. 1994], após a imposição de controle, pode-se retornar ao modelo inicial. Desta forma, o lugar de controle inserido ao modelo inicial é apresentado na Figura 2.17. Como as transições *t1'* e *t6'* não fazem parte do modelo inicial, elas serão substituídas por transições semelhantes no modelo completo. Assim, a transição *t1'* se equivale à transição *t2* e a transição *t6'* se equivale à transição *t5*.

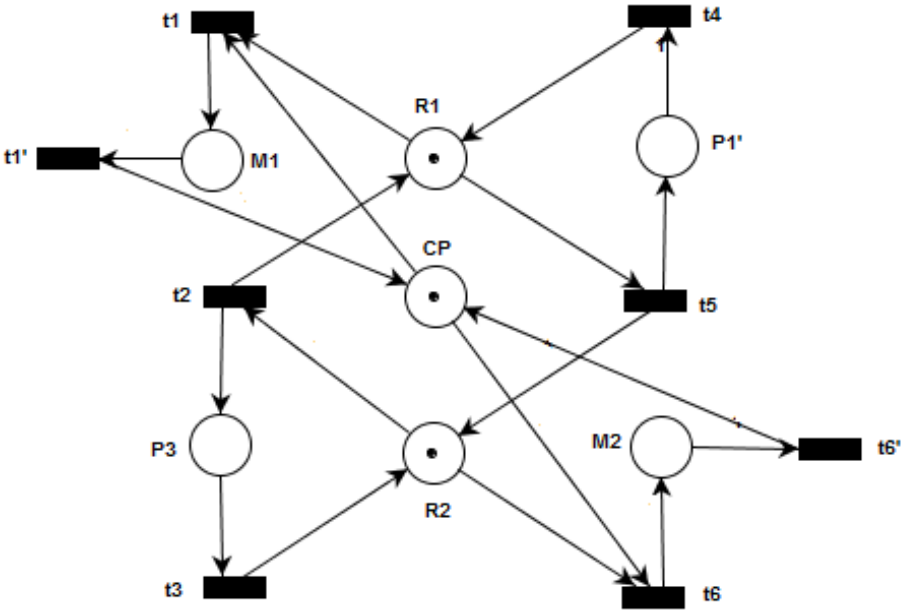

FIGURA 2.16: Sifão *S4* com a inclusão do lugar de controle *CP*.

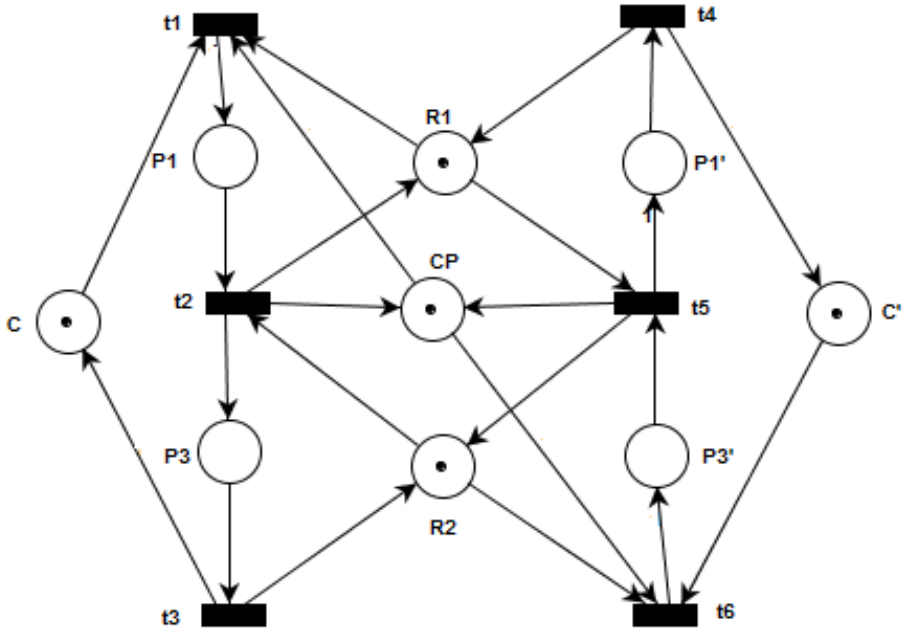

FIGURA 2.17: Rede de Petri da Figura 2.11 com lugar de controle *CP*.

Após a inserção do lugar de controle na rede, percebe-se que o lugar de controle impõe a restrição de disparo sequencial das transições: *t1* e *t6*. Por exemplo, inicialmente temos as transições *t1* e *t6* habilitadas para disparo, se for efetuado o disparo da transição *t1*,

uma ficha será consumida do lugar *R1*, outra de *C* e uma outra do lugar de controle *CP*, e consequentemente, uma ficha será formada no lugar *P1*. Neste momento temos habilitada apenas a transição *t2*, pois o lugar *CP*, que também é entrada para o disparo da transição *t6* não contém fichas. O mesmo acontece se fizermos inicialmente o disparo da transição *t6*, quando uma ficha será consumida do lugar *R2*, outra de *C'* e uma outra do lugar de controle *CP*, para formar uma ficha em *P3'*. Neste momento, apenas a transição *t5* está habilitada ao disparo. Podemos concluir que o lugar de controle *CP* conseguiu, de forma efetiva, estabelecer as restrições de controle propostas e controlar a situação de travamento que existia inicialmente na rede.

2.6 Arquitetura de Software

Com a evolução e experiências adquiridas no desenvolvimento de software, as instituições que trabalham nesta área têm-se preocupado mais com as questões de qualidade, definindo processos de desenvolvimento, incluindo técnicas e ferramentas de modelagem, destacando a importância da documentação como registro de decisões e definições adotadas e não como atividade burocrática que gera um sentimento de perda de tempo.

Por meio do aumento do tamanho e complexidade de sistemas de software, os problemas do projeto vão além das estruturas de dados e algoritmos. Portanto, se faz necessário o uso de uma disciplina que auxilie na obtenção de resultados de baixo custo e maior qualidade. Neste contexto, a Arquitetura de Software tem-se mostrado eficiente para lidar com sistemas grandes e complexos.

Projetar a estrutura geral de sistemas cada vez mais complexos tem sido um problema [Garlan 2000]. Questões que envolvem organização e estrutura geral de controle, protocolos de comunicação e sincronização, atribuição de funcionalidade a componentes de projeto, escalabilidade e desempenho, seleção de alternativas de projeto, compreendem o projeto de software em nível arquitetural.

Os princípios de Arquitetura de Software vêm sendo aplicados junto à Engenharia de Software desde a década de 80. A ideia central da Arquitetura de Software está na redução da complexidade por meio da abstração e separação de interesses. Assim, pode-se conceituar Arquitetura de Software como uma estrutura ou estruturas de um sistema, composta de elementos arquiteturais dos quais os sistemas serão construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões [Pfleeger e Atlee 2009].

Em [Garlan 2000] é sugerido que a arquitetura de software é um nível de design que vai: "além dos algoritmos e das estruturas de dados da computação. A projeção e a especificação da estrutura geral do sistema emergem como um novo tipo de problema. As questões estruturais incluem organização total e estrutura de controle global; protocolos de comunicação, sincronização e acesso a dados; atribuição de funcionalidade a elementos de design; distribuição física; composição de elementos de design; escalonamento e desempenho; e seleção entre as alternativas de design."

A Arquitetura de Software é a base de organização de um sistema, permitindo seu entendimento em termos de componentes, inter-relacionamentos e propriedades consistentes ao longo do tempo e desenvolvimento. Projetar a arquitetura de um sistema pode fornecer vantagens como aumento de produtividade, alinhamento com área de negócio do sistema, facilidades no gerenciamento da tecnologia utilizada e, dentre outros, apoio ao reuso de software em grande escala. Além disso, as decisões de projeto de uma arquitetura podem ter efeito sobre se o sistema pode ou não atender requisitos importantes como desempenho, confiabilidade e manutenibilidade [Bass et al. 2012].

Existem diferentes abordagens de arquitetura de software destacando aspectos importantes da aplicação modelada e que acabam se tornando padrões de organização de sistemas. São os estilos arquiteturais que definem um padrão estrutural a ser aplicado em um sistema, compreendendo os componentes e tipos de conexões entre eles, um conjunto de restrições e formas de como esses componentes podem ser combinados.

A escolha de um padrão arquitetural deve ser dirigida pelas características do sistema a ser desenvolvido e, especialmente, pelos requisitos não funcionais, tais como confiabilidade e segurança. Os padrões podem ser combinados entre si para suportar os requisitos necessários e apoiar a definição de uma arquitetura mais apropriada para o problema. Pode-se destacar dentre os vários padrões de arquitetura de software: a arquitetura hierárquica, a arquitetura centralizada, arquitetura baseada em componentes e a arquitetura distribuída.

2.6.1 Arquitetura Hierárquica

A arquitetura hierárquica ou em camadas [Medeiros 1997] tem como objetivo dividir a organização do sistema em níveis. Baseia-se na decomposição do sistema em componentes que, de acordo com o detalhamento de suas funções, vão pertencer a um determinado nível de abstração. Cada um dos componentes pode interagir com seu nível superior e inferior. O fluxo de controle (solicitação de operações) é *top-down*, ou seja, realizado de forma descendente, partindo dos níveis superiores para os inferiores.

O fluxo de dados contendo o resultado das operações é *bottom-up*, realizado de forma ascendente, dos níveis inferiores para os superiores. Os níveis mais baixos, em que quase não existem abstrações, caracterizam-se por pouca dependência de informação externa, enquanto que os níveis superiores são muito dependentes de dados externos representando os acionadores da dinâmica do sistema.

Em [Albus et al. 1987] temos um exemplo deste tipo arquitetura, desenvolvida por *Albus* e colaboradores da Divisão de Sistemas em Robótica do *National Institute of Standards and Technology (NIST)*, conhecida como NASREM - NASA *Standard Reference Model*, apresentada na Figura 2.18.

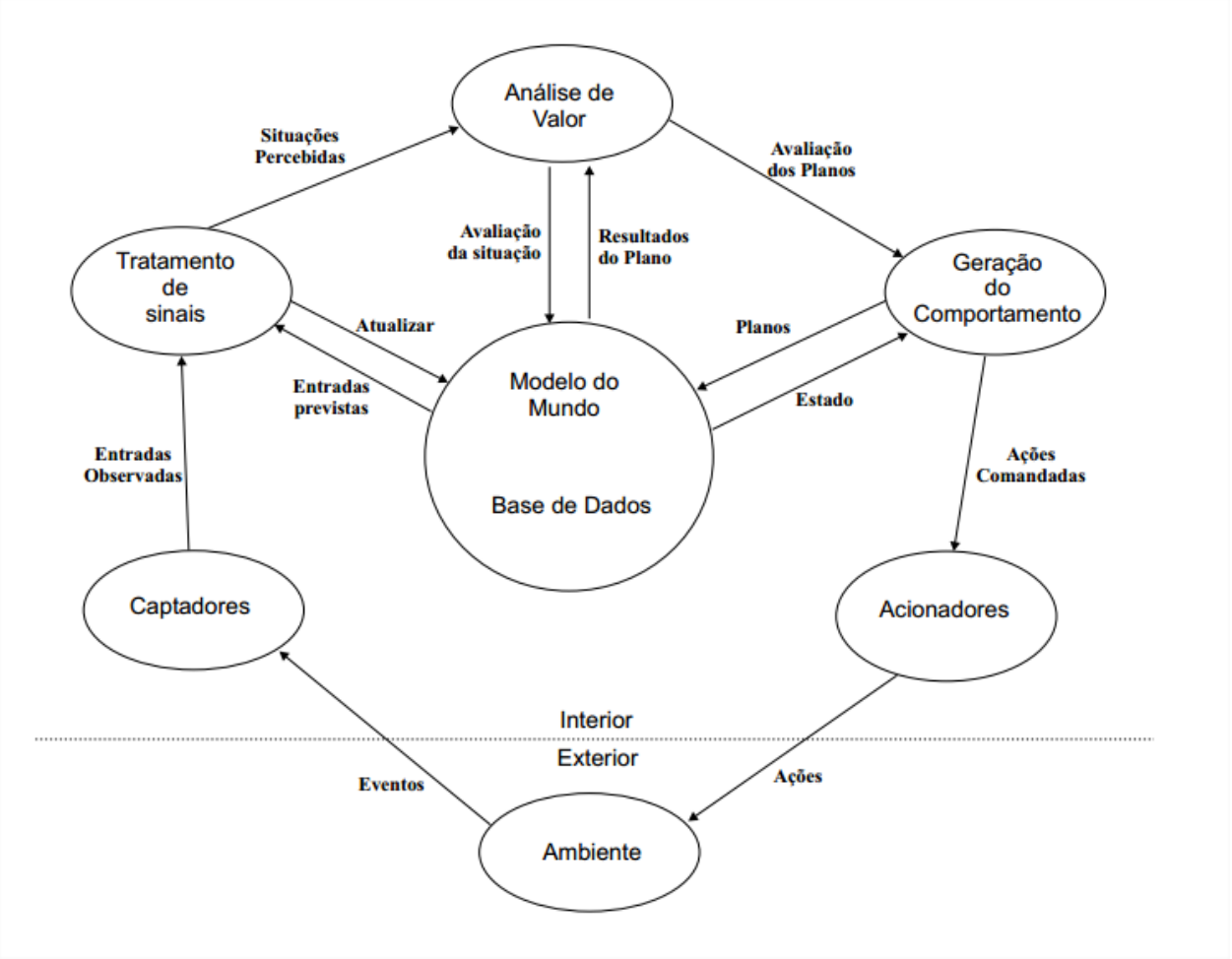

FIGURA 2.18: Exemplo de Arquitetura Hierárquica - NASREM

O modelo teórico proposto por [Albus et al. 1987] consiste em seis elementos básicos (atuadores, sensores, tratamento de sinais, modelo de mundo, geração de comportamento e análise de valor) integrados em uma padrão de sistema hierarquizado que adota uma arquitetura estrita para decomposição de tarefas, da percepção e da modelagem do mundo. A decomposição hierárquica divide as tarefas complexas em subproblemas mais simples e maleáveis.

Dentre as vantagens desta arquitetura pode-se destacar a segurança oferecida devido a possibilidade de abstração, assimilação e fusão de informações divididas em níveis. Em contrapartida, pode-se citar o fato de que se houver alguma mudança em funcionalidades, grande parte dos componentes precisam ser redefinidos, além do mais, a especificação de cada módulo depende muito de suas entradas e saída, fazendo com que os módulos sejam definidos somente após uma especificação estável das interfaces.

2.6.2 Arquitetura Centralizada

No padrão centralizado os módulos do sistema que são baseados nesta arquitetura utilizam um núcleo central para a coordenação das interações entre componentes independentes. Em [Hayes-Roth 1985] o padrão de arquitetura centralizado é destacado por algumas características básicas, tais como, os elementos da solução que são gerados durante a solução dos problemas são gravados em uma base de dados global e estruturada denominando "Blackboard"; todo conhecimento necessário para solucionar um problema é particionado em células separadas e independentes conhecidas como "Bases do Conhecimento"; e para cada ciclo de resolução de um problema é escolhida arbitrariamente uma base de conhecimento dentre as possíveis para executar sua ação.

Um exemplo de aplicação que usa arquitetura centralizada é a TCA (*Task Control Architecture)* [Simmons 1994] empregada no controle de um robô de seis pernas "AMBLER (*Autonomous Orthogonal Legged Walking Robot*)". Nessa arquitetura, mostrada na Figura 2.19, o controle é centralizado, mas os dados são distribuídos. Cada agente possui autonomia para a tomada de decisões parciais e resolução de problemas locais. Os componentes comunicam-se por meio do envio de mensagens que passam pelo controle central. Tais mensagens podem ser dados resultantes de operações requisitadas ou pedidos de operações.

A arquitetura centralizada pode ser considerada mais flexível que outras abordagens devido à sua organização não hierárquica [Tigli et al. 1993]. A combinação de uma sintaxe bem definida, fornece uma orientação para a concepção e o conjunto dos módulos deliberativos e reativos possibilitam uma grande adaptabilidade do sistema,

Uma vantagem desta arquitetura, além da flexibilidade, é a possibilidade de execução de módulos individuais em paralelo. Entretanto, apresenta limitação devido a sobrecarga de informações tratadas no núcleo central [Julia e Soares 2003].

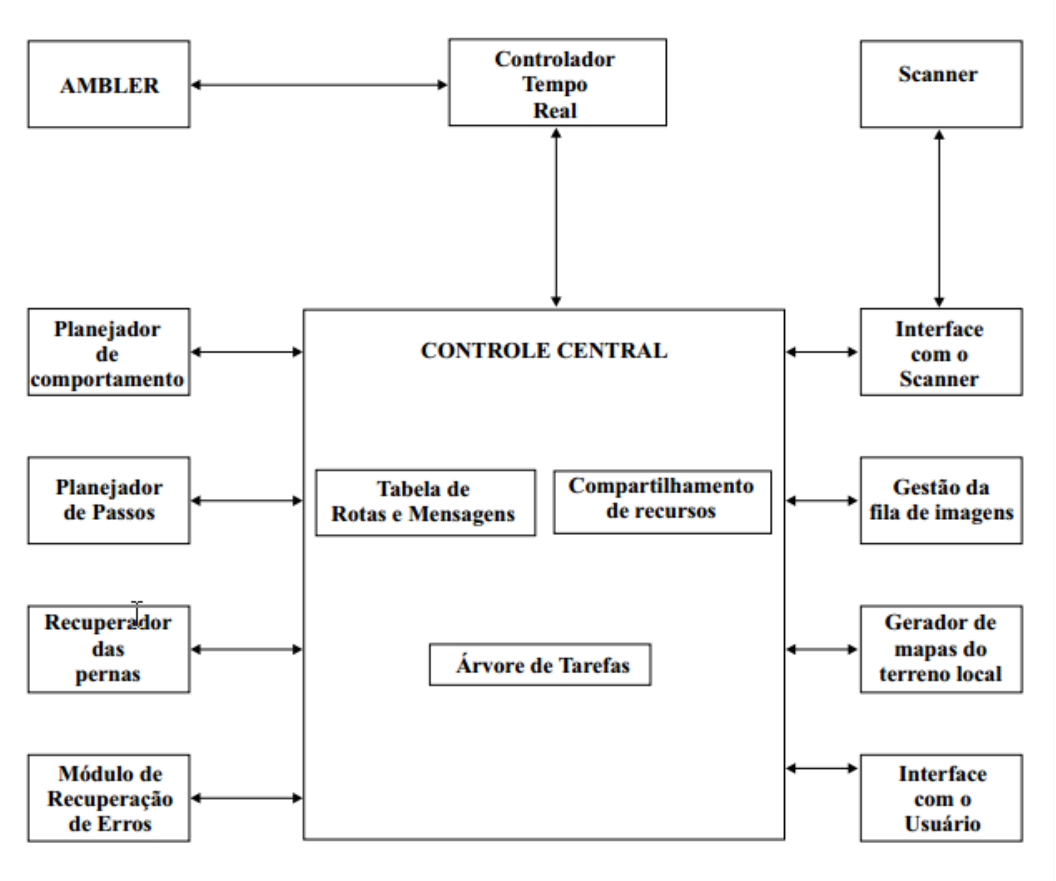

FIGURA 2.19: Exemplo de Arquitetura Centralizada

2.6.3 Arquitetura Baseada em Componentes

A arquitura baseada em componentes baseia-se na decomposição da estrutura do sistema em componentes funcionais e lógicos com interfaces bem definidas, utilizadas para a comunicação entre os componentes. Do ponto de vista da organização dos componentes, alguns autores [Jacobson 1997] [Szyperski 2002] [Brown 2000] [Cai et al. 2000] sugerem que uma arquitetura de componentes deve ser organizada em diferentes camadas, em que os componentes possuem diferentes nívies de abstração, e as com camadas inferiores prestando serviços as superiores, semelhante a estrutura em camadas.

Em [Brown 2000] é apresentada uma proposta para a organização de arquiteturas de componentes, conforme mostra a Figura 2.20. Segundo [Brown 2000], a primeira camada compreende os componentes de negócio, que implementam um conceito ou processo de negócio autônomo. Estes componentes são artefatos necessários para representar, implementar e implantar um conceito. A camada intermediária contém os componentes utilitários, que prestam serviços básicos necessários ao desenvolvimento de aplicações. Componentes utilitários não pertencem a nenhum domínio específico, mas

são utilizados por inúmeras aplicações de negócio, por exemplo, calendários e formulários genéricos. A última camada envolve os componentes de infraestrutura, que são responsáveis por estabelecer a comunicação com as plataformas de execução do software. Estes componentes estão vinculados às questões de tecnologia de desenvolvimento dos componentes das camadas superiores. Dentre os serviços providos por componentes de infraestrutura, estão: comunicação entre componentes distribuídos, persistência, tratamento de exceções e portabilidade.

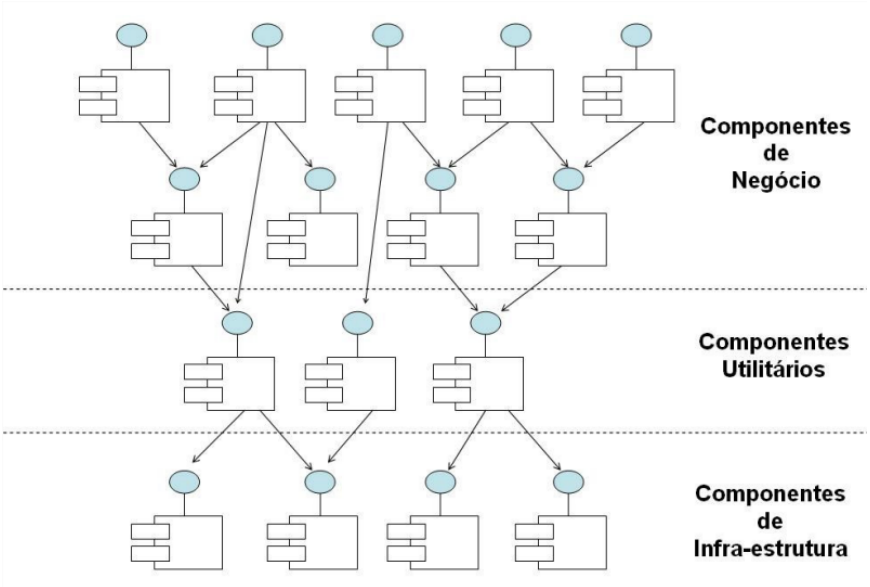

FIGURA 2.20: Exemplo de Arquitetura Baseada em Componentes

A principal vantagem da arquitetura baseada em componentes é o alto nível de modularização, reduzindo significativamente os custos de manutenção e evolução. Além disso, novas funcionalidades podem ser facilmente adicionadas por meio da extensão de componentes pré-existentes ou da criação de novos componentes.

2.6.4 Arquitetura Distribuída

O padrão distribuído surgiu pela necessidade de atender a demanda atual de sistemas com base em grandes computadores distribuídos [Tanenbaum 2010]. Um sistema distribuído é um sistema em que o processamento de informações é distribuído entre vários computadores ao invés de ficar confinado numa única máquina. Em [Tanenbaum 2010], os sistemas distribuídos são caracterizados como "uma coleção de computadores independentes vistos pelos usuários do sistema como um único computador". Esta definição pode ser vista sob dois aspectos: hardware, no qual as máquinas são autônomas, e software, no qual o usuário entende o sistema como sendo um único computador.

Os sistemas distribuídos possuem as seguintes características:

- os componentes do sistema se comunicam através de mensagens (não existem variáveis globais);
- Concorrência: o sistema pode ser utilizado simultaneamente por vários usuários, porém há uma necessidade de coordenação de acesso aos recursos compartilhados como hardware, software e dados;
- Compartilhamento de recursos: um sistema distribuído permite o compartilhamento de recursos de hardware, software e dados que estão associados à diferentes computadores;
- são assíncronos, permitem diferentes velocidades de processamento e não existe limitação para tempo de comunicação;
- as falhas são independentes, ou seja, a falha de um componente não impede necessariamente os outros de funcionarem;
- Heterogeneidade: um sistema distribuído pode ter diferentes tipos de rede, de hardware, de sistemas operacionais e linguagens de programação.;
- permitem ampliação com adição de novos recursos, por isto é considerado um sistema aberto;
- um usuário pode ter acesso transparente aos recursos sem saber da natureza distribuída do sistema.

Como vantagens do uso de sistemas distribuídos com relação aos centralizados, além do fator econômico, uma vez que um conjunto de computadores de menor porte tem melhor relação custo/desempenho que grandes computadores, podemos apontar:

- a capacidade de processamento que pode ser maior em sistemas distribuídos ou mesmo gradativamente aumentada;
- a maior tolerância a falhas, pois se uma máquina falha o sistema como um todo ainda pode continuar trabalhando normalmente;
- a distribuição das máquinas propriamente dita, isto é, algumas aplicações podem envolver máquinas localizadas longe uma da outra.

Entretanto, este padrão apresenta desvantagens como a complexidade maior comparada a outros padrões, a segurança comprometida por intercepções no tráfego da rede, a dificuldade de gerenciamento pela quantidade de máquinas na rede e a imprevisibilidade do tempo de resposta de um pedido que depende do tráfego do sistema e da carga da rede.

Para projetar sistemas distribuídos há diferentes abordagens arquiteturais que devem ser selecionadas de acordo com as necessidades de hardware e software desejáveis em cada sistema.

A arquitetura multiprocessadores [Sommerville 2010] é o modelo mais simples de um sistema distribuído. É um sistema multiprocessador que consiste numa série de diferentes processos que podem, mas não necessariamente, ser executados em computadores separados. A distribuição de processos para os processadores pode ser predeterminada ou ficar sob o controle de um escalonador que decide qual processo alocar para cada processador.

A arquitetura cliente-servidor [Sommerville 2010] é o modelo de arquitetura que mostra como os dados e o processamento são distribuídos em uma série de processadores. Compõe este modelo um conjunto de servidores que oferecem serviços a outros subsistemas, um conjunto de clientes que solicitam serviços oferecidos pelos servidores e uma rede que permite o acesso aos serviços.

Já a arquitetura de objetos distribuídos [Sommerville 2010] é uma abordagem que elimina a distinção entre cliente e servidor, e torna o projeto como uma arquitetura de objetos distribuídos. Estes objetos são os componentes fundamentais do sistema e oferecem uma interface para um conjunto de serviços fornecidos por eles. Os objetos podem ser distribuídos em uma série de computadores de uma rede e podem se comunicar.

Como exemplo de uma arquitetura de objetos distribuídos temos o *Object Request Broker (ORB)* que trabalha analogamente ao barramento, permitindo que várias placas sejam conectadas a ele e realizando a comunicação entre os diversos dispositivos de hardware. O *ORB* faz a comunicação entre os objetos como se fosse um barramento de software.

Capítulo 3

Literatura Correlata

A ideia inicial de suporte a processos de negócio é relatada na década de 70, quando o primeiro protótipo de um sistema de *workflow* foi desenvolvido por *Zisman* em sua tese de doutorado. De acordo com [Desel e Erwin 2000], os processos de negócio e seus projetos ganharam importância desde o início dos anos 1990. A habilidade de simplificar um processo desse tipo, de forma eficiente e flexível, é o fator de sucesso mais difícil de ser alcançado pelas organizações atualmente. Tal dificuldade criou a necessidade de desenvolver e lidar com técnicas e ferramentas adequadas para identificar, analisar e simular processos de negócio.

O termo *workflow* de acordo com [Aalst e Hee 2002] é utilizado na bibliografia com o significado de "processo de negócio". Em [Leymann e Roller 1997] é definido um *workflow* como um conjunto de atividades que podem, ou não, ser executadas simultaneamente e possuem, entre as atividades do negócio, fluxo de dados e algumas especificações de controle. Segundo [Aalst e Hee 2002], o processo de *workflow* representa a sequência de atividades que precisam ser executadas em uma organização para tratar casos específicos e atingir uma meta previamente estabelecida.

De acordo com [Hofstede et al. 2009], a área de Gerenciamento de Processos de Negócio, do inglês *Business Process Management* (BPM), tem recebido uma atenção considerável nos últimos anos em razão do seu potencial em aumentar significativamente a produtividade e economizar gastos. *Hofstede* destaca que o conceito de processo é fundamental e serve como um ponto de partida para o entendimento de como o negócio opera e quais oportunidades existem para coordenar suas atividades constituintes. Para [Aalst 1998], a meta fundamental do gerenciamento de *workflow* é ter certeza de que atividades corretas são executadas pelas pessoas certas no tempo certo.

O termo Gerenciamento de *Workflow* (*Workflow Management*) refere-se a ideias, métodos, técnicas e softwares usados para apoiar processos de negócios estruturados. O fluxo e a distribuição de trabalho devem ser controlados pelos Sistemas de Gerenciamento de *Workflow* (*WFMSs − Workflow Management Systems*). Em geral, os *WFMS* predefinem a sequência em que as atividades serão executadas, determinam qual participante executará cada atividade e acompanham a situação de uma determinada instância de *workflow*. É responsabilidade do *WFMS* verificar se as atividades são executadas na sequência correta e notificar sobre a execução das tarefas. Outros termos para caracterizar Sistemas de Gerenciamento de *Workflow* são: sistemas de operações de negócio, gerenciamento de *workflow*, gerenciamento de casos (*case manager*) e sistema de controle logístico [Aalst e Hee 2002].

Apesar de existirem várias ferramentas para Sistemas de Gerenciamento de *Workflow*, poucas destas ferramentas dispõem de mecanismos de análise. A verificação de processos de negócio não é tão simples do ponto de vista computacional. Como consequência, poucos *workflows* que são checados completamente antes de serem colocados em execução, o que frequentemente resulta em erros que serão corrigidos com custos mais altos [Aalst 1998].

No mercado atual destaca-se também a cooperação entre as organizações para, juntas, atingirem alguns objetivos comuns de negócio. Isto significa que o gerenciamento de *workflow* deve ser capaz de lidar com processos de *workflow* que abrangem múltiplas organizações. Os Sistemas de Gerenciamento de *Workflows* Interorganizacionais (*WFMS-I*) [Fantinato et al. 2005] são extensões dos sistemas de gerenciamento de *workflows* tradicionais. O gerenciamento dos *workflows* interorganizacionais precisa ser realizado com cooperações transparentes entre múltiplos *workflows* locais que podem executar de forma independente as suas atividades, porém precisam ser sincronizados em certos pontos para garantir a correta execução do processo do negócio global.

Alguns trabalhos, como o desenvolvido por [Aalst e Hee 2002] consideram a teoria das redes de Petri como uma ferramenta eficiente na modelagem e análise de processos de negócio. Isto é evidenciado por características das redes de Petri como: a semântica formal, a natureza gráfica, expressividade, boas propriedades e técnicas de análise. Em [Aalst e Hee 2002] destaca-se ainda que o uso de tal formalismo tem uma série de vantagens importantes como o fato de forçar uma definição precisa dos processos. Características como ambiguidade, incerteza e contradições são prevenidas, em contraste com a maioria das técnicas de diagramação informais.

De acordo com [Aalst e Hee 2002], as redes de Petri têm sido amplamente estudadas e aplicadas com sucesso na área de sistemas dinâmicos de eventos discretos, que são

caracterizados por paralelismo e sincronização. Os incentivos que levam as pesquisas nessa área são a forte fundamentação matemática e a disponibilidade de ferramentas para análise.

Em [Oberweis et al. 1997] os autores identificam outros motivos para optar pelo uso de redes de Petri no contexto de *workflow* como, por exemplo, a integração de dados e aspectos comportamentais, o apoio concorrente e corporativo, disponibilidade de técnicas e ferramentas de análise e a flexibilidade. Resumindo, a principal vantagem de empregar redes de Petri na modelagem de *workflow* é a combinação de fundamentação matemática, representação gráfica compreensiva e possibilidade de simulações e verificações [Merz et al. 1995].

Em [Aalst e Hee 2002] os autores definem uma rede de Petri que modela o processo de *workflow* como uma *WorkFlow net*. No contexto de Sistemas de Gerenciamento de *Workflow*, um dos problemas que podem surgir são os *deadlocks* que são uma das causas que inviabilizam as boas propriedades que os modelos baseados em *WorkFlow net* devem ter. Em [Awad 2008] um *deadlock* em um modelo de processo é dado se uma determinada instância do modelo (não necessariamente todas) não pode continuar sendo executada, sendo que esta instância ainda não chegou ao fim de sua execução.

O risco de situações de *deadlock* aumenta proporcionalmente ao aumento de subprocessos que comunicam e do conhecimento parcial das estruturas dos processos que colaboram. Assim, *workflows* interorganizacionais estão mais suscetíveis à situações de *deadlock*. Em [Shim et al. 2002] os autores fazem referência à dificuldade de detectar situações de *deadlock* em processos de *workflow* interorganizacional em tempo de execução. Por isso, é desejável detectar e remover o *deadlock* na definição do processo de *workflow* antes da execução real do processo correspondente.

Em um *workflow* interorganizacional, várias tarefas podem ser ativadas para iniciarem juntas e podem trocar mensagens. Nessa situação, uma tarefa pode ser bloqueada por algumas razões, como, por exemplo, o não recebimento de uma mensagem. Em [Shim et al. 2002] define-se que uma instância de um processo está em *deadlock* de comunicação se, e somente se, nesta instância, excetuando-se as tarefas concluídas, todas as tarefas estão bloqueadas ou em estado de espera (*sleep*) e se não há nenhuma mensagem em trânsito entre as tarefas. Uma tarefa está em *deadlock* se esta faz parte da instância do processo que está em *deadlock*.

Um *workflow* interorganizacional [Aalst 1998] é essencialmente um conjunto de processos de *workflow* de n parceiros de negócio que estão envolvidos em um único *workflow* global. Cada um dos parceiros tem completo controle de seu próprio *workflow* local.

Todavia, estes *workflows* locais precisam se comunicar por dependerem uns dos outros para a sua correta execução. Existem dois tipos de comunicação: assíncrona e síncrona. Comunicação assíncrona corresponde à troca de mensagens entre *workflows* locais e a comunicação síncrona força o *workflow* requisitando a passar o controle para o *workflow* chamado.

A verificação da propriedade *Soundness* em *WorkFlow nets* interorganizacionais de acordo com [Aalst 1998] é feita realizando a verificação em cada *WorkFlow net* local. De acordo com Aalst, um *workflow* interorganizacional é localmente *sound* se, e somente se, cada uma uma das *WorkFlow nets* isoladas for *sound*. Aalst destaca também que a única solução, caso a propriedade *Soundness* não seja verificada, é refazer os modelos até que esta propriedade seja respeitada.

Em [Aalst e Hee 2002] são apresentados também três métodos para provar a propriedade *Soundness*: o primeiro baseado na construção de grafos de alcançabilidade, o segundo composto pela análise das propriedades vivacidade e limitação para uma rede "*short-circuited*" e o terceiro fundamentado na substituição de blocos bem formados. Em [Aalst 1997], por exemplo, o método para provar *Soundness* é baseado na construção de grafos de cobertura que podem consumir muito tempo em sua construção. O autor também propõe uma nova classe de *Workflow nets*, as "*Free-choice Workflow nets*" que permitem determinar a propriedade *Soundness* em tempo polinomial. Em [Aalst 1996], são apresentadas as "*Well-Structured Workflow nets*", que têm um número desejável de propriedades e para as quais a propriedade *Soundness* também pode ser verificada em tempo polinomial. Como uma abordagem alternativa, em [Soares Passos e Julia 2009] é apresentado um método para análise qualitativa de *Workflow nets* baseada na construção de árvores de prova canônica da lógica linear. A análise qualitativa proposta neste trabalho diz respeito à prova do critério de corretude *Soundness* para *Workflow nets*.

Um dos motivos que inviabilizam a propriedade *Soundness* é a existência de situações de *deadlock*, as quais impedem o processo de finalizar sua execução de forma correta, alcançando assim um estado de travamento. Situações de *deadlock* nas redes de Petri estão relacionadas à presença de sifões que, ao ficarem livres de fichas, impedem o disparo das transições da rede levando o sistema a um estado de travamento [Barkaoui e Abdallah 1995b] [Iordache 2003] [Chu e Xie 1997] [Tricas e Martinez 1995]. Controlar estes sifões é uma condição necessária para impedir que os sistemas alcancem estados em que estejam em situações de *deadlock*.

Alguns autores classificam os possíveis tratamentos do problema de *deadlock* dentro de três métodos básicos [Coffman et al. 1971], [Tanenbaum 2010], [Silbertschatz et al. 2004], [Li e Zhou 2008]:

- método de prevenir (*Deadlock Prevention*): consiste em conceber um sistema livre de *deadlock* determinando todos os estados alcançáveis do sistema e, mediante um arranjo estrutural do sistema, não se permite que o mesmo evolua para os estados indesejáveis de travamento. Considerando que o método envolve um processamento para a determinação de todos os estados alcançáveis do sistema, este pode tornar-se computacionalmente inviável devido à explosão de estados decorrentes da quantidade de elementos envolvidos e as suas inter-relações. Porém, requer menos esforço computacional em tempo de execução do controle se os problemas forem identificados e solucionados na fase de projeto do controle [Coffman et al. 1971].

- método de detectar e restabelecer (*Deadlock Detection and Recovery*): consiste na execução de algoritmos que detectam o estado de *deadlock* e executam ações para sair deste estado. Nos sistemas em que há apenas o processamento de informações, a volta ao passado e a eliminação de processos são as ações mais usuais. A complexidade computacional deste método é mais reduzida se comparada aos demais [Coffman et al. 1971].

- método de evitar (*Deadlock Avoidance*): consiste na execução de algoritmos que monitoram os processos quanto à alocação de recursos. A identificação de estados imediatamente anteriores ao travamento dispara ações em tempo real para que o sistema não evolua para o estado de *deadlock*. A grande vantagem deste método é quanto ao uso otimizado de recursos. Entretanto, em alguns casos, é possível gerar um novo estado de *deadlock* decorrente das restrições introduzidas no sistema [Coffman et al. 1971].

É necessário avaliar a natureza das atividades realizadas para identificar as limitações dos métodos existentes e adequar o melhor método ou a melhor composição de métodos. Em [Wang et al. 2009], os autores apresentam uma abordagem de prevenção de *deadlock* baseado em redes de Petri com realização de uma análise parcial do grafo de alcançabilidade da rede de Petri. Os autores classificam os estados em: marcação morta, marcação inadequada e marcação ameaçadora. A marcação inadequada significa que a rede de Petri evoluirá inevitavelmente para um estado de *deadlock* e a marcação ameaçadora poderá alcançar o estado de travamento.

O modelo especial de redes de Petri denominado "Sistemas de Processos Sequenciais Simples com Recursos" (S^3PR) [Abdallah e ElMaraghy 1998] representa a composição de uma sequência de processos independentes e paralelos com uso de recursos comuns e restrições de capacidade. A classe S^3PR é uma subclasse ordinária e conservativa de redes de Petri [Murata 1989], isto é, os pesos dos arcos são constantes iguais a um (1) e a quantidade total de marcas na rede é sempre constante. Um dos métodos apresentados no trabalho de [Ezpeleta et al. 1993] foi o de aplicar o método de prevenção de *deadlock* em sistemas de manufatura flexíveis modelado com a classe de rede de Petri S^3PR. Em [Gang e Ming 2004] também apresentam uma abordagem do problema de deadlock aplicadas à manufatura utilizando a classe S^3PR e aplicando métodos de detecção, prevenção e correção de *deadlock*.

Em [López-Grao e Colom 2006] é apresentado o uso da classe S^4PR para caracterização de situações de *deadlock*. Os autores dizem que a classe S^4PR tem despertado um interesse considerável em razão de um compromisso equilibrado entre a flexibilidade de modelagem e técnicas efetivas de correção. A classe S^4PR é capaz de modelar sistemas em que os processos estão autorizados a decidir entre os caminhos de execução alternativos ao longo de toda sua execução. Além disso, vários recursos de vários tipos podem ser reservados ao mesmo tempo e podem ser adquiridos e liberados em qualquer estado de execução.

O *deadlock* é um assunto bastante estudado quando se refere a sistemas de manufatura. Alguns autores como [Ezpeleta et al. 1993], [Tricas e Ezpeleta 2003], [Li e Zhou 2003], [Li e Zhou 2004], [Li e Zhou 2008], entre outros, buscaram soluções explorando as redes de Petri para encontrar uma maneira de especificar o controle do processo global. Embora os resultados sejam significativamente interessantes, para sistemas de grande porte o envolvimento de grande quantidade de variáveis pode desencadear uma explosão combinatória de estados alcançáveis do sistema. Assim, exigindo um grande esforço computacional que pode inviabilizar a especificação do controle global neste caso.

Outros grupos de pesquisadores como [Lawley e Ferreira 1994], [Santos Filho 1998], [Lawley et al. 1998], [Fanti et al. 2000], [Santos Filho 2000], [Fanti e Turchiano 2002], [Gang e Zhiming 2003], [Wu e Zhou 2003], [Wu e Zhou 2004], [Gang e Ming 2004], [Wu e Zhou 2005], [Wu e Zhou 2007], entre outros, buscaram um controle baseado em restrições em vez de um controle global. Ou seja, o comportamento do sistema é conhecido até um dado estado e a partir deste estado não é possível determinar qual será o novo comportamento do sistema, restringindo-se, portanto, a evolução para que os estados indesejados não sejam alcançados.

Em [Ezpeleta et al. 1993] os autores propuseram um algoritmo para calcular sifões e inserir um controle supervisório para cada sifão identificado na rede de Petri. A maior dificuldade observada nos trabalhos de *Ezpeleta* é o esforço computacional necessário para a determinação de sifões.

São apresentados em [Li e Zhou 2003] os algoritmos de *deadlock prevent* aplicados a redes de Petri baseados no controle de sifões elementares. De acordo com os autores, o controle dos sifões elementares garante que os sifões redundantes também sejam controlados. A classificação de sifões em sifão elementar e redundante foi proposta por [Li e Zhou 2003]. *Li* e *Zhou* apresentam o controle de sifões elementares que permite a redução do número de controladores supervisórios necessários mediante a inserção de apenas um controlador supervisório para cada sifão elementar identificado.

Em [Iordache 2003] foram propostas novas tecnologias para o controle supervisório de sistemas concorrentes. Utilizando as redes de Petri para modelagem destes sistemas concorrentes, *Iordache* aborda em seu trabalho vários problemas de controle supervisório e apresenta procedimentos para prevenção de *deadlock* em redes de Petri mais genéricas que os modelos S^3PR e S^4PR. A abordagem utilizada permite conceber um supervisor independente do estado inicial do sistema, ou seja, o controle do sistema é realizado localmente sem modificar por inteiro o modelo na rede Petri e somente atua nos sifões existentes.

Vários artigos que usam explicitamente o controle supervisório para restringir o comportamento de uma rede de Petri foram encontrados, tais como [Giua e DiCesare 1992] [Moody et al. 1994] [Iordache 2003]. O uso de lugar de controle para controlar sifões também podem ser vistos em trabalhos, como [Barkaoui e Abdallah 1995a] [Ezpeleta et al. 1993] [Lautenbach e Ridder 1996]. Em [Moody et al. 1999] supervisores são criados utilizando a metodologia de invariante de lugar de [Moody e Antsaklis 1998]. Neste trabalho, quando uma rede de Petri é supervisionada de tal forma que sua marcação satisfaça um conjunto de inequações lineares, prova-se que esta rede é livre de *deadlock* para todas as marcações iniciais que satisfaçam as restrições de supervisão. Em [Lautenbach e Ridder 1996] é utilizada uma metodologia de controle onde cada novo sifão mínimo é controlado e a principal característica deste trabalho é que ele trabalha com redes de Petri que não são ordinárias.

Em se tratando de trabalhos com *workflows* interorganizacionais, [Aalst 1998] trata situações de *deadlock* nas redes de Petri interorganizacionais por meio de alterações manuais no modelo, para evitar que o fluxo que leva a tais situações não ocorra, desviando o fluxo para alguma rota que finalize a execução sem a ocorrência da situação

indesejada. Tal abordagem, que não utiliza um procedimento sistemático, faz alterações no modelo para torná-lo *sound* e se torna inviável dependendo do tamanho e complexidade do modelo.

Em trabalhos, como o de [Xiong et al. 2009], é utilizada uma abordagem para encontrar situações de *deadlock* em *Web-services* (processos sequenciais distribuídos), mas não é utilizado um lugar de controle para a prevenção desta situações. Em [Barkaoui et al. 2008], os diferentes processos de *workflow* compartilhados entre vários recursos são controlados por meio dos lugares de controle, mas não no caso interorganizacional, no qual os diferentes processos de *workflow* não comunicam entre si.

Existem várias abordagens quando se trata de controle supervisório modelados por redes de Petri. Em [Holloway e Krogh 1994] as abordagens são divididas em controle por realimentação de estados e controle por realimentação de eventos.

Introduzido por [Ichikawa e Hiraishi 1988], o controle por realimentação de estados foi inicialmente estudado para um modelo denominado rede de Petri controlada. Rede de Petri controlada é uma classe das redes de Petri com condições de habilitação externas, denominadas entradas de controle, que permitem um controlador externo influenciar a progressão das fichas na rede. Nestas redes, os lugares são denominados de lugares de estado, enquanto uma transição é dita controlada caso exista pelo menos um arco ligando esta transição a uma entrada de controle. Nesta abordagem, a intenção de controle é restringir o comportamento para permitir que somente as marcações admissíveis, dadas em um conjunto de marcações admissíveis dos Sistemas a Eventos Discretos a serem controlados.

Em [Yamalidou e Kantor 1991] são utilizados invariantes de lugar para calcular um controlador por realimentação de estados mediante a multiplicação de matrizes. Este método de controle supervisório pode ser aplicado a sistemas cujas restrições são expressas como inequações algébricas ou expressões lógicas que contém elementos dos vetores de marcação ou disparo. Para isto, aproveita-se um resultado apresentado em [Yamalidou e Kantor 1991], onde foi mostrado como restrições escritas como expressões *booleanas* podem ser transformadas em conjuntos de inequações lineares envolvendo vetores de disparo de marcações. Além da eficiência computacional, este método não realiza enumeração de estados durante o processo de síntese. Como consequência, este método de controle é genérico e possui grande potencial para aplicações práticas em Sistemas a Eventos Discretos grandes e complexos.

O conceito de invariantes de lugar é generalizado a uma rede de Petri que possui transições não-controláveis ou não-observáveis, mas não garante a máxima permissividade

para o caso geral [Moody e Antsaklis 1996]. Nesta metodologia, não devem existir arcos conectando os lugares de controle às transições não controláveis, ao mesmo tempo que não devem existir arcos das transições não-observáveis aos lugares de controle. A restrição sobre as transições não observáveis se aplica, pois, como a ocorrência de tais transições não é percebida, elas não podem ser utilizadas por um lugar de controle para atualizar sua respectiva lei. Esta metodologia é apresentada de maneira mais aprofundada por [Moody e Antsaklis 1996].

Em [Moody e Antsaklis 1998], a síntese de supervisores é realizada utilizando a teoria de Sifões. O método proposto parte do princípio que, para evitar situações de *deadlock*, é necessário prevenir que os sifões sejam esvaziados. Assim, uma restrição linear é imposta a cada sifão, garantindo que a quantidade de fichas seja no mínimo unitária. Para determinar um lugar de controle para cada restrição linear, o método baseado nos invariantes de lugar proposto em [Moody e Antsaklis 1996] é utilizado em sequência. Esta abordagem possui uma desvantagem pois a adição de lugares de controle que visa prevenir o esvaziamento de fichas do sifão pode induzir a novas situações de *deadlock* [Ezpeleta et al. 1995].

O trabalho [Iordache e Antsaklis 2002] generaliza os resultados de [Yamalidou e Kantor 1991] e [Moody e Antsaklis 1996], ao incluir o vetor de *Parikh* como um novo termo no conjunto das restrições a serem satisfeitas. De maneira geral, este vetor conta com qual frequência uma dada transição foi disparada desde a inicialização do sistema, restringindo a diferença entre o número de disparo de duas transições, para que sejam elas, no máximo, unitárias. De acordo com *Iordache* (2002), quando uma rede de Petri é supervisionada de tal forma que sua marcação satisfaça um conjunto de inequações lineares, prova-se que esta rede é livre de *deadlock* para todas as marcações iniciais que satisfaçam as restrições de supervisão. Embora o procedimento de síntese seja computacionalmente custoso, o supervisor resultante requer poucos recursos computacionais em tempo de execução.

3.1 Considerações Finais do Capítulo

Atualmente, sistemas de *workflow* se encontram cada vez mais em evidência, em especial sistemas onde há colaboração entre mais de um processo de negócio (*workflows* interorganizacionais). Sistemas que possuem um *workflow* bem elaborado e ausente de situações indesejadas, também podem oferecer padronização, maior qualidade e agilidade na execução e, como consequência, tem-se um sistema mais eficiente e eficaz na execução das atividades.

Modelar um sistema de *workflow* consiste em representar, por meio de um formalismo, atividades que compõem os processos, as sequências de execução destes processos e seus inter-relacionamentos, bem como os recursos responsáveis pela execução e demais agentes que o influenciam. As redes de Petri permitem esta modelagem formal e a grande vantagem de possibilitar a verificação mediante um rastreamento minucioso de cada etapa do *workflow*.

Por fim, os trabalhos relacionados permitiram constatar que a especificação de um novo procedimento de controle supervisório era necessária para detectar e prevenir situações de *deadlock* nos *workflows* interorganizacionais. Pois, percebe-se que grande parte dos procedimentos de prevenção de *deadlock* existentes estão relacionados às classes mais restritas das redes de Petri, como a S^3PR, utilizada em sistemas de manufaturas flexíveis. Ou então, os procedimentos para correção de *deadlock* não se baseiam em um procedimento sistemático, apenas em correções manuais do modelo, como o utilizado por [Aalst 1998].

Capítulo 4

Metodologia

Este capítulo dedica-se a apresentar a metodologia utilizada para garantir que os objetivos propostos para este trabalho sejam alcançados. A seguir são apresentados os procedimentos metodológicos que subsidiaram a pesquisa de modo a individualizar e a detalhar os conceitos adotados, assim como as etapas e a sistemática utilizadas para sua efetivação, visando conferir-lhe sustentação e validade científica.

Em [Marconi e Lakatos 2003] afirma-se que não pode haver ciência se não houver uma metodologia científica na resolução de um problema, a qual consiste numa série de atividades sistemáticas e racionais para se buscar, de maneira confiável, a solução para o problema. Para [Wazlawick 2008] o método consiste na sequência de passos necessários para demonstrar que o objetivo proposto foi alcançado. Então, se os passos definidos no método forem executados, os resultados obtidos deverão ser convincentes. Para [Minayo 1996], entende-se por metodologia "o caminho do pensamento e a prática exercida na abordagem da realidade".

É importante que o objetivo do trabalho esteja bem definido para que se faça a melhor decisão relacionada à escolha do método. Assim, uma vez que este trabalho pretende desenvolver uma arquitetura distribuída de um modelo livre de *deadlock*, o método deve contemplar as etapas de análise, modelagem e especificação desta arquitetura.

Em [Silva e Menezes 2005] são apresentados critérios para classificar os estudos sob os seguintes pontos de vista: quanto aos seus objetivos, quanto à sua natureza, quanto à abordagem do problema e quanto aos procedimentos técnicos adotados.

As pesquisas podem ser classificadas em três grupos de acordo com [Gil 2002]: exploratórias, descritivas e explicativas. Para o trabalho proposto nesta dissertação, a

pesquisa exploratória proporciona mais detalhes sobre o problema proposto, tornando-o mais explícito. Assim, o levantamento bibliográfico juntamente com o estudo de caso têm o papel de explorar os conceitos e analisar as possíveis soluções. O objetivo da pesquisa exploratória é então de modificar conceitos e ideias já propostas e estudadas para buscar o desenvolvimento de abordagens melhores.

Este trabalho, para melhor compreensão, foi dividido em uma pesquisa essencialmente exploratória, compreendendo a revisão bibliográfica e seleção de procedimentos existentes para solucionar partes da descrição do problema, assim como uma parte demonstrativa, com aplicação, validação e avaliação da metodologia proposta.

Esta pesquisa pode ser classificada segundo sua natureza como uma pesquisa do tipo aplicada, uma vez que são apresentadas soluções para problemas concretos. Em [Gil 2002] é destacado que neste modelo a preocupação está menos voltada para o desenvolvimento de teorias de valor universal que para aplicação imediata numa realidade circunstancial.

Referente à abordagem do problema, uma pesquisa pode utilizar uma metodologia qualitativa e quantitativa. A pesquisa qualitativa envolve significados, motivos, valores e atitudes que podem ser difíceis de serem quantificados, quando pensamos em um mundo intangível de significados das ações e das relações humana [Minayo 1996]. Na metodologia qualitativa não se requer um tratamento estatístico, e a interpretação dos acontecimentos e atribuição de significados a estes constituem um fundamento básico. Já na pesquisa quantitativa as informações são quantificadas, analisadas e classificadas [Silva e Menezes 2005]. Esta pesquisa possui aspectos essencialmente qualitativos que estão relacionados ao levantamento bibliográfico, ao estudo e escolha das técnicas para detecção e controle de situações de *deadlock*, à demonstração da utilização dos métodos selecionados juntamente ao procedimento proposto e à validação deste procedimento.

É preciso lembrar também que o positivismo define que todo conhecimento deve ser baseado em inferências lógicas a partir de um conjunto básico de fatos. Positivistas são reducionistas e preferem métodos baseados em teorias precisas. O positivismo é associado a experimentos controlados, questionários e estudos de casos [Creswell 2011]. Este trabalho será validado por meio de um estudo de caso, portanto também pode ser associado a uma visão positivista.

De acordo com [Yin 2008], os estudos de caso envolvem um exame amplo de uma única instância. Quando selecionado utilizando critérios bem definidos, mesmo um único estudo de caso pode ser suficiente para formular teorias e apresentar resultados. Um

estudo de caso será proposto neste trabalho para validação dos procedimentos para detecção e correção de situações de *deadlock* em *WorkFlow nets* interorganizacionais.

Segundo [Yin 2008], para projetar um estudo de caso é necessário um plano de pesquisa. Este plano de pesquisa é o esquema de pesquisa necessário para a definição inicial de quais questões estudar, quais dados são relevantes, quais dados coletar e como analisar os resultados.

Neste contexto, o método utilizado neste trabalho compreende as seguintes etapas:

- realizar um estudo minucioso sobre o referencial teórico, com ênfase em processos de negócio interorganizacionais, *WorkFlow nets* interorganizacionais e algoritmos para detecção e correção de situações de *deadlock*;
- formalizar processos de negócio por meio das *WorkFlow nets* interorganizacionais para possibilitar a análise qualitativa do modelo;
- apresentar um procedimento de detecção de *deadlock* nas *WorkFlow nets* interorganizacionais e mostrar que estas situações de *deadlock* estão relacionadas com estruturas em forma de sifão que se esvaziam;
- apresentar um procedimento de controle supervisório de sifões para remover as situações de *deadlock*;
- elaborar um modelo de arquitetura distribuída livre de *deadlock*;
- realizar pelo um estudo de caso ilustrativo colocando em prática a abordagem proposta.

A figura 4.1 apresenta e caracteriza o método utilizado. A etapa de "Caracterização do Problema" estabelece o vínculo entre a percepção do problema de situações de *deadlock* em processos de negócio interorganizacionais e a necessidade de um procedimento sistemático para solucioná-los. Já as etapas de "Análise do Modelo", "Modelagem e Especificação" e "Proposta de Arquitetura" incluem as contribuições desta dissertação e se voltam para a análise, modelagem e especificação de um modelo livre de situações de *deadlock*. Finaliza-se o método com uma proposta de arquitetura distribuída livre de tais situações. O detalhamento de cada uma destas etapas é feito nas seções subsequentes.

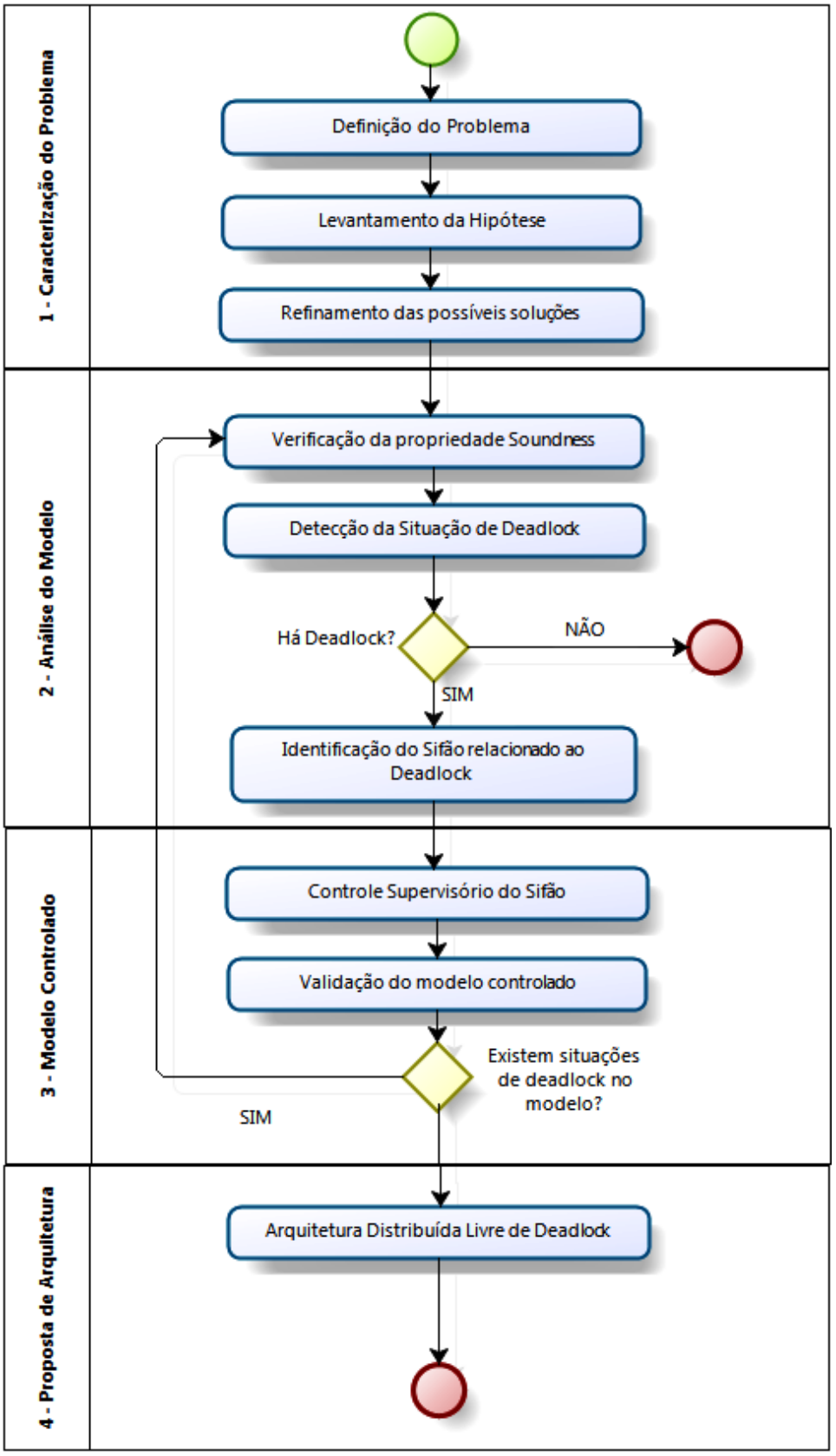

FIGURA 4.1: Método.

4.1 Caracterização do Problema

Em [Silbertschatz et al. 2004] os autores utilizam uma metáfora para definir um *deadlock*, que diz: "Quando dois trens se aproximarem um do outro em um cruzamento, ambos deverão parar completamente e nenhum dos dois deverá ser acionado até que o outro tenha partido". Podemos observar que a lei gera uma situação onde nenhum dos trens pode mais ser acionado. Apesar de ser uma lei inadequada, observa-se com frequência este tipo de situação no contexto de sistemas computacionais. No contexto de processos, o *deadlock* também pode ocorrer devido à competição entre processos concorrentes pela utilização de recursos compartilhados ou devido a problemas de sincronização entre roteiros que são executados em paralelo [Tanenbaum 2010].

O risco de situações de *deadlock* aumenta proporcionalmente ao aumento de sub processos que se comunicam e do conhecimento parcial das estruturas dos processos que colaboram. Assim, *workflows* interorganizacionais estão mais suscetíveis a situações de *deadlock*. Em [Shim et al. 2002] os autores fazem referência à dificuldade de detectar situações de *deadlock* em processos de *workflow* interorganizacional em tempo de execução. Por isso, é desejável detectar e remover o *deadlock* na definição do processo de *workflow* antes da execução real do processo correspondente.

Para encontrar e prevenir tais situações de *deadlock*, este trabalho está fundamentado na hipótese seguinte:

Hipótese: O caráter distribuído dos processos em *WorkFlow net* interorganizacionais produz com mais frequência construções de roteiros inconsistentes, gerados devido às estruturas chamadas sifões que, quando esvaziadas das fichas contidas nelas inicialmente, produzem situações de *deadlock*.

Para avaliar esta hipótese, as seguintes perguntas de pesquisa são propostas:

- Como detectar por meio de um procedimento sistemático situações de *deadlock* que inviabilizam a boa propriedade de *Soundness* das *WorkFlow nets* Interorganizacionais?
- É possível por meio de um procedimento sistemático corrigir as estruturas que produzem os *deadlocks* sem modificar por inteiro o modelo de especificação das *Workflow nets* Interorganizacionais?
- Como traduzir na forma de uma arquitetura distribuída os resultados obtidos por meio da detecção e correção automática das situações de *deadlock* em Sistemas de Gerenciamento de *Workflows* Interorganizacionais?

Para exemplificar e tornar mais claro o procedimento utilizado neste trabalho, consideremos o workflow interorganizacional apresentado em [Aalst 1998] e representado na Figura 4.2.

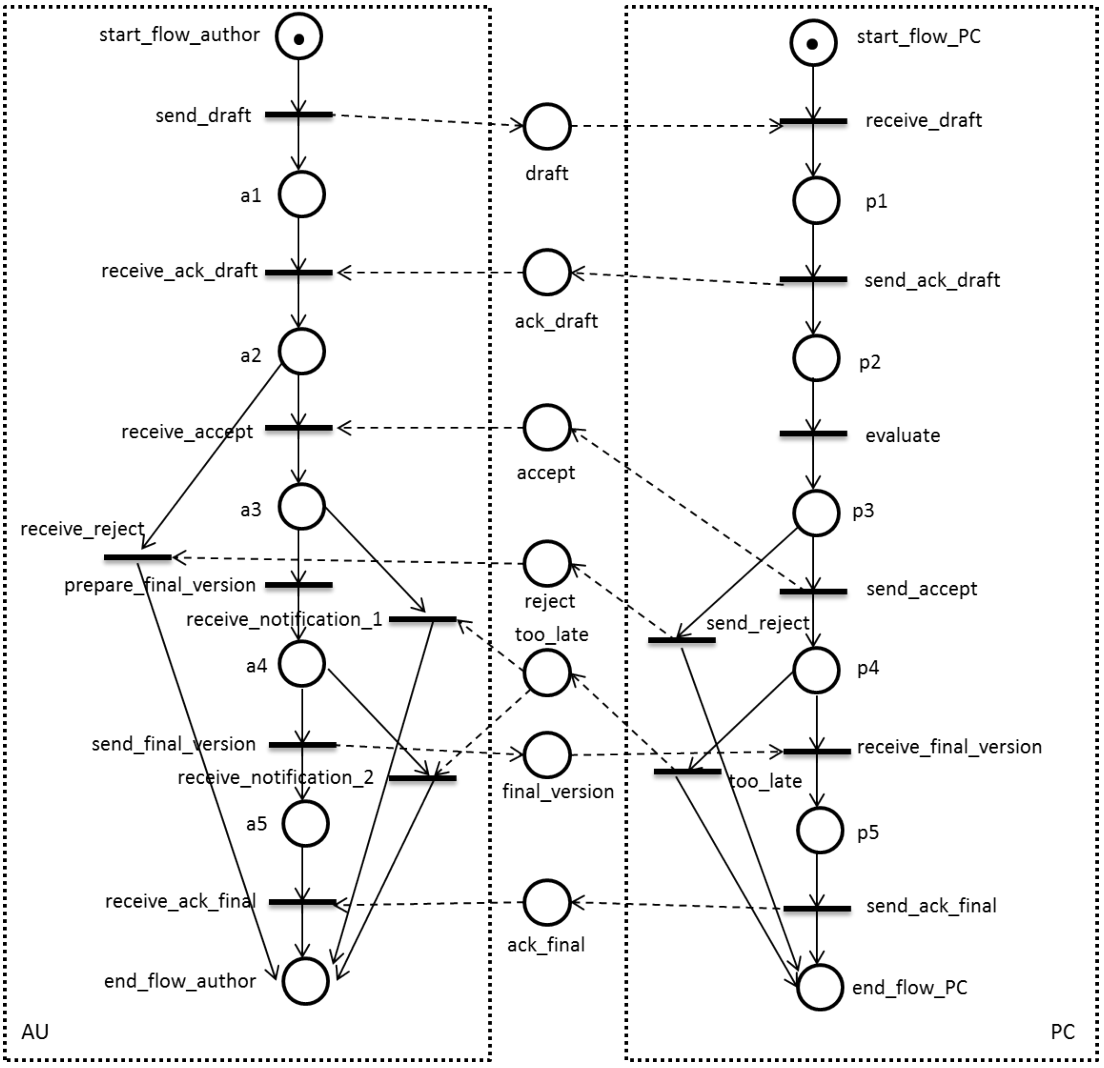

FIGURA 4.2: *Workflow* Interorganizacional.

Este exemplo modela o processo que precede a apresentação de um artigo em uma conferência: "Este *workflow* possui dois processos de negócio fracamente acoplados: (1) o processo de um autor preparar, submeter e revisar o artigo, e (2) o processo de avaliação e monitoramento das submissões pelo comitê do programa. Neste caso, existem duas organizações envolvidas no processo de negócio interorganizacional: o autor (AU) e o comitê do programa ($PC - program\ commitee$). O autor envia a versão preliminar do artigo para o comitê do programa. O comitê confirma o recebimento e avalia a submissão. O artigo é aceito ou rejeitado pelo comitê. Em ambos os casos o autor é notificado. Se o artigo for rejeitado, o processo termina, caso contrário o autor começa a preparar a versão final. Depois de completar a versão final, uma cópia é

enviada ao comitê do programa que confirma o recebimento da versão final. Se a versão final não for recebida pelo comitê até a data final especificada, o autor é notificado que o artigo não pode ser mais considerado. O artigo que é recebido tardiamente não será publicado nos anais."

O *workflow* interorganizacional apresentado na Figura 4.2 possui duas *Local WorkFlow nets (LWF-nets)*: *AU* e *PC*. Cada uma delas tem seu único lugar de entrada e lugar de saída. No caso da *LWF-net AU*, o lugar de entrada é $start\_flow\_author$ e o lugar de saída é end_flow_author. Na *LWF-net PC*, o lugar de entrada e o lugar de saída são, respectivamente, $start\_flow\_PC$ e end_flow_PC. Os lugares *draft*, ack_draft, *accept*, *reject*, *too_late*, $final\_version$ e ack_final são lugares de comunicação assíncronos.

Tomando este exemplo como base, as próximas seções deste capítulo apresentarão passo-a-passo o procedimento aplicado, facilitando assim o entendimento do mesmo.

4.2 Análise das boas propriedades

No contexto dos *workflows* interorganizacionais também é desejável que a propriedade *Soundness* seja verificada. Como é apresentado em [Aalst 1998], o fato de cada *LWF-net* ser *sound* não garante a propriedade *Soundness* para o processo global. Também é possível ter um *workflow* interorganizacional que é globalmente *sound* mas não é localmente *Sound*, como é mostrado em [Aalst 1998]. Para resolver este problema e garantir que o *workflow* seja localmente e globalmente *sound*, *Aaalst* define que a verificação, se uma *WorkFlow net* interorganizacional *(IOWF-net)* é globalmente *sound*, é baseada em uma transformação desta *IOWF-net* em uma *WorkFlow net* com um único processo. No modelo transformado, citado por *Aalst* como *unfolded net* ou *U(IOWF-net)*, todas as *WorkFlow nets* locais *(LWF-nets)* são incluídas em um único processo considerando uma transição inicial *ti* e uma transição final *to*. Um lugar de entrada *i* e um lugar de saída *o* também são adicionados para respeitar a estrutura básica de uma *WF-net* simples, e os elementos de comunicação assíncronos são mapeados como lugares comuns. A *U(IOWF-net)* da Figura 4.2 é apresentada na Figura 4.3.

4.2.1 Verificação das propriedades *Soundness* e vivacidade

Na verificação da propriedade *Soundness* um requisito importante a ser atendido é que, para cada caso a ser tratado, eventualmente o processo irá terminar e no momento que

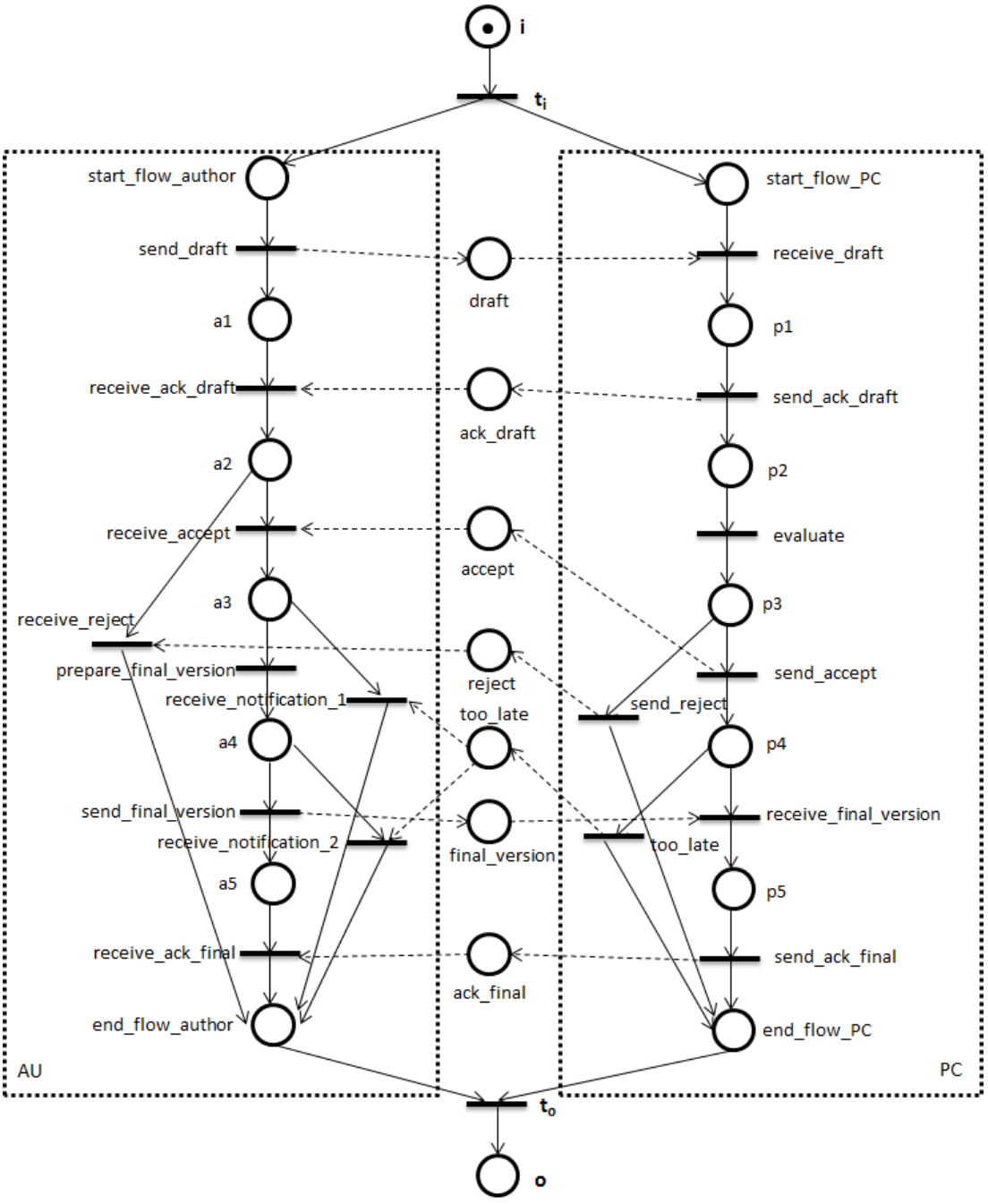

FIGURA 4.3: *U(IOWF-net)*.

o caso terminar de ser tratado deverá haver uma ficha no lugar de saída o e todos os outros lugares deverão estar vazios.

Considerando a *IOWF-net* apresentada na Figura 4.2, as *LWF-nets AU* e *PC* são ambas *sound*. A *U(IOWF-net)* apresentada na Figura 4.3 não é *sound*. O fato do modelo não ser *sound* não é uma condição suficiente para garantir a presença da situação de *deadlock*, mas é necessária como investigação inicial, pois uma rede de Petri *sound* não possui *deadlock*.

Ao analisarmos o disparo das transições e a disposição das fichas na Figura 4.3 é facilmente notado que se a transição *too_late* pertencente à *LWF-net PC* for disparada e se a transição $send\_final\_version$ que pertence à *LWF-net AU* também for disparada em sequência, as mensagens *too_late* e $final\_version$ se cruzarão, levando a uma situação de *deadlock* com uma ficha no lugar $a5$ e com duas mensagens que nunca serão recebidas (uma ficha no lugar *too_late* e uma ficha no lugar $final\_version$). Assim, como a *U(IOWF-net)* não é *sound*, a *IOWF-net* apresentada na Figura 4.2 também não é globalmente *sound*.

Sabendo disto, o próximo passo é verificar se a rede é viva. Caso não seja, identificar a presença da situação de total travamento, na qual a rede alcança um estado onde nenhuma transição poderá ser disparada. Para verificar isto, foi gerado, utilizando o software *PIPE*, um grafo de alcançabilidade, o qual é apresentado na Figura 4.4.

Considerando o vetor da marcação inicial da rede M_0 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0) representando, respectivamente, a marcação nos lugares: $start\_flow\_author$, $a1$, $a2$, $a3$, $a4$, $a5$, end_flow_author, $draft$, ack_draft, $accept$, $reject$, *too_late*, $final\_version$, ack_final, $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$ e end_flow_PC, ao fazer a análise dos estados alcançáveis pela rede é perceptível a situação de *deadlock* quando a rede alcança o estado *E19*, pois a partir deste estado nenhum outro é alcançado. A marcação da rede no estado *E19* é dada pelo vetor de marcação M_{E19} = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1), ou seja, a situação de *deadlock* acontece quando há fichas nos lugares $a5$, *too_late*, $final\_version$ e end_flow_PC. Para alcançar o estado *E19* é efetuado o disparo, nesta sequência, das seguintes transições: $send\_draft$, $receive\_draft$, $send\_ack\_draft$, $evaluate$, $receive\_ack\_draft$, $send\_accept$, *too_late*, $receive\_accept$, $prepare\_final\_version$ e $send\_final\_version$.

Identificado o *deadlock* e a marcação do sistema no momento que a rede alcança o estado *E19*, podemos concluir que:

- quando há fichas ao mesmo tempo nos lugares $a5$, *too_late*, $final\_version$ e end_flow_PC a rede está em situação de *deadlock*. Assim, o objetivo é evitar que o sistema alcance esta marcação;
- o disparo sequencial das transições $send\_final\_version$ e *too_late* é quem leva o sistema a alcançar tal travamento. Portanto, deve haver um sifão na rede, que se torna vazio de fichas, ao disparar as transições $send\_final\_version$ e *too_late*.

Desta forma, temos como próximo passo identificar, dentre o conjunto de sifões existentes, qual sifão (pode ser que exista mais de um sifão relacionado ao *deadlock*) caracteriza

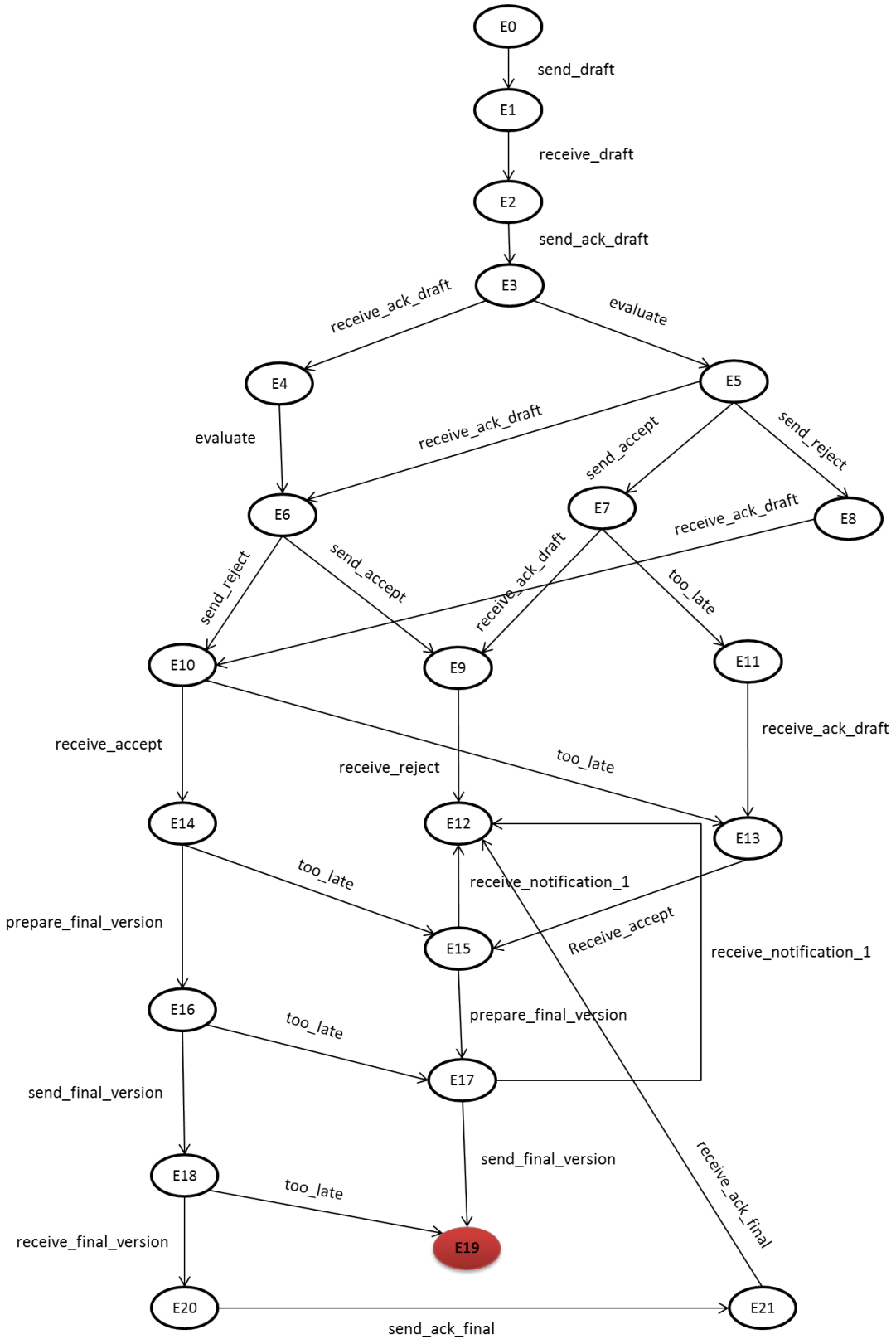

FIGURA 4.4: Grafo de alcançabilidade.

o esvaziamento de fichas que leva a este *deadlock*. É possível que existam sifões que se esvaziam, porém que não geram situações de *deadlock*. A existência de um sifão

vazio não implica na existência de um *deadlock*, porém, se existe um *deadlock*, podemos afirmar que existe um sifão que fica livre de fichas.

4.2.2 Detecção dos Sifões

Sabe-se que a presença de situações de *deadlock* nas redes de Petri está associada à existência de uma estrutura chamada sifão [Ezpeleta et al. 1993] [Barkaoui e Abdallah 1995b] [Iordache 2003] [Chu e Xie 1997] [Tricas e Martinez 1995]. O fato de existirem mais transições de saída que transições de entrada no sifão significa que este pode ser esvaziado, ficando livre de fichas, assim nenhuma transição poderá ser sensibilizada para eventualmente ser disparada, constituindo uma situação de total travamento do processo. Para que um sifão não se torne completamente livre de fichas é necessário que ele contenha uma *Trap* ou possua um controle supervisório. É provado em [Hack 1972] que uma condição suficiente para a propriedade *liveness* (vivacidade) em uma rede de Petri marcada é que para cada sifão em uma rede é necessário conter ao menos uma *Trap* marcada. Também é provado por *Hack* que uma condição necessária para se ter uma situação de *deadlock* em uma rede de Petri é ter ao menos um sifão vazio quando consideramos o conjunto de marcações acessíveis.

Como o foco deste trabalho não é apresentar um novo algoritmo para encontrar sifões, foram utilizados softwares para a detecção automática destas estruturas. Duas ferramentas foram utilizadas neste trabalho, uma delas foi o programa aplicativo *PIPE* (*Plataform Independent Petri net Editor*) [PIPE 2003] que foi criado como um projeto de um grupo de alunos de pós-graduação do Departamento de Computação do *Imperial College London*, em 2003. *PIPE* foi utilizado na criação e análise dos modelos deste trabalho e destaca-se por ser simples de usar, possuir uma plataforma independente e código fonte aberto, apresentando fácil extensibilidade. Outra ferramenta utilizada para esta pesquisa foi a *PNetLab* [PnetLab 2006] desenvolvida pelo grupo de Controle e Automação da Universidade de Salerno, Itália. Também foi usada no intuito de modelagem e análise das redes de Petri, porém esta ferramenta possui a limitação de analisar modelos com até vinte e cinco lugares na rede.

Uma condição necessária para encontrar sifões em uma rede de Petri é a de que a rede seja repetitiva ou parcialmente repetitiva. Assim, iremos modificar a Figura 4.2 acrescentando uma nova transição *SR*, arcos de entrada à esta transição saindo dos lugares end_flow_author e end_flow_PC e arcos de saída desta transição que serão entradas

para as transições *start_flow_author* e *start_flow_PC*. A rede de Petri repetitiva equivalente a IOWF-net da Figura 4.2 é apresentada na Figura 4.5.

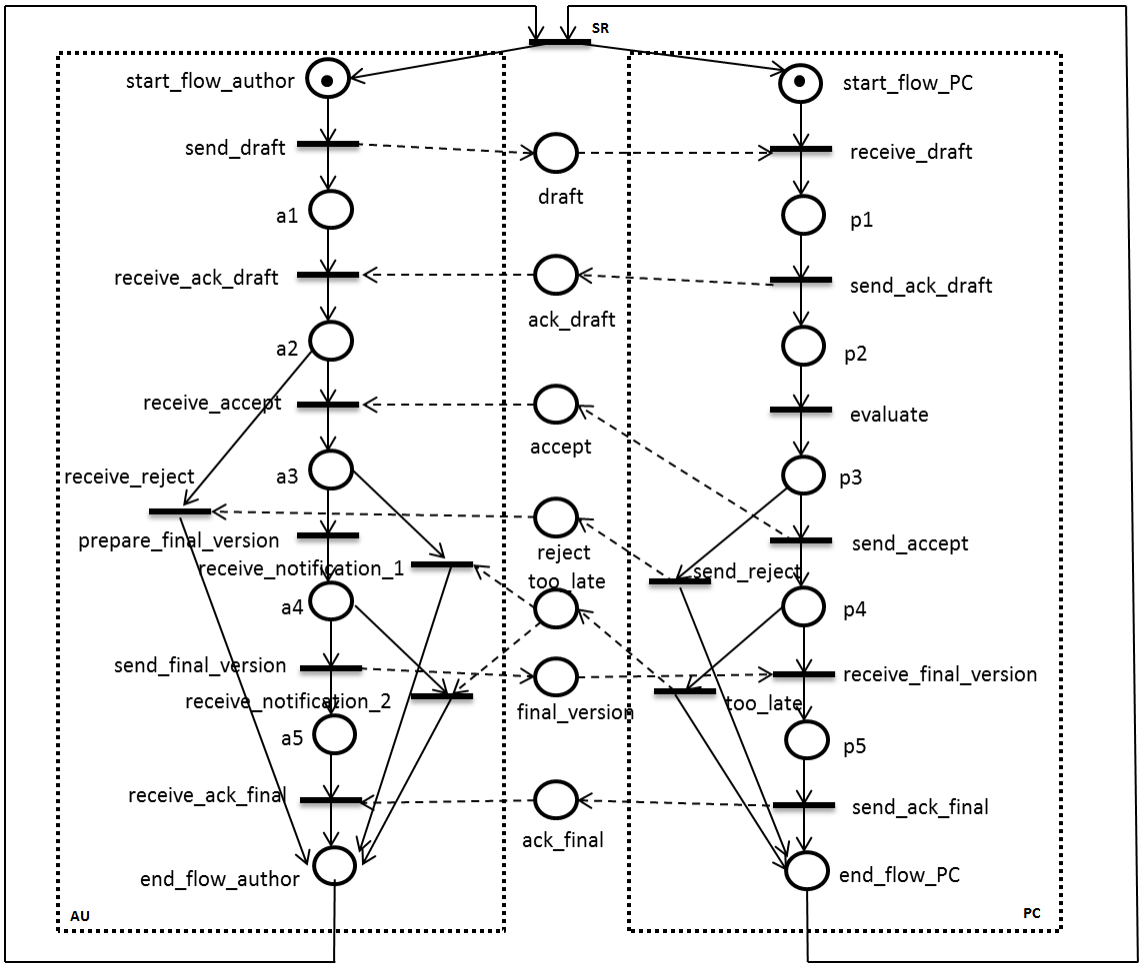

FIGURA 4.5: Rede de Petri repetitiva.

Utilizando como exemplo a *IOWF-net* da Figura 4.5 e o software *PIPE*, é possível encontrar os sifões apresentados na Tabela 4.1. A marcação com "x" na coluna *Trap* indica que o sifão possui uma *Trap*.

TABELA 4.1: Sifões e Traps referentes à Figura 4.5.

ID	Sifão	Trap
S1	$start\_flow\_author$, $p1$, $p2$, $p3$, $reject$, $a3$, $a4$, $a5$, end_flow_author, $draft$, $accept$	x
S2	$start\_flow\_PC$, $p1$, $p2$, $p3$, $reject$, $a3$, $a4$, $a5$, end_flow_author, $accept$	x
S3	$start\_flow\_PC$, $p1$, $a2$, ack_draft, $a3$, $a4$, $a5$, end_flow_author	x
S4	$final\_version$, ack_final, $start\_flow\_PC$, $p1$, $p5$, ack_draft, $a2$, $a3$, $a4$, end_flow_author	x
S5	$start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, end_flow_PC	x

S6	*too_late*, *ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *a*2, *ack_draft*, *end_flow_author*	
S7	*ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *ack_draft*, *a*2, *a*3, *a*4, *end_flow_author*	
S8	*too_late*, *ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *reject*, *end_flow_author*	x
S9	*ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *reject*, *a*3, *a*4, *end_flow_author*, *accept*	
S10	*final_version*, *ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*5, *reject*, *a*3, *a*4, *end_flow_author*, *accept*	x
S11	*start_flow_author*, *a*1, *a*2, *a*3, *a*4, *a*5, *end_flow_author*	x
S12	*start_flow_author*, *final_version*, *ack_final*, *p*5, *a*1, *a*2, *a*3, *a*4, *end_flow_author*	x
S13	*start_flow_author*, *p*1, *ack_draft*, *a*2, *a*3, *a*4, *a*5, *end_flow_author*, *draft*	x
S14	*start_flow_author*, *final_version*, *ack_final*, *p*1, *p*5, *ack_draft*, *a*2, *a*3, *a*4, *end_flow_author*, *draft*	x
S15	*start_flow_author*, *p*1, *p*2, *p*3, *p*4, *p*5, *end_flow_PC*, *draft*	x
S16	*start_flow_author*, *too_late*, *ack_final*, *p*1, *p*2, *p*3, *p*4, *p*5, *a*1, *a*2, *end_flow_author*, *draft*	
S17	*start_flow_author*, *too_late*, *ack_final*, *p*1, *p*2, *p*3, *p*4, *p*5, *a*2, *ack_draft*, *end_flow_author*, *draft*	
S18	*start_flow_author*, *too_late*, *ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *a*1, *a*2, *end_flow_author*	
S19	*start_flow_author*, *ack_final*, *p*1, *p*2, *p*3, *p*4, *p*5, *ack_draft*, *a*2, *a*3, *a*4, *end_flow_author*, *draft*	
S20	*start_flow_author*, *ack_final*, *p*1, *p*2, *p*3, *p*4, *p*5, *a*1, *a*2, *a*3, *a*4, *end_flow_author*, *draft*	
S21	*start_flow_author*, *ack_final*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *a*1, *a*2, *a*3, *a*4, *end_flow_author*	
S22	*start_flow_author*, *too_late*, *ack_final*, *p*1, *p*2, *p*3, *p*4, *p*5, *reject*, *end_flow_author*, *draft*	x
S23	*start_flow_author*, *ack_final*, *p*1, *p*2, *p*3, *p*4, *p*5, *reject*, *a*3, *a*4, *end_flow_author*, *draft*, *accept*	
S24	*start_flow_author*, *final_version*, *ack_final*, *p*1, *p*2, *p*3, *p*5, *reject*, *a*3, *a*4, *end_flow_author*, *draft*, *accept*	x

Analisando a tabela acima e tomando como ponto de partida o trabalho de [Hack 1972], todos os sifões que possuírem uma *Trap* relacionada não se esvaziam, portanto podem ser descartados de nossa próxima análise, visto que queremos encontrar o sifão que esvazia e que caracteriza a situação de *deadlock* identificada ao gerar o grafo de alcançabilidade apresentado na Figura 4.4. Sendo assim, dos vinte e quatro sifões encontrados, catorze deles possuem uma *Trap* relacionada, logo teremos que analisar dez sifões para encontrar o qual está relacionado ao *deadlock* identificado.

4.2.3 Identificação dos Sifões que esvaziam

O primeiro passo para a identificação do sifão que gera a situação de *deadlock* é encontrar o(s) sifão(ões) que, ao disparar sequencialmente as transições $send\_final\_version$ e *too_late*, se esvazia(m), ficando permanentemente livre de fichas. Assim, temos que analisar os dez, dos vinte e quatro sifões, que não possuem uma *Trap* relacionada.

Com auxílio do *PIPE*, o primeiro sifão a ser analisado da Tabela 4.1 é o *S6*, apresentado na Figura 4.6.

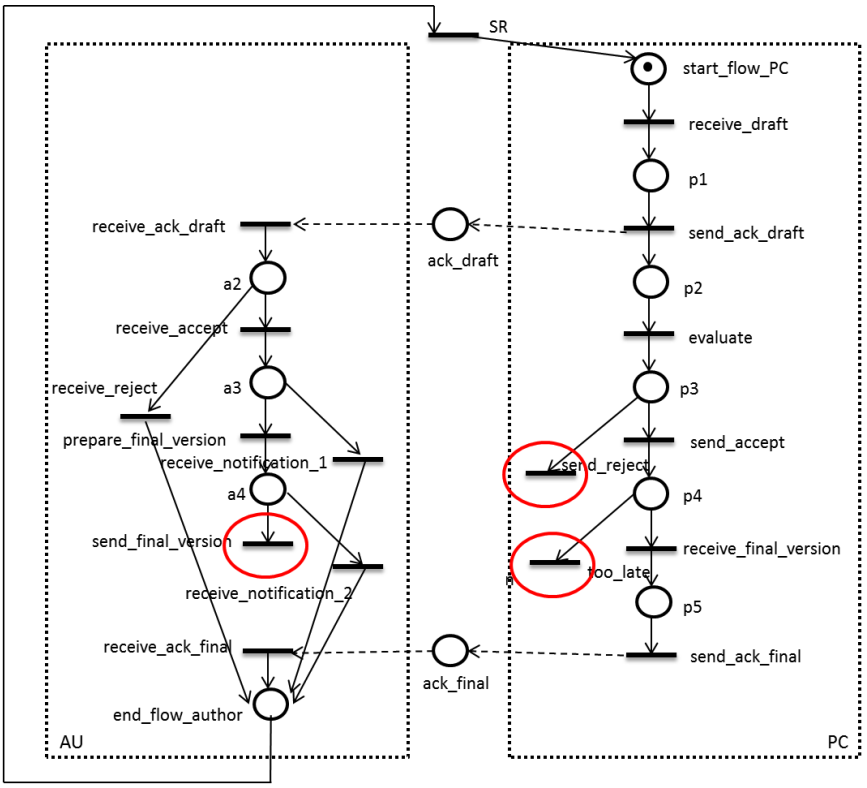

FIGURA 4.6: Sifão S6.

É visível que este sifão, ao disparar *send_final_version* e *too_late*, em sequência, fica livre de fichas. Portanto é um dos sifões que levam à situação de *deadlock*. O esvaziamento das fichas é caracterizado pelo disparo sequencial das transições *receive_draft*, *send_ack_draft*, *evaluate*, *send_reject*, *receive_ack_draft*, *receive_accept*, *prepare_final_version* e *send_final_version*. Porém, se dispararmos a transição *send_accept* em vez da *send_reject*, temos a possibilidade de disparar a transição *too_late* que também leva ao esvaziamento de fichas e que caracteriza a situação que procuramos.

Como podem existir outros sifões que também esvaziam pelo disparo sequencial das transições *send_final_version* e *too_late*, todos os outros nove sifões restantes deverão ser analisados. Assim, todo sifão que atenda a este requisito fará parte de um conjunto *SC*, que conterá todos os sifões que geram a situação de *deadlock* e que precisam ser controlados.

O próximo sifão a ser analisado é o *S7*, representado na Figura 4.7.

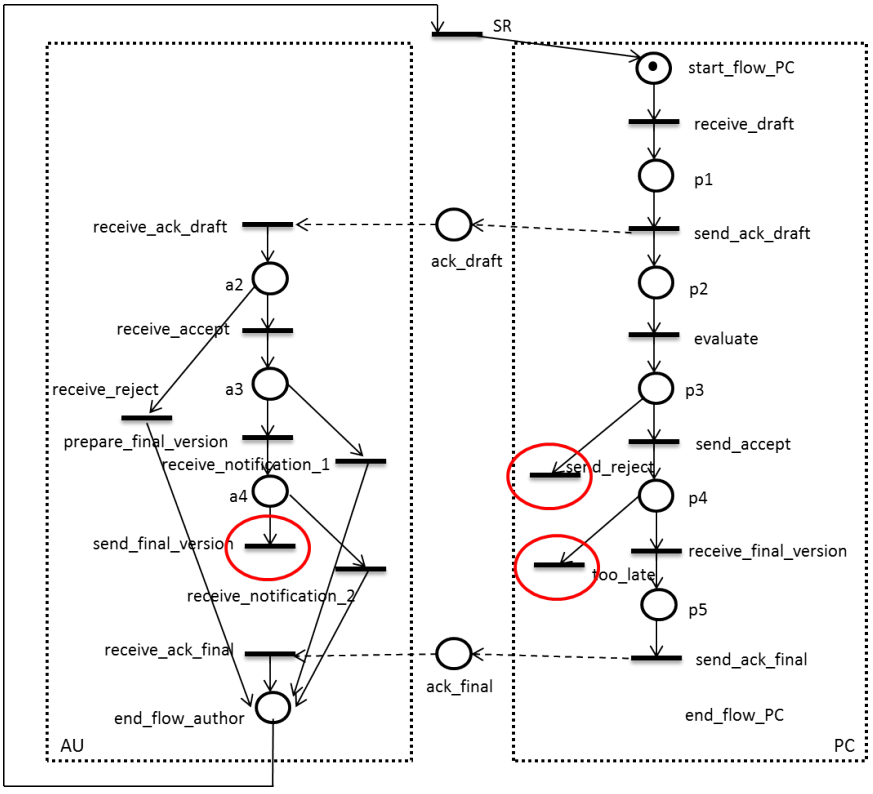

FIGURA 4.7: Sifão S7.

De forma similar ao Sifão *S6*, o esvaziamento das fichas é caracterizado pelo disparo sequencial das transições *receive_draft*, *send_ack_draft*, *evaluate*, *send_reject*, *receive_ack_draft*, *receive_accept*, *prepare_final_version* e *send_final_version*. Neste caso também, se dispararmos ao invés da transição *send_reject* a transição

send_accept, temos a possibilidade de disparar a transição *too_late* que também leva ao esvaziamento de fichas. Portanto, o sifão *S7* também será incluído em *SC*.

Representado na Figura 4.8, a análise agora é sobre o sifão *S9*, que caracteriza o esvaziamento das fichas por intermédio do disparo sequencial das transições: $receive\_draft$, $send\_ack\_draft$, *evaluate*, *send_accept*, *too_late*, *receive_accept*, $prepare\_final\_version$ e $send\_final\_version$. Portanto, pelo fato de o esvaziamento ocorrer devido às transições $send\_final\_version$ e *too_late* o sifão *S9* também será incluído em *SC*.

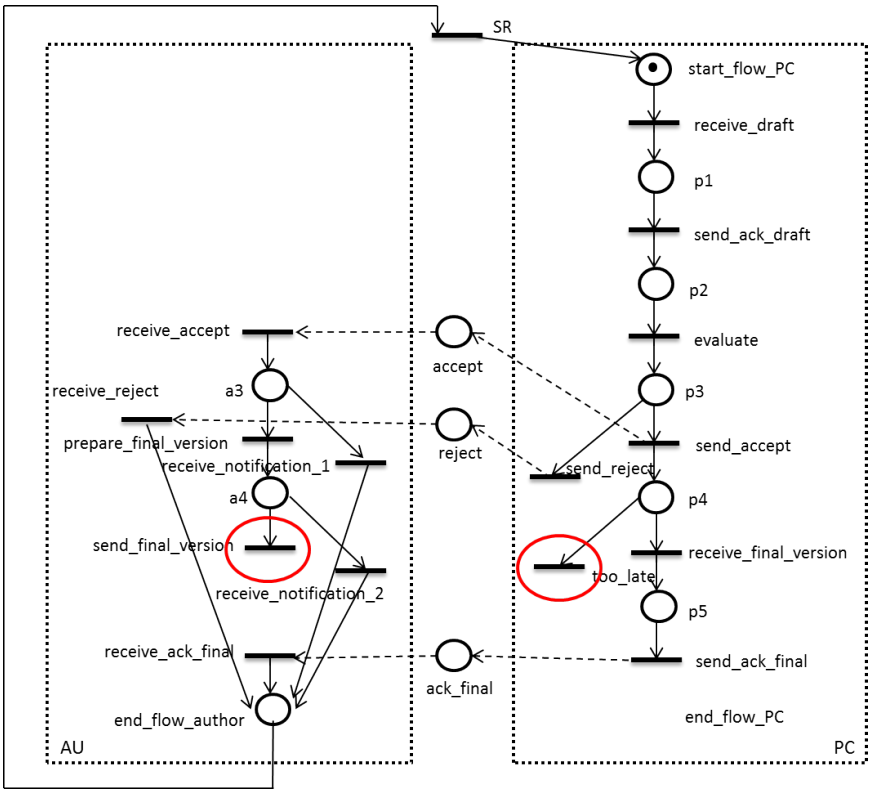

FIGURA 4.8: Sifão S9.

O sifão *S16*, apresentado na Figura 4.9, apresenta como sequência de disparo que leva ao esvaziamento das fichas: $send\_draft$, $receive\_ack\_\ draft$, *receive_accept*, $receive\_draft$, $send\_ack\_draft$, *evaluate*, *send_accept* e *too_late*. É perceptível que este sifão não possui a transição $send\_final\_version$, portanto não é o sifão que caracteriza a situação de *deadlock* que procuramos. Assim, este sifão será desconsiderado e não será incluído a $SC = \{S6, S7, S9\}$.

O próximo sifão para ser analisado é o *S17*, apresentado na Figura 4.10. A sequência de disparo que esvazia o sifão tem as transições: $send\_draft$, $receive\_draft$, $send\_ack\_draft$, $receive\_ack\_draft$, *receive_accept*, *evaluate* e $send\_reject$. Similar ao sifão *S16*, podemos perceber que o sifão não possui a transição $send\_final\_version$

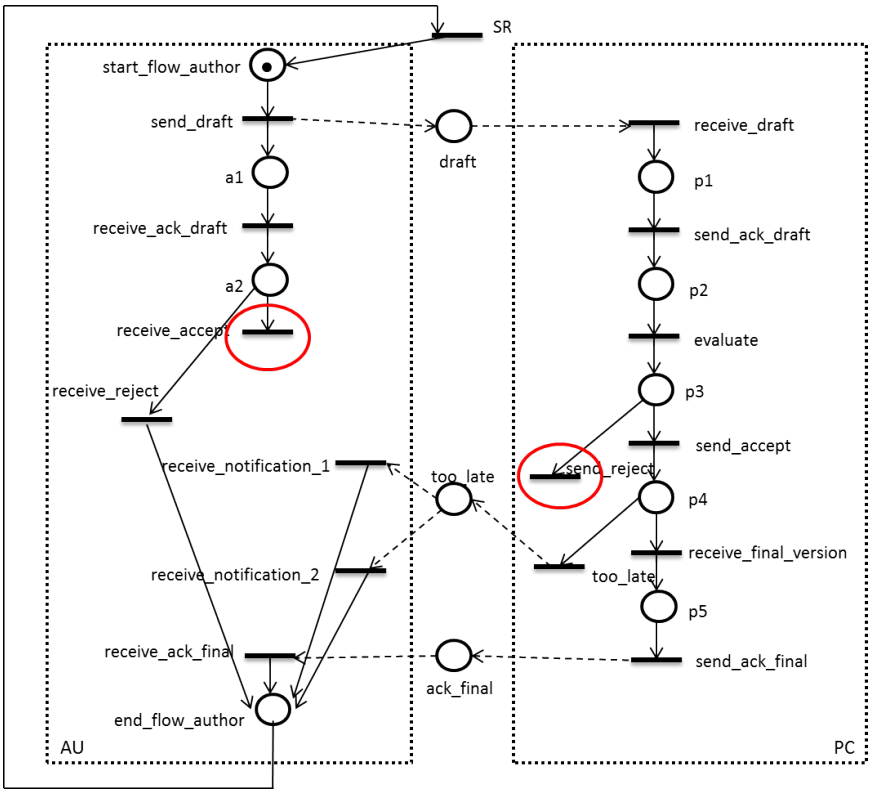

FIGURA 4.9: Sifão S16.

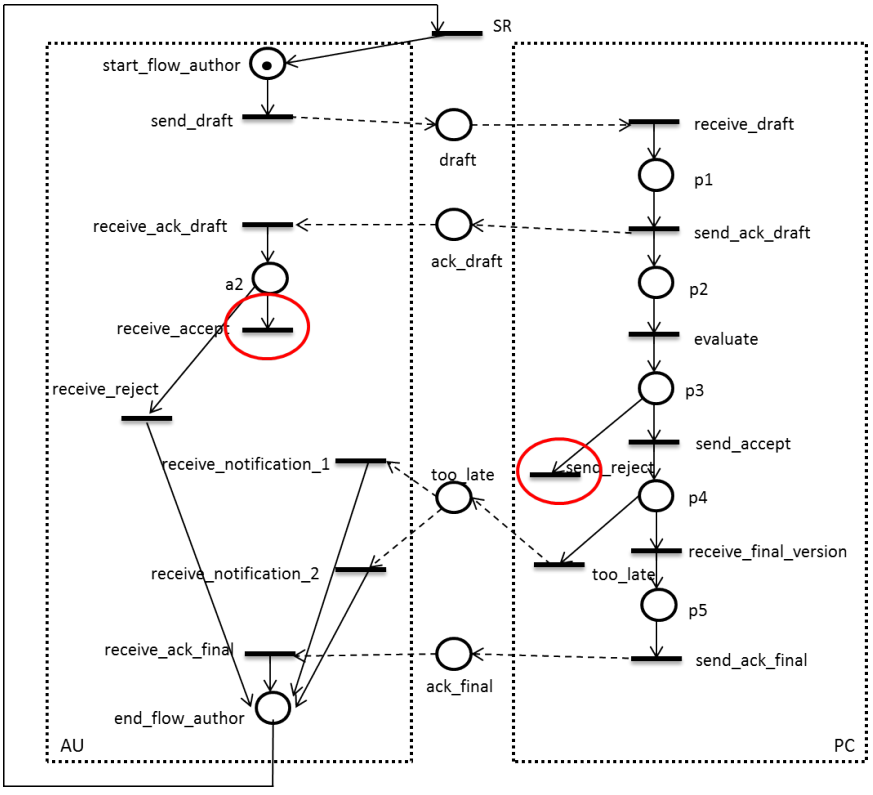

FIGURA 4.10: Sifão S17.

e podemos perceber também que o disparo da transição *too_late* não leva ao esvaziamento das fichas, assim, também iremos desconsiderar este sifão. Portanto, o conjunto *SC* permanece apenas com {*S6, S7, S9*} até o momento.

O sifão *S18*, apresentado na Figura 4.11, apresenta uma sequência de disparo que leva ao esvaziamento das fichas: $send\_draft$, $receive\_ack\_draft$, $receive\_accept$, $receive\_draft$, $send\_ack\_draft$, $evaluate$, $send\_accept$, too_late. Esta sequência é a mesma do sifão *S16*, assim, da mesma maneira, este sifão não possui a transição $send\_final\_version$, portanto não caracteriza a situação de *deadlock* que procuramos e não será acrescentado ao conjunto *SC*.

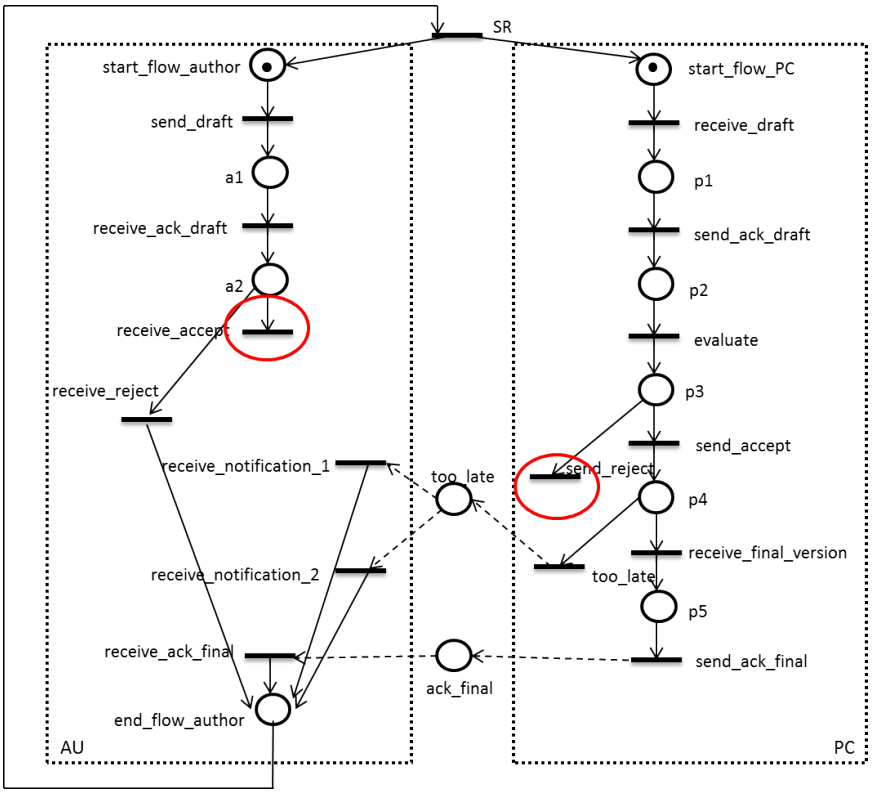

FIGURA 4.11: Sifão S18.

Na Figura 4.12 está representado o sifão *S19*. Analisando este sifão temos uma sequência de disparo que leva ao esvaziamento de fichas as transições: $send\_draft$, $receive\_draft$, $send\_ack\_draft$, $evaluate$, $send\_reject$, $receive\_ack\_draft$, $receive\_accept$, $prepare\_final\_version$, $send\_final\_version$. E também, se dispararmos ao invés da transição $send\_reject$ a transição $send\_accept$, temos a possibilidade de disparar a transição too_late esvaziando o sifão. Portanto, o sifão *S19* também será incluído ao conjunto $SC = \{S6, S7, S9, S19\}$.

Na figura 4.13 está representado o sifão *S20*. Analisando este sifão temos uma sequência de disparo que leva ao esvaziamento de fichas a mesma sequência de disparo do sifão

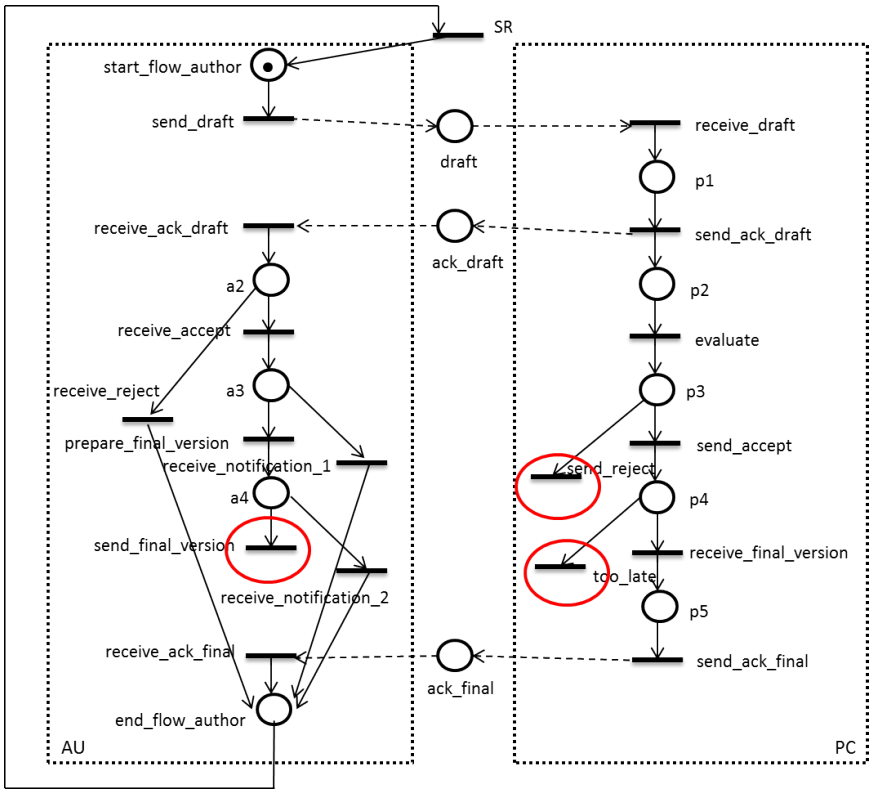

FIGURA 4.12: Sifão S19.

S19. Portanto, o sifão *S20* também será incluído ao conjunto $SC = \{S6, S7, S9, S19, S20\}$. Da mesma forma o sifão *S21* na Figura 4.14, que também é similar aos sifões *S19* e *S20*, possui as transições que levam ao esvaziamento de fichas que caracterizam a situação de *deadlock* que pretendemos controlar. Assim, *S21* também fará parte de *SC*.

E por fim, o sifão *S23*, representado na Figura 4.15, que tem uma sequência de transições que tornam o sifão vazio: $send\_draft$, $receive\_draft$, $send\_ack\_draft$, $evaluate$, $send\_accept$, too_late, $receive\_accept$, $prepare\_final\_version$ e $send\_final\_version$. Semelhante aos sifões que já pertencem a *SC*, o sifão *S23* também apresenta as transições que caracterizam a situação de *deadlock* encontrada.

Desta forma, após analisar todos os possíveis sifões, o conjunto dos sifões a serem controlados será composto pelos sifões $SC = \{S6, S7, S9, S19, S20, S21, S23\}$. Por meio da análise do conjunto *SC*, mesmo que todos estes sifões caracterizem a situação de *deadlock*, devido ao fato de que o esvaziamento ocorre em função do disparo das transições $send\_final\_version$ e too_late, estes sifões apresentam características diferentes, tais como marcação inicial e a presença de uma outra transição ($send\_reject$) que também leva ao esvaziamento de fichas. Assim, em relação ao conjunto *SC* podemos extrair as seguintes características:

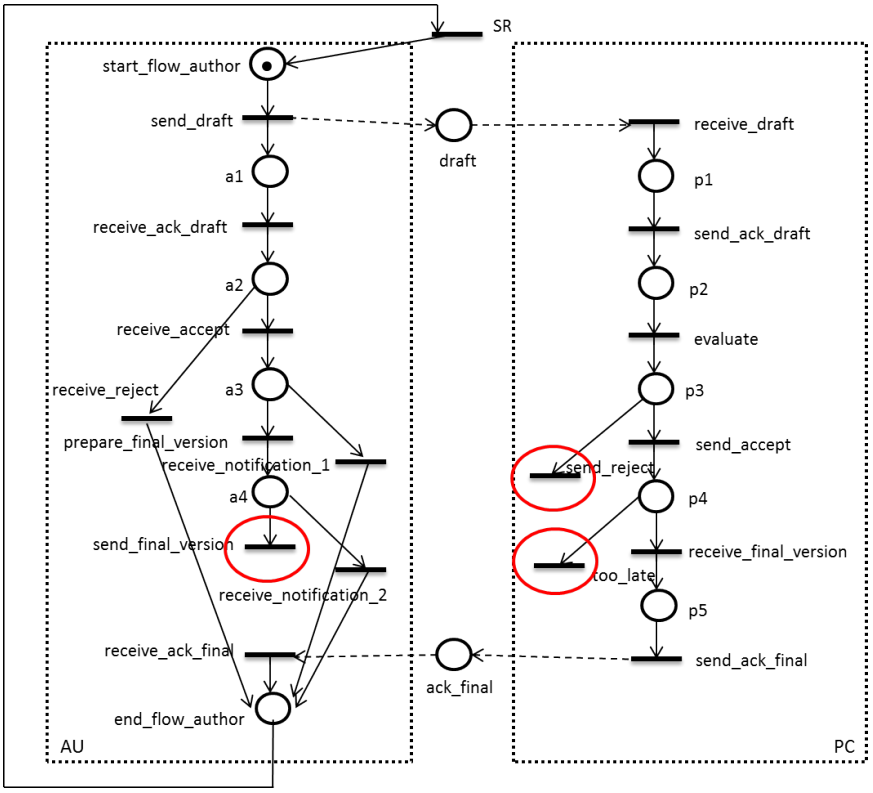

FIGURA 4.13: Sifão S20.

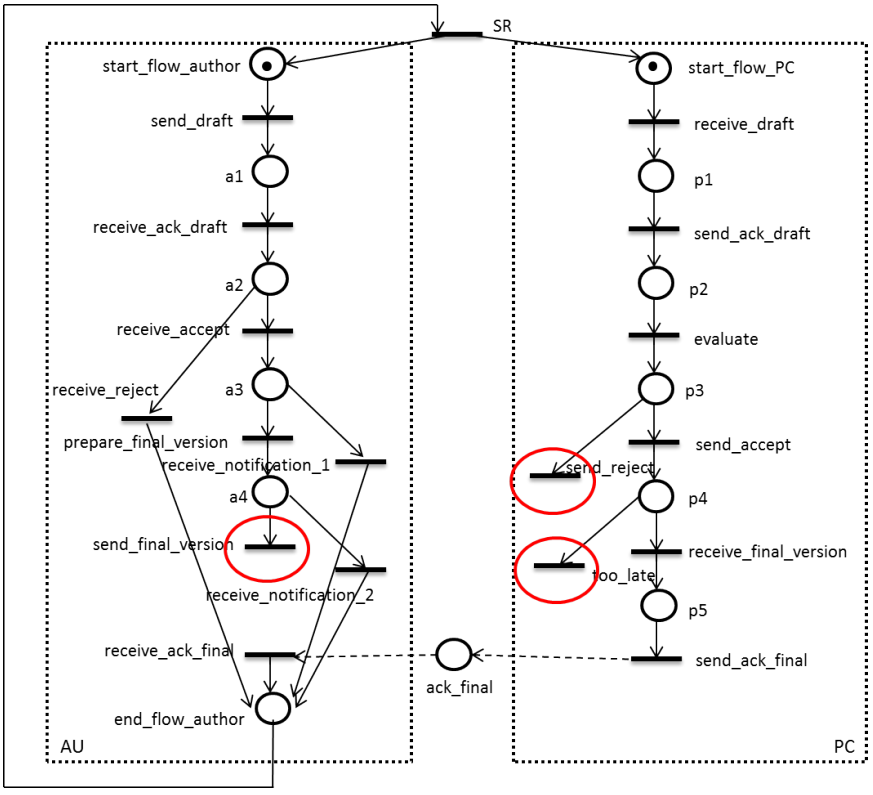

FIGURA 4.14: Sifão S21.

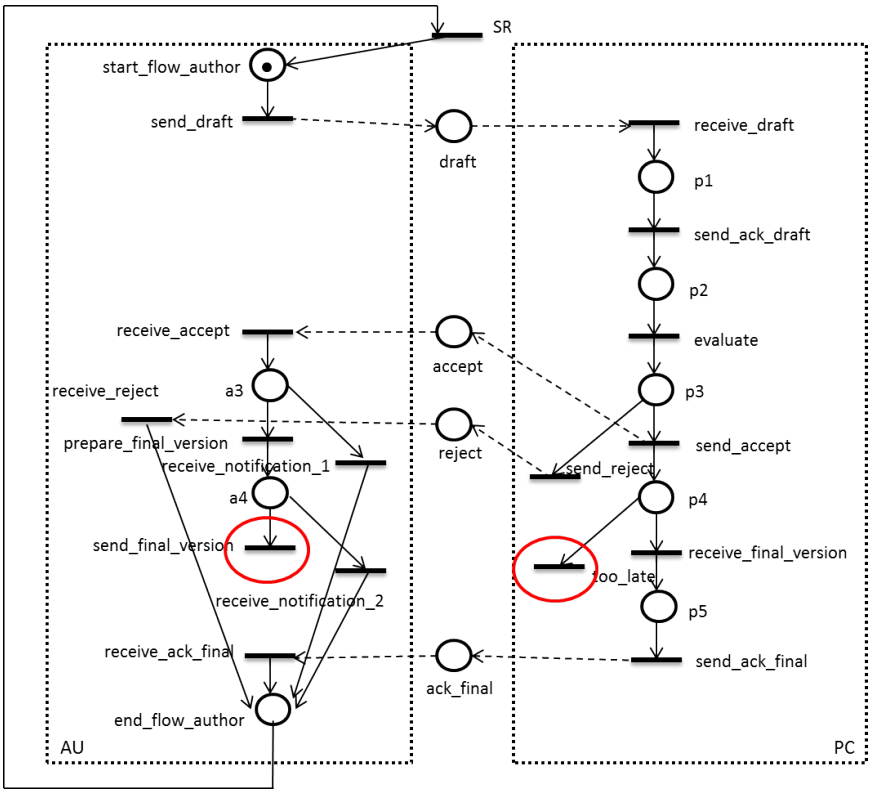

FIGURA 4.15: Sifão S23.

- os sifões *S6, S7, S9, S19, S20* e *S23* têm como marcação inicial uma ficha no lugar $start\_flow\_author$ ou uma ficha no lugar $start\_flow\_PC$. Já o sifão *S21* tem como marcação inicial duas fichas: uma ficha no lugar $start\_flow\_author$ e outra ficha no lugar $start\_flow\_PC$.

- os sifões *S9* e *S23* possuem duas transições que levam ao esvaziamento de fichas do sifão: $send\_final\_version$ e too_late. Os sifões *S6, S7, S19, S20* e *S21* possuem três transições que levam o esvaziamento de fichas: $send\_final\_version$, $send\_reject$ e too_late. Como existe um caso mais específico e não nos interessa para esta situação de *deadlock* o controle da transição $send\_reject$, iremos desconsiderar os sifões *S6, S7, S19, S20* e *S21*. Sabendo que se controlarmos o disparo das transições $send\_final\_version$ e too_late é suficiente para impedir que estas duas transições sejam sequencialmente disparadas e, logo, impedir que a rede alcance a marcação *E19*, identificada no grafo de alcançabilidade como estado de total travamento do sistema, podemos então apenas controlar os sifões *S9* e *S23* para garantirmos que esta situação de *deadlock* seja impedida.

Como todos os sifões vazios relacionados ao *deadlock* devem ser controlados, iremos aplicar o processo de controle supervisório aos sifões *S9*, *S21* e *S23*.

4.3 Modelo Controlado

Ao identificar quais sifões precisam ser controlados para impedir que a rede alcance o estado que gere a situação de *deadlock*, o próximo passo apresentado na seção 4.3.1 é a inserção do controle supervisório que impedirá o esvaziamento destes sifões. Na seção 4.3.2 faremos a validação da inclusão do lugar de controle na Figura 4.2, para garantir que a inclusão de um novo lugar irá solucionar a situação de travamento identificada e que não irá gerar novas situações.

4.3.1 Controle Supervisório

O método de controle supervisório escolhido para este trabalho, apresentado junto ao referencial teórico no Capítulo 2, é o método dos invariantes de lugar [Moody et al. 1994] [Iordache e Antsaklis 2006]. Este método consiste em sintetizar um supervisor que coordene, no caso interorganizacional abordado neste trabalho, as atividades das *LWF-nets* de modo que o *IOWF-net* global seja *sound*, ou seja, livre de *deadlock*. Os invariantes de lugar correspondem ao conjunto de lugares para os quais a soma de fichas permanece constante para todas as marcações acessíveis pela rede que modela o invariante, garantindo assim que não haverá esvaziamento das fichas.

O primeiro sifão a ser controlado é o *S9*, composto pelo conjunto de lugares $S9 = \{start\_flow\_PC,\ p1,\ p2,\ p3,\ p4,\ p5,\ accept,\ reject,\ ack\_final,\ a3,\ a4,\ end\_flow\_author\}$. Já foi identificado previamente também que as transições a serem controladas para evitar que este sifão se esvazie são $send\_final\_version$ e *too_late*.

Como desejamos que estas duas transições não sejam disparadas sequencialmente, para estabelecer as restrições de controle sobre o disparo de duas transições precisamos fazer uma modificação no sifão, também apresentada no referencial teórico. A modificação do sifão *S9* pode ser vista na Figura 4.16. A transição $send\_final\_version$ foi transformada para acrescentar um lugar *m1* e uma transição $send\_final\_version'$ e a transição *too_late* foi transformada para acrescentar um lugar *m2* e uma transição *too_late*′. Assim, é possível estabelecer as restrições baseadas nos invariantes de lugar, na qual não se pode ter fichas em *m1* e *m2* simultaneamente. Pois, se esta situação ocorrer, caracteriza-se o esvaziamento de fichas no sifão *S9* original. Desta forma, podemos estabelecer a seguinte especificação de controle: $M(m1) + M(m2) \leq 1$, ou seja, a marcação do lugar *m1* somada à marcação do lugar *m2* tem de ser menor ou igual a um.

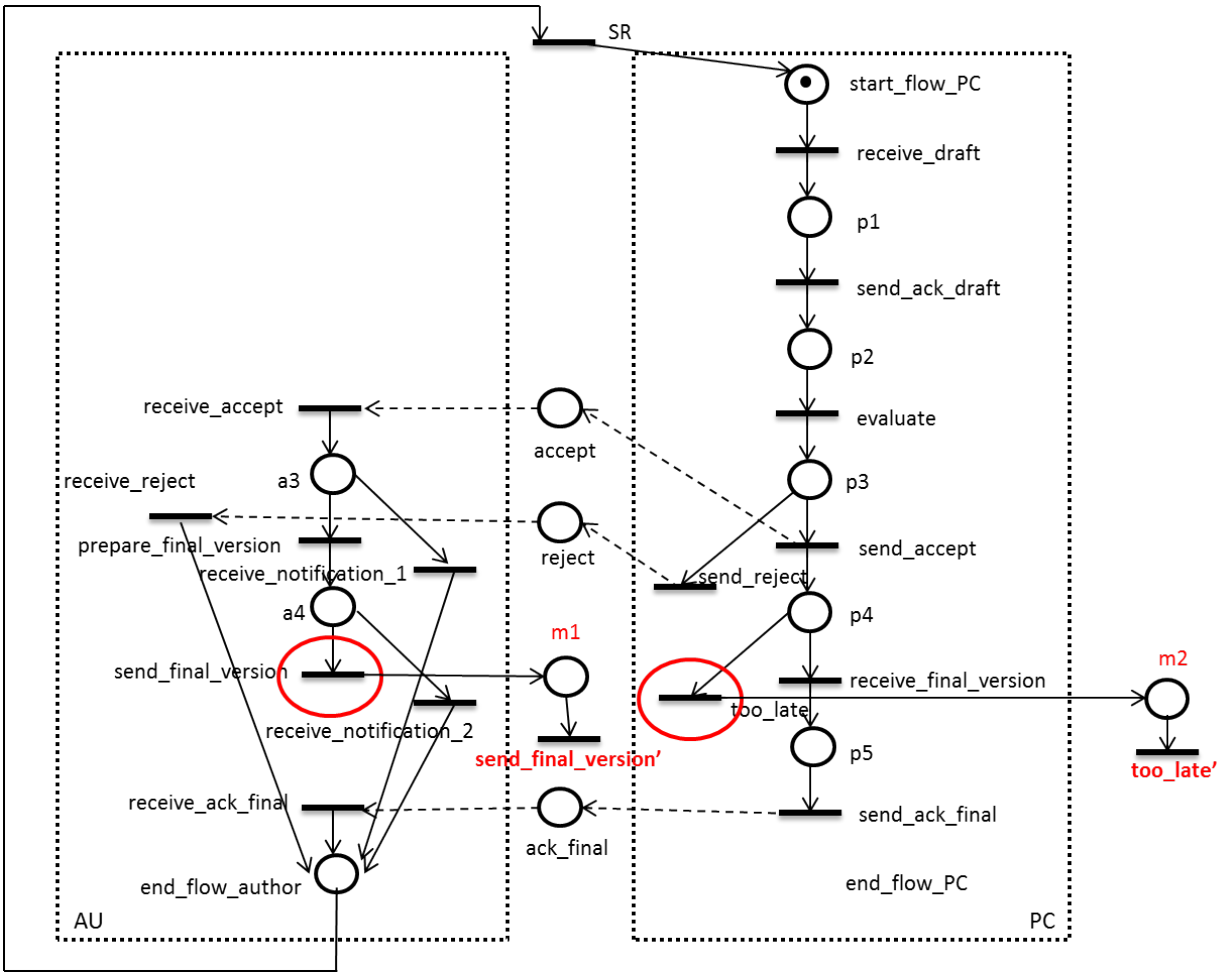

FIGURA 4.16: Sifão S9 modificado.

Dada a especificação acima e os lugares: $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, $accept$, $reject$, ack_final, $a3$, $a4$, end_flow_author, $m1$ e $m2$ temos que:

$$\mathrm{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Para se encontrar a matriz de incidência I_C tendo a matriz L com a marcação das restrições, são necessários valores correspondentes à matriz de incidência do sifão *S9* modificado.

Essa matriz está representada a seguir:

$I_{S9} =$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1
\end{bmatrix}$$

Com:

- **Linhas:** $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, $accept$,$reject$, ack_final, $a3$, $a4$, end_flow_author, $m1$ e $m2$.
- **Colunas:** $evaluate$, $prepare\_final\_version$, $receive\_accept$, $receive\_ack\_final$, $receive\_draft$, $receive\_final\_version$, $receive\_notification\_1$, $receive\_notification\_2$, $receive\_reject$, $send\_accept$, $send\_ack\_draft$, $send\_ack\_final$, $send\_final\_version'$, $send\_reject$, SR, too_late, too_late'.

A Tabela 4.2 apresenta os valores da aplicação da função $I_C = -$ L.I para se encontrar o valor da matriz I_C, que contém uma linha a mais com os valores referentes ao controlador a ser inserido.

Tabela 4.2: Transições de entrada e saída do lugar de controle do sifão S9.

ID	**Transição**	**−L.I**	**Arco**
1	$evaluate$	0	–
2	$prepare\_final\_version$	0	–
3	$receive\_accept$	0	–
4	$receive\_ack\_final$	0	–
5	$receive\_draft$	0	–
6	$receive\_final\_version$	0	–
7	$receive\_notification\_1$	0	–
8	$receive\_notification\_2$	0	–

9	$receive\_reject$	0	–
10	$send\_accept$	0	–
11	$send\_ack\_draft$	0	–
12	$send\_ack\_final$	0	–
13	$send\_final\_version$	-1	saída
14	$send\_final\_version'$	1	entrada
15	$send\_reject$	0	–
16	SR	0	–
17	too_late	-1	saída
18	too_late'	1	entrada

Da tabela acima, podemos então gerar a matriz I_C que possui a marcação do lugar de controle. Sendo que o valor 1 representa os arcos de entrada ao controlador e o valor -1 os arcos de saída. Portanto:

$$I_C = \left[\begin{array}{cccccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 \end{array} \right].$$

A marcação do lugar de controle é dada pela marcação inicial da rede menos 1. Portanto a $M_{C0} = 1 - 1 = 0$.

$$M_0 = (1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)$$

Isso significa que um lugar de controle será inserido ao sifão: *CP1* e este lugar terá marcação inicial igual a 0 (nenhuma ficha inicialmente). *CP1* terá como entrada os arcos que sairão das transições $send\_final\_version'$ e too_late' e como saída os arcos que serão entrada das transições $send\_final\_version$ e too_late. Para a inserção do lugar de controle, tomaremos a versão original do Sifão *S9* representado na Figura 4.8. Segundo [Moody et al. 1994], após a imposição do controle, pode-se retornar ao modelo inicial. Então, observamos que comparada à versão modificada, a transição correspondente à transição $send\_final\_version'$ é a transição $receive\_ack\_final$ e a transição too_late' tem como transições correspondentes $receive\_notification\_1$ e $receive\_notification\_2$.

A Figura 4.17 apresenta a inclusão do lugar de controle ao sifão *S9*. Mediante a execução simulada desta rede, utilizando o software *PIPE*, percebemos que o sifão será controlado, impedindo que as transições $send\_final\_version$ e too_late sejam disparadas. Logo, o lugar de controle inserido conseguiu executar o que foi proposto na sua inclusão, que é controlar o disparo destas transições.

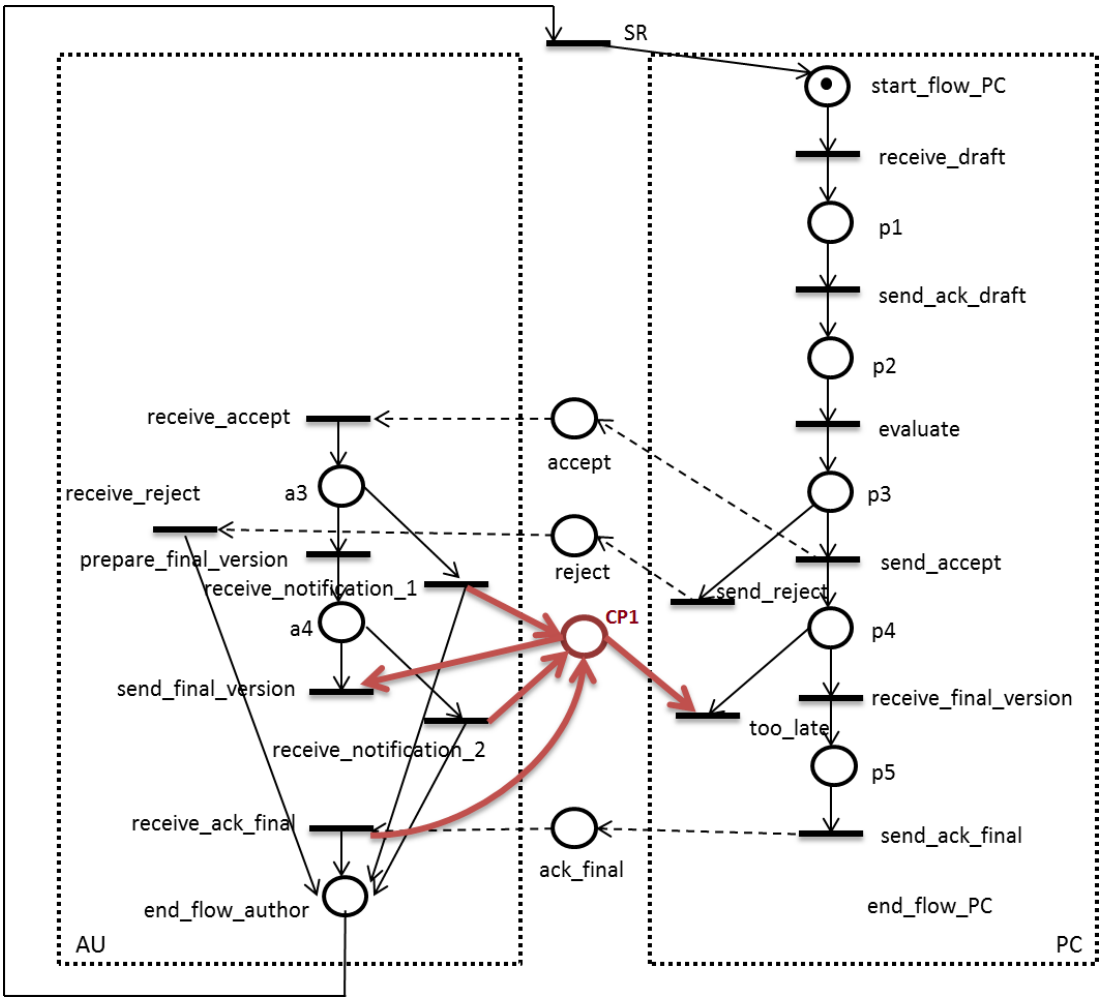

FIGURA 4.17: Sifão S9 com lugar de controle CP1.

O próximo sifão a ser controlado é o *S21*, composto pelo conjunto de lugares $S21 = \{start\_flow\_author,\ ack\_final,\ start\_flow\_PC,\ p1,\ p2,\ p3,\ p4,\ p5,\ a1,\ a2,\ a3,\ a4,\ end\_flow\_author\}$. Da mesma forma que foi feito no sifão *S9*, será necessário modificar o sifão *S21* para que atenda os requisitos para utilizarmos o método dos invariantes de lugar. A estrutura modificada do sifão *S21* é apresentada na Figura 4.18.

Um detalhe referente ao sifão *S21* é que ele possui uma transição a mais (*send_reject*) que também leva ao esvaziamento do sifão. Mesmo essa transição não fazendo parte do cenário que leva a situação de *deadlock*, neste trabalho ela também será levada em consideração ao encontrar o controle supervisório, pois o interesse é controlar todas as transições do sifão que levam ao esvaziamento das fichas e controlá-las não irá interferir no resultado final esperado, apenas acrescentará arcos de entrada e saída no controlador. Assim, a especificação de controle é dada pela expressão: $M(m1) + M(m2) + M(m3) \leq 1$.

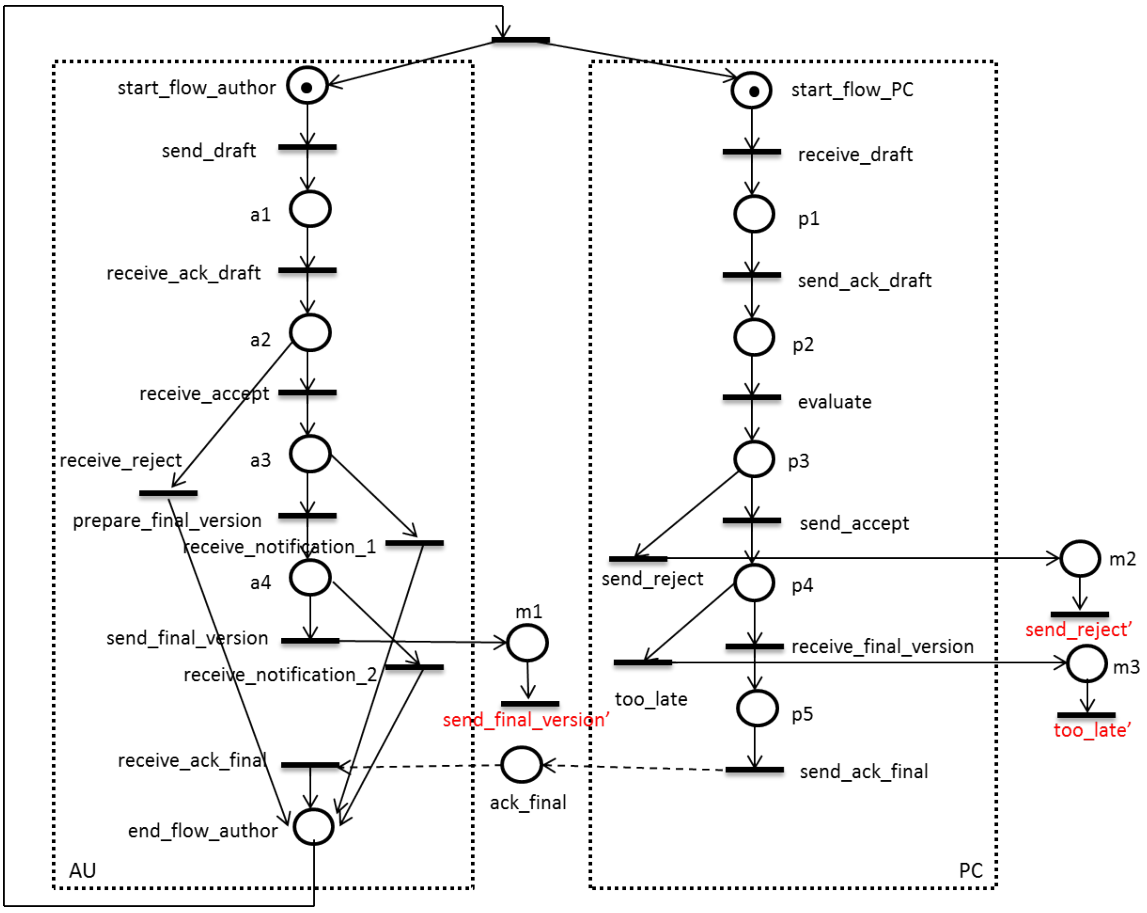

FIGURA 4.18: Sifão S21 modificado.

Dada a especificação $M(m1)+M(m2)+M(3) \leq 1$ e o conjunto de lugares: $start\_flow\_author$, ack_final, $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, $a1$, $a2$, $a3$, $a4$, end_flow_author, $m1$, $m2$ e $m3$, temos que:

$$\mathrm{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Abaixo temos a matriz de incidência do sifão *S21* modificado:

$I_{S21} =$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Com:

- **Linhas:** $evaluate$, $prepare\_final\_version$, $receive\_accept$, $receive\_ack\_draft$, $receive\_ack\_final$, $receive\_draft$, $receive\_final\_version$, $receive\_notification\_1$, $receive\_notification\_2$, $receive\_reject$, $send\_accept$, $send\_ack\_draft$, $send\_ack\_\ final$, $send\_draft$, $send\_final\_version$, $send\_reject$, $send\_final\_version'$, $send\_\ reject'$, too_late', SR e too_late.
- **Colunas:** $start\_flow\_author$, $a1$, $a2$, $a3$, $a4$, end_flow_author, $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, ack_final, $m1$, $m2$ e $m3$.

Tendo a matriz de incidência I_{S21}, o próximo passo para encontrar os arcos do lugar de controle é identificar a matriz I_C, que é calculada por meio da expressão: $I_C = -\text{L.C.}$

Os cálculos para encontrar matriz I_C referente ao sifão *S21* são apresentados na Tabela 4.3.

Tabela 4.3: Transições de entrada e saída do lugar de controle do sifão S21.

ID	Transição	−L.I	Arco
1	*evaluate*	0	–
2	$prepare\_final\_version$	0	–
3	$receive\_accept$	0	–
4	$receive\_ack\_draft$	0	–
5	$receive\_ack\_final$	0	–
6	$receive\_draft$	0	–
7	$receive\_final\_version$	0	–
8	$receive\_notification\_1$	0	–
9	$receive\_notification\_2$	0	–
10	$receive\_reject$	0	–
11	$send\_accept$	0	–
12	$send\_ack\_draft$	0	–
13	$send\_ack\_final$	0	–
14	$send\_draft$	0	–
15	$send\_final\_version$	-1	saída
16	$send\_reject$	-1	saída
17	$send\_final\_version'$	1	entrada
18	$send\_reject'$	1	entrada
19	SR	0	–
20	*too_late*	-1	saída
21	*too_late*′	1	entrada

Da tabela acima, temos que:

$$I_C = \Big[\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ -1\ \ -1\ \ 1\ \ 1\ \ 1\ \ 0\ \ -1\ \Big].$$

Ao obter a matriz I_C e sabendo que marcação inicial do sifão *S21* é M_0 = (1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), com duas fichas, uma em $start\_flow\_author$ e outra em $star\_flow\_PC$, a marcação do lugar de controle é dada por: $M_{CP2} = 2 - 1 = 1$. É possível verificar que o lugar de controle terá como entrada os arcos que sairão das transições $send\_final\_version'$, $send\_reject'$ e *too_late*′ e como saída os arcos que serão entradas das transições $send\_final\_version$, $send\_reject$ e *too_late*.

A figura 4.19 representa a inserção do lugar de controle no sifão *S21*. Para confirmar que o lugar de controle inserido conseguiu controlar o disparo simultâneo das transições

send_final_version e *too_late*, foi realizada a execução utilizando o *PIPE*, e foi comprovado que o sifão será controlado e não ficará vazio em nenhuma execução a partir desta marcação inicial.

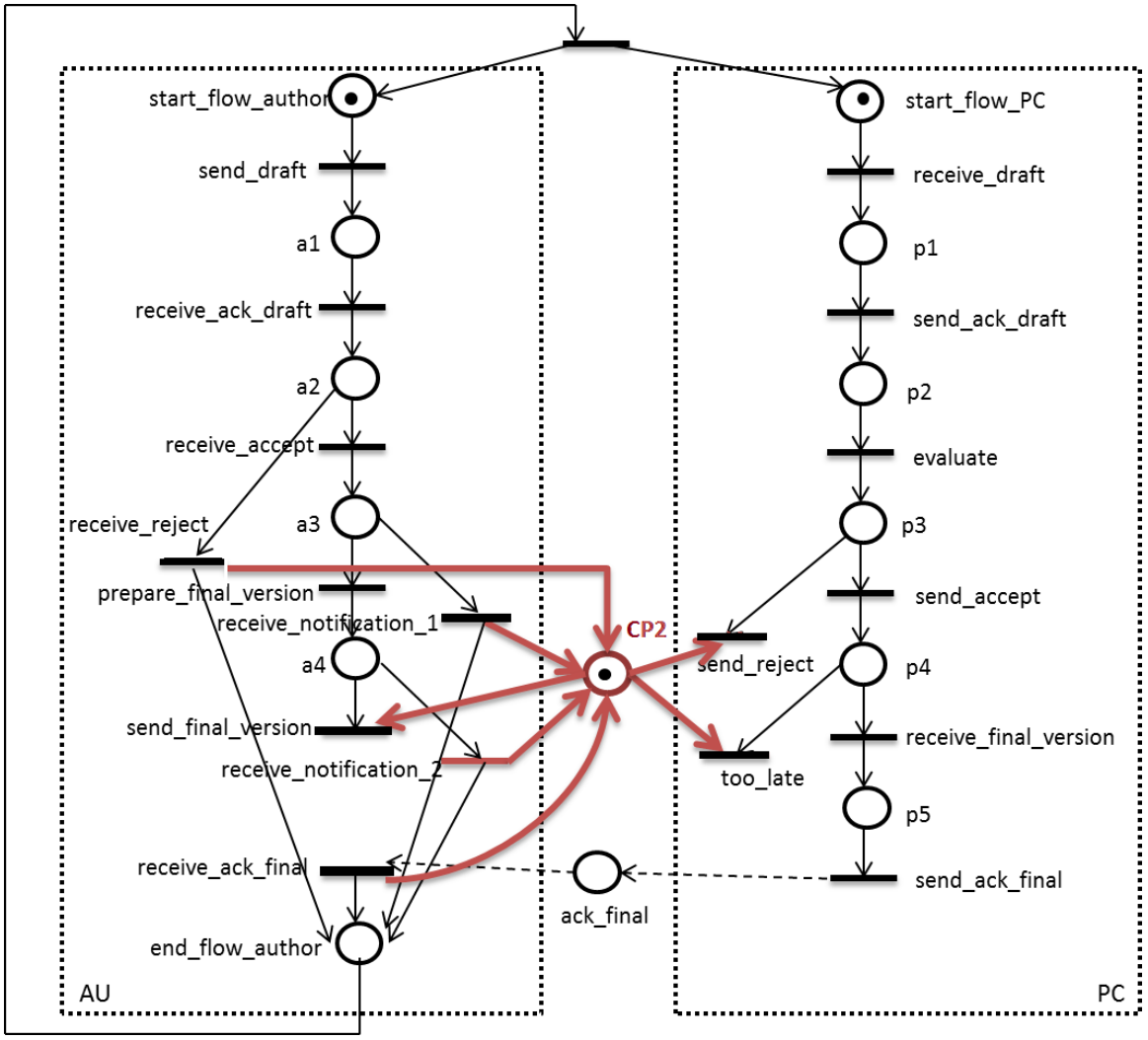

FIGURA 4.19: Sifão S21 com lugar de controle CP2.

Por fim, o último sifão a ser controlado é o *S23*, composto pelo conjunto de lugares $S23 = start\_flow\_author$, *draft*, *p1*, *p2*, *p3*, *p4*, *p5*, *accept*, *reject*, ack_final, *a3*, *a4*, end_flow_author. Da mesma maneira que foi feito nos sifões anteriores, será necessário modificar o sifão *S23* para que atenda os requisitos necessários à utilização do método dos invariantes de lugar. A estrutura modificada do sifão *S23* é apresentada na figura 4.20.

O sifão *S23* é semelhante ao sifão *S9*, portanto a especificação de controle é dada pela expressão: $M(m1) + M(m2) \leq 1$.

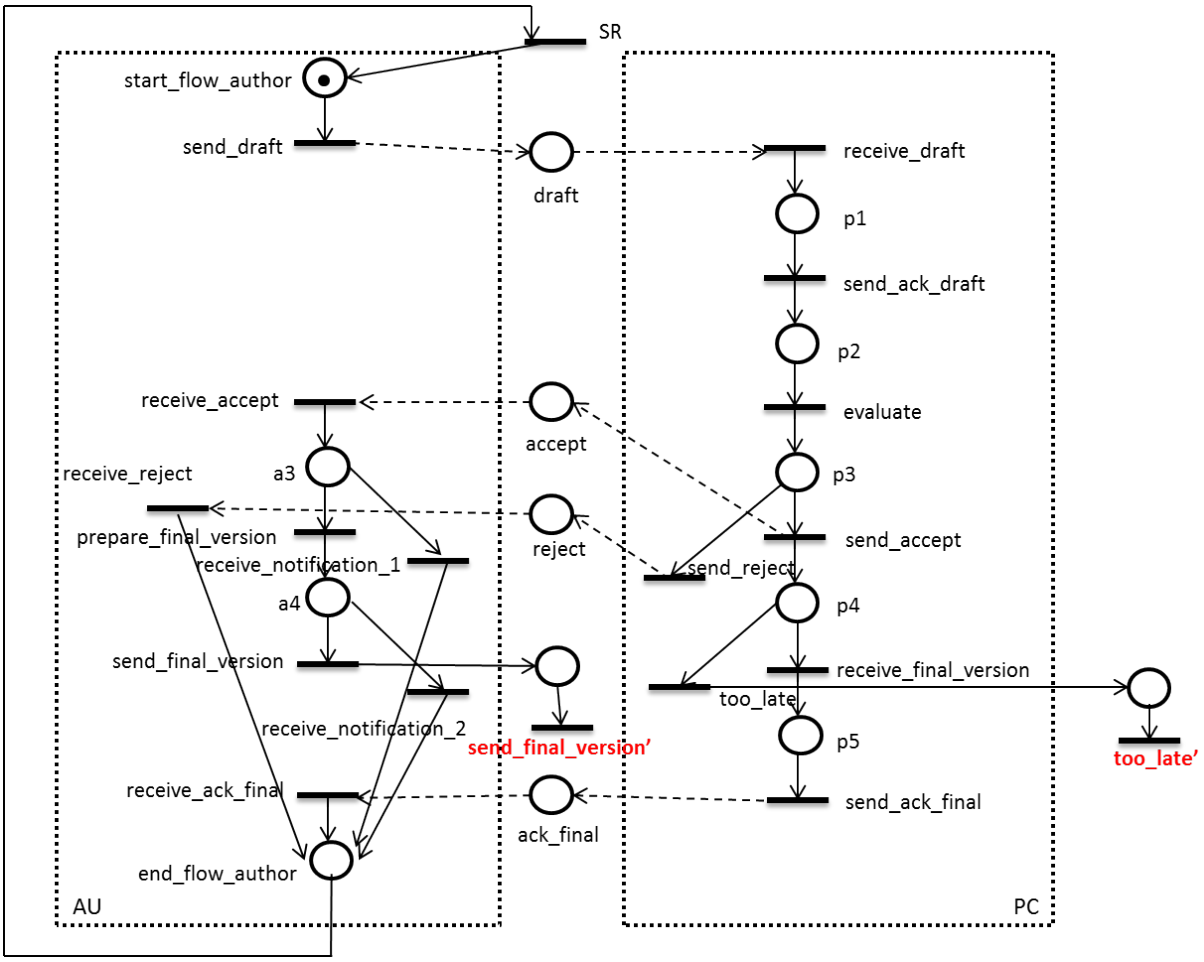

FIGURA 4.20: Sifão S23 modificado.

Dada a especificação $M(m1) + M(m2) \leq 1$ e os lugares: $start\_flow\_author$, $draft$, $p1$, $p2$, $p3$, $p4$, $p5$, $accept$, $reject$, ack_final, $a3$, $a4$, end_flow_author, $m1$ e $m2$, temos que:

$$\mathrm{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

A matriz de incidência do sifão $S23$ modificado é apresentada a seguir:

$I_{S23} =$

$$\begin{bmatrix}
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{bmatrix}$$

Com:

- **Linhas:** $evaluate$, $prepare\_final\_version$, $receive\_accept$, $receive\_ack\_final$, $receive\_draft$, $receive\_final\_version$, $receive\_notification\_1$, $receive\_notification\_2$, $receive\_reject$, $send\_accept$, $send\_ack\_draft$, $send\_ack\_final$, $send\_draft$, $send\_final\_version$, $send\_final\_version'$, $send\_reject$, SR, too_late e too_late'.
- **Colunas:** $start\_flow\_author$, $draft$, $p1$, $p2$, $p3$, $p4$, $p5$, $accept$, $reject$, ack_final, $a3$, $a4$, end_flow_author, $m1$ e $m2$

Os cálculos para encontrar matriz I_C referente ao sifão *S23* são apresentados na Tabela 4.4.

Tabela 4.4: Transições de entrada e saída do lugar de controle do sifão S23.

ID	Transição	$-$L.I	Arco
1	*evaluate*	0	–
2	$prepare\_final\_version$	0	–
3	$receive\_accept$	0	–
4	$receive\_ack\_final$	0	–
5	$receive\_draft$	0	–
6	$receive\_final\_version$	0	–
7	$receive\_notification\_1$	0	–
8	$receive\_notification\_2$	0	–
9	$receive\_reject$	0	–
10	$send\_accept$	0	–
11	$send\_ack\_draft$	0	–
12	$send\_ack\_final$	0	–
13	$send\_final\_version$	-1	saída
14	$send\_final\_version'$	1	entrada
15	$send\_reject$	0	–
16	SR	0	–
17	*too_late*	-1	saída
18	*too_late*$'$	1	entrada

Da tabela acima, temos que:

$$I_C = \Big[\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ -1\ \ -1\ \ 1\ \ 1\ \ 1\ \ 0\ \ -1\ \Big].$$

De maneira similar aos outros sifões apresentados acima, a marcação inicial deste lugar de controle é dada pela subtração de 1 à marcação inicial do sifão, portanto $M_{CP3} = 1 - 1 = 0$. Assim, podemos perceber que o lugar de controle será o mesmo do sifão *S9*. A figura com o sifão *S23* controlado é apresentado na Figura 4.21. Para confirmar se a inclusão deste lugar de controle no sifão *S23* cumpriu as especificações estabelecidas, a simulação da rede também foi realizada no *PIPE* e as transições $send\_final\_version$ e *too_late*, após o controle supervisório, realmente não puderam mais ser disparadas sequencialmente. Logo o controle do sifão está ocorrendo do modo esperado.

Após a inclusão do controle supervisório nos três sifões (*S9, S21* e *S23*), podemos perceber que temos dois tipos de controle supervisório: um marcado inicialmente com uma ficha e que controla três transições ($send\_reject$, $send\_final\_version$ e *too_late*) e outro sem marcação inicial e que controla duas transições ($send\_final\_version$ e *too_late*).

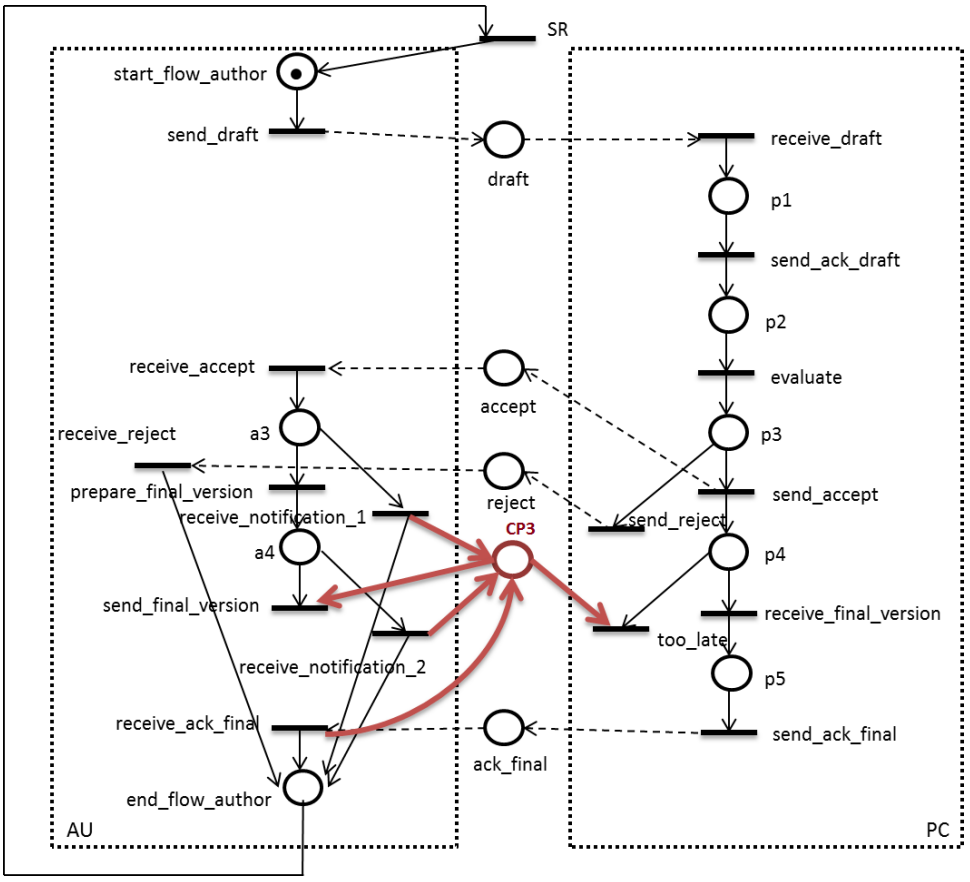

FIGURA 4.21: Sifão S23 com lugar de controle CP3.

4.3.2 Validação do modelo com novo lugar de controle

Além de validar se a inclusão do lugar de controle está efetivamente executando o que foi proposto para o controle do sifão, ou seja, se realmente está impedindo o sifão de ficar completamente vazio de fichas, é necessário incluir o lugar de controle ao modelo inicial, apresentado na Figura 4.2, e validar se o controle também se aplica a rede completa impedindo-a de alcançar o estado de travamento. Sabendo que, com a inclusão de um novo lugar na rede, novas situações indesejadas podem surgir, é importante verificar, para toda inserção de um controle supervisório, se nenhuma outra situação de *deadlock* foi gerada. Se isto ocorrer, ou seja, se outro *deadlock* aparecer, será necessário executar novamente todo o procedimento apresentado neste método para o controle do novo *deadlock* gerado.

Como temos dois tipos de controle supervisório, vamos inserir cada um deles em uma *IOWF-net*. O lugar de controle *CP1* correspondente aos sifões *S9* e *S23* acrescentado à *IOWF-net* apresentada na Figura 4.22 e o lugar de controle *CP2* correspondente ao sifão *S21* acrescentado à *IOWF-net* é apresentado na Figura 4.23.

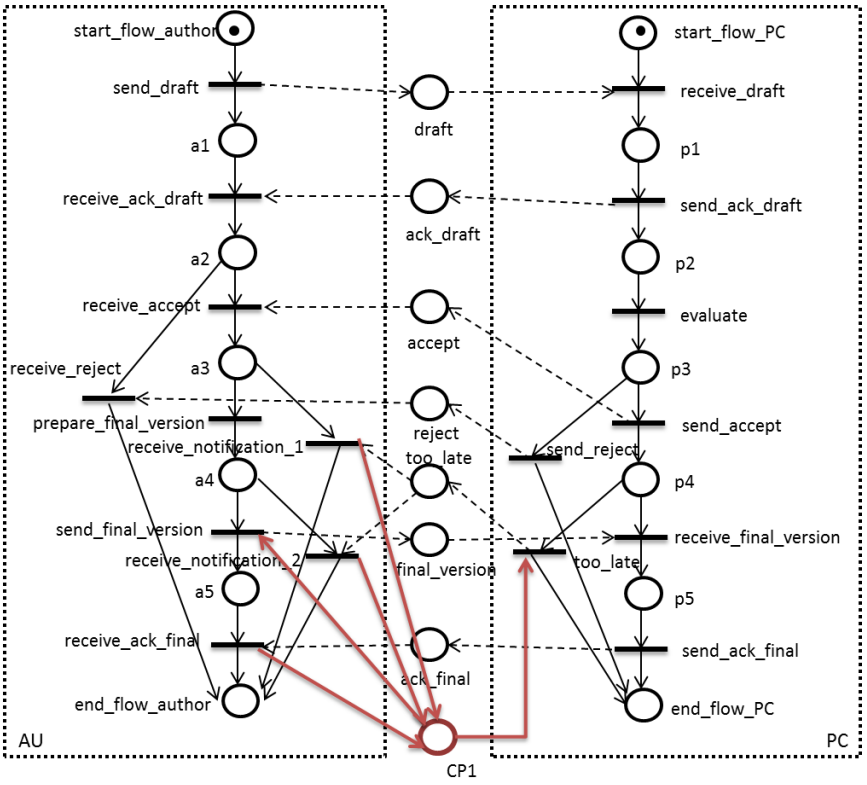

FIGURA 4.22: *IOWF-net* com lugar de controle encontrado nos sifões *S9* e *S23*.

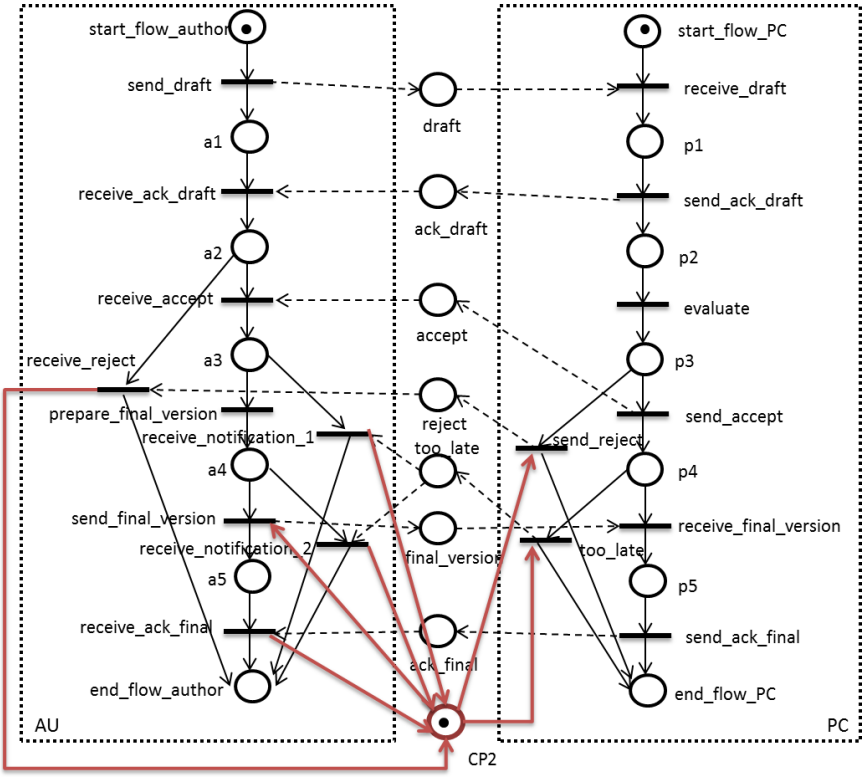

FIGURA 4.23: *IOWF-net* com lugar de controle encontrado no sifão *S21*.

Primeiramente analisaremos a *IOWF-net* controlada da Figura 4.22. Ao gerar o grafo de alcançabilidade desta rede (Figura 4.24), percebemos que a inclusão do lugar de controle, além de diminuir os estados acessíveis da rede, quando comparado à Figura 4.4 que mostra o grafo sem o lugar de controle, gerou outra situação de *deadlock* quando a rede alcança o estado *E13*, levando a uma nova situação de travamento onde nenhuma outra transição poderá ser disparada.

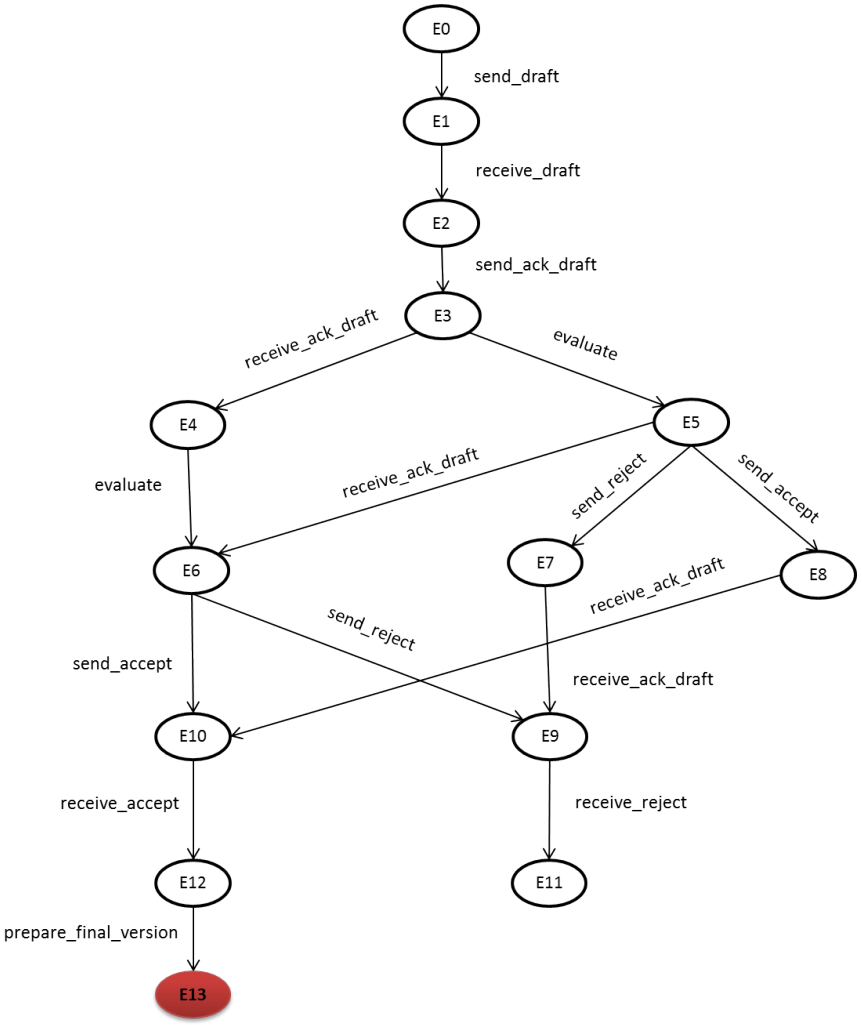

FIGURA 4.24: Grafo de alcançabilidade da Figura 4.22.

Considerando a marcação inicial da rede M_0 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0) representando, respectivamente, a marcação nos lugares: $start\_flow\_author$, $a1$, $a2$, $a3$, $a4$, $a5$, end_flow_author, $draft$, ack_draft, $accept$, $reject$, too_late, $final\_version$, ack_final, $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, end_flow_PC e CP, ao fazer a análise dos estados alcançáveis pela rede é perceptível a situação de *deadlock* quando a rede alcança o estado *E13*, pois a partir deste estado nenhum outro é alcançado. A marcação da rede no estado *E13* é dada pelo vetor de marcação M_{E13} = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0), ou seja, a situação de *deadlock* acontece quando há fichas nos lugares $a4$ e $p4$. Para alcançar o estado *E13* é preciso efetuar o disparo, nesta sequência, das seguintes transições: $send\_draft$, $receive\_draft$, $send\_ack\_draft$, $evaluate$, $receive\_ack\_draft$, $send\_accept$, $receive\_accept$ e $prepare\_final\_version$.

Identificado o *deadlock* e a marcação do sistema no momento em que a rede alcança o estado *E13*, podemos concluir:

- quando há fichas nos lugares $a4$ e $p4$ ao mesmo tempo, a rede está em situação de *deadlock*;
- o disparo sequencial das transições $prepare\_final\_version$ e $send\_accept$ é quem leva o sistema a alcançar tal travamento. Logo, deve haver um novo sifão na rede, que se torna vazio de fichas, ao disparar estas as transições.

Portanto, sabemos que a inclusão do lugar de controle *CP1* na rede irá gerar outra situação de *deadlock* e, consequentemente, um novo sifão que esvazia. Como temos outra opção de lugar de controle, iremos verificar se este lugar controla a rede sem gerar um novo *deadlock*. Se isso acontecer, podemos desconsiderar este lugar (*CP1*) inserido à Figura 4.22, pois encontramos outro lugar de controle (*CP2*) que conseguiu controlar a rede, no caso, o inserido na Figura 4.23. Se o lugar de controle encontrado por meio do controle do sifão *S21* quando inserido na rede também gerar outro *deadlock*, então um novo procedimento de controle da nova situação de *deadlock* gerada terá que ser executado.

Analisando agora a *IOWF-net* controlada da Figura 4.23, percebemos que o grafo de alcançabilidade que pode ser visualizado na Figura 4.25 não gera nenhuma outra situação de *deadlock* e que a situação de *deadlock* identificada previamente foi completamente controlada. Portanto, percebemos que a adição do lugar de controle *CP2* identificado por meio do controle do sifão *S21*, além de controlar a situação de *deadlock*, não gera nenhuma outra situação. Desta forma, podemos concluir que a rede está livre de *deadlocks*.

Como conseguimos um lugar de controle que controla a *IOWF-net*, o lugar de controle utilizado para controlar o sifão *S9* e *S23* pode ser descartado e não precisamos verificar a possibilidade de um novo controlador para deixar a rede livre destes travamentos. Portanto, apresentaremos na próxima seção a arquitetura distribuída livre de bloqueio para o lugar de controle *CP* adicionado à rede apresentado na Figura 4.23.

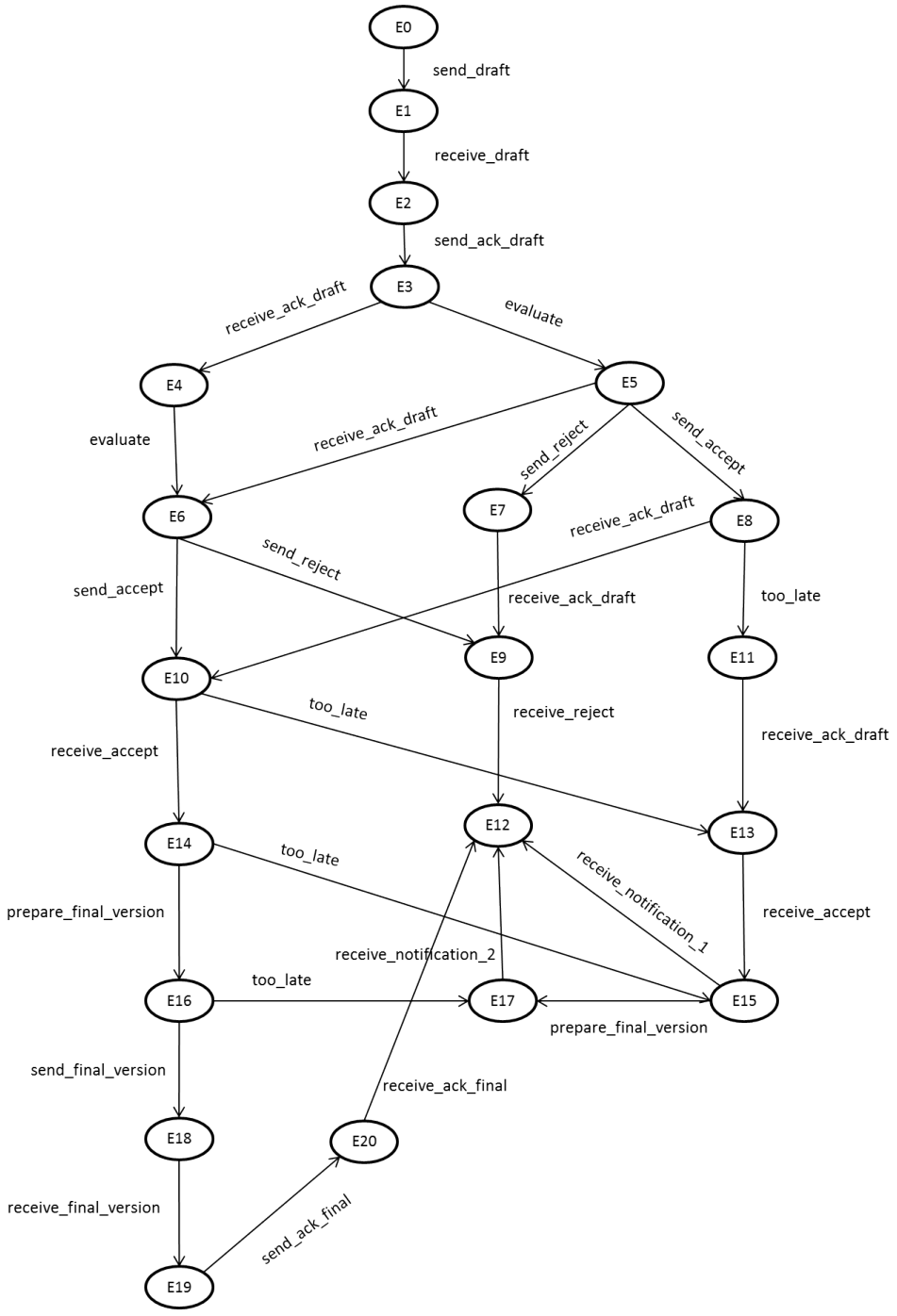

FIGURA 4.25: Grafo de alcançabilidade da Figura 4.23.

4.4 Modelo Aumentado da rede livre de *deadlock*

Uma das principais preocupações no momento da escolha da arquitetura de software a ser empregada na solução é garantir que o modelo atenda às especificações. Outro fator essencial é garantir que o sistema irá atender às necessidades associadas às *WorkFlow nets* interorganizacionais, sem que o processo seja degradado ou crie novas situações de

deadlock. Uma vez feita a análise para prevenção e correção de situações de *deadlock*, dentre os vários padrões de arquitetura de software destaca-se, para a arquitetura livre de *deadlock*, o estilo de arquitetura distribuída.

Neste padrão arquitetural, a responsabilidade pelo controle da execução dos casos fica disposta em processos distintos. Outra característica desta arquitetura é que o tempo de resposta a uma requisição é controlado individualmente. Por ser uma arquitetura aberta, flexível e escalável, possibilita a adição de novos recursos quando necessário, porém, para garantir um sistema livre de *deadlock*, todo o processo de especificação, modelagem e análise necessitará ser refeito.

No exemplo utilizado neste capítulo, após a análise do modelo e o controle supervisório da situação de *deaadlock*, o lugar de controle *CP2* apresentado na Figura 4.23, inserido para controlar tal situação, sempre será inicialmente marcado (conterá 1 ficha). Como consequência disto, o fato deste lugar conter marcação o impede de ser visto apenas como um lugar de comunicação assíncrona. Portanto, este lugar de controle será transformado em uma nova *WorkFlow net* que será acrescentada ao processo global da IOWF-net da Figura 4.3.

A introdução do lugar de controle dentro da estrutura interorganizacional pode ser vista como um conjunto adicional de restrições que podem ser traduzidas em mais um caso a ser tratado, e este caso irá pertencer a uma *WorkFlow-net* específica que será compartilhada entre as *LWF-nets* originais por intermédio de um meio de comunicação. A estrutura de controle da nova *WorkFlow net* irá reproduzir de forma acíclica (a fim de respeitar a estrutura acíclica de uma *WorkFlow net*) o comportamento dado pelo invariante de lugar que contém o lugar de controle *CP2*. Após a inserção do local de controle *CP2* na *IOWF-net* representada pela Figura 4.23, observa-se claramente que *CP2* pertence ao invariante de lugar mostrado na Figura 4.26.

A nova *WorkFlow net CWF* que será incluída no processo global é apresentada na Figura 4.27.

Como a *WorkFlow net* é uma estrutura acíclica, o novo processo representado na Figura 4.27 terá que respeitar esta característica, portanto terá um lugar de entrada marcado com nome *start_control* que possui um único arco de saída, o qual será direcionado para a transição *start*, e um lugar de saída denominado *end_control* com um único arco de entrada, no caso um arco de saída da transição $final$. As outras transições da nova *WorkFlow net* serão as mesmas que aparecem no invariante de lugar da Figura 4.26 e serão nomeadas: too_late', $receive\_notification'_1$, $receive\_notification\_2'$,

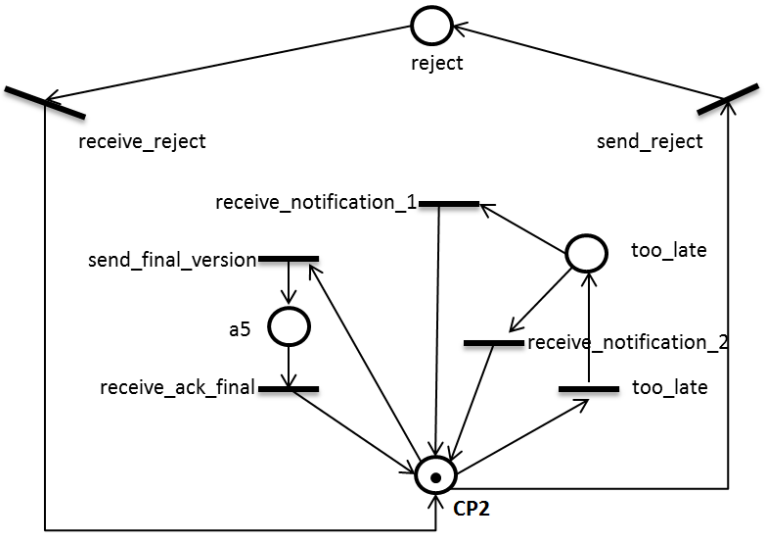

FIGURA 4.26: Invariante de Lugar com o lugar CP2.

$send\_final\_version'$, $receive\_ack\_final'$, $send\_reject$ e $receive\_reject$. O invariante de lugar foi destacado na Figura 4.27.

Na arquitetura proposta, o controle principal da execução está nas *LWF-nets*. Isto significa que o disparo das transições $send\_final\_version'$, too_late' e $send\_reject'$ está condicionado a uma solicitação vinda da *LWF-net*, a qual requisita o disparo de suas transições correspondentes, respectivamente, $send\_final\_version$, too_late e $send\_reject$. Por exemplo, se a *LWF-net PC* quiser disparar a transição $send\_reject$, terá que ser disparada a transição $t3$ que irá gerar uma ficha no lugar de espera *E3* e outra ficha que enviará a requisição que permitirá o disparo da transição $send\_reject'$ em *C3*. Se a transição $send\_reject'$ for disparada, uma ficha será consumida do lugar *CWF1*, impedindo que as transições $send\_final\_version'$ e too_late' sejam disparadas nesta execução; então duas fichas surgirão nos lugares *CWF4* e *C3'*. Assim, com uma ficha em *E3* e outra em *C3'*, a transição $send\_reject$ poderá ser disparada, produzindo uma ficha no lugar $reject$ e outra em end_flow_PC.

Outra situação passível de acontecimento surge após o disparo sequencial de $t1$ e $t2$, quando haverá uma ficha em *C1* e outra em *C2* e, consequentemente, em *E1* e *E2*. Neste caso, o controlador que deverá escolher qual transição disparar: $send\_final\_version'$ ou too_late'. Podemos considerar que a primeira solicitação que chegar será a atendida, visto que não haverá disparos simultâneos, apenas sequenciais. Se $send\_final\_version$ for disparada, uma ficha será consumida em *CWF1* e *C1*, e assim uma ficha surgirá no lugar *C1'*, permitindo o disparo da transição $send\_final\_version$. As fichas que estão em *C2* e *E2*, após aguardarem um tempo de espera $[\alpha,\alpha]$ atribuído à transição $t2'$, serão consumidas por esta transição e, após seu disparo, uma ficha será devolvida ao lugar $p4$.

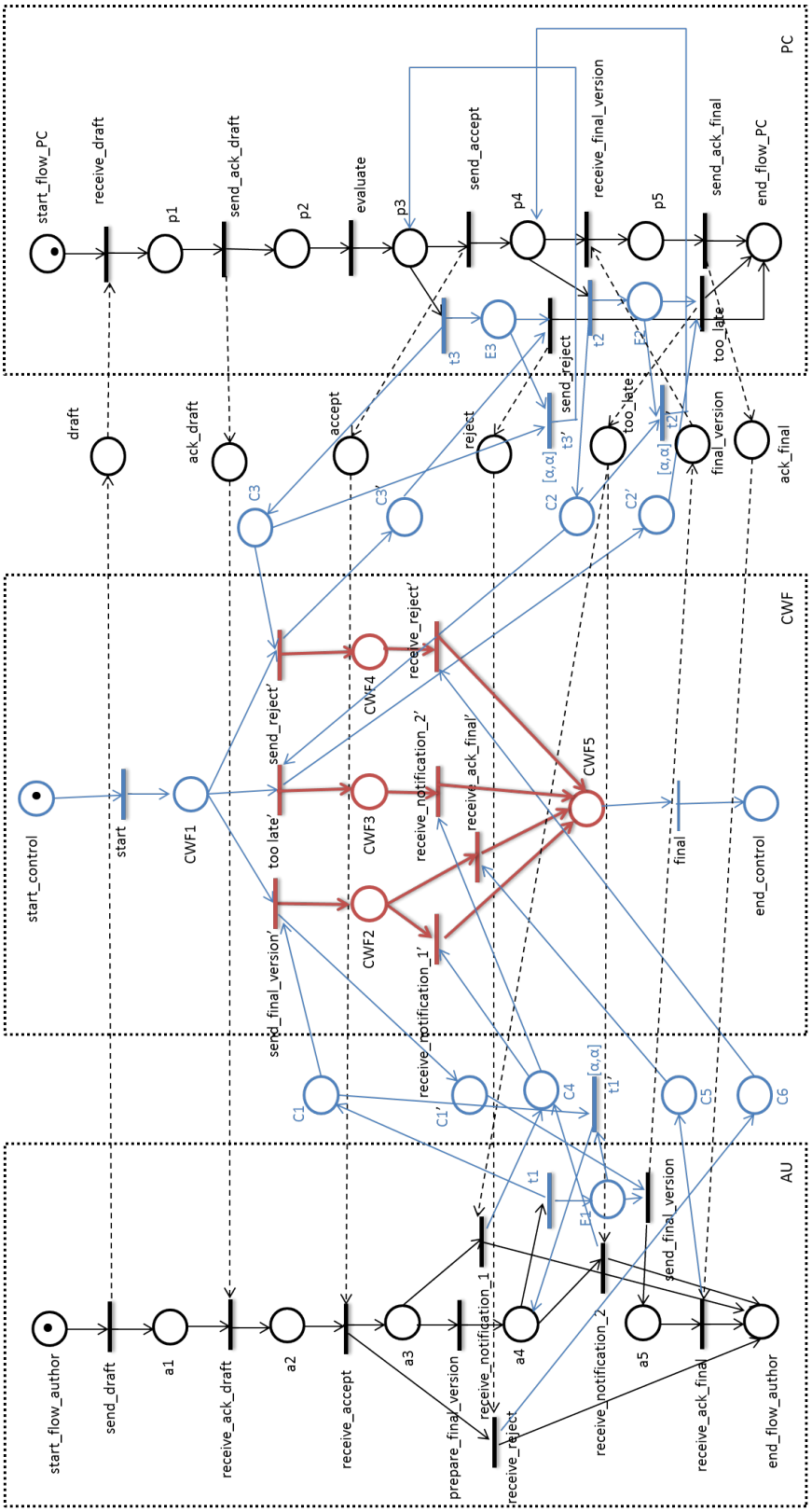

FIGURA 4.27: IOWF-net com a inclusão de CWF.

Com a ficha novamente no lugar $p4$, tanto a transição $t2$ pode ser disparada, quanto a execução da rede pode seguir para o disparo da transição $receive\_final\_version$.

Tendo a nova *IOWF-net* aumentada na Figura 4.27, vamos verificar novamente a propriedade *Soundness*. Para isto, faremos o mesmo processo realizado na Figura 4.3, na qual transformou-se a rede aumentada em uma *U(IOWF-net)*, com um lugar de entrada *i*, um lugar de saída *o*, uma transição inicial *ti* e a transição final *to*. A *U(IOWF-net)* da Figura 4.27 é apresentada na Figura 4.28.

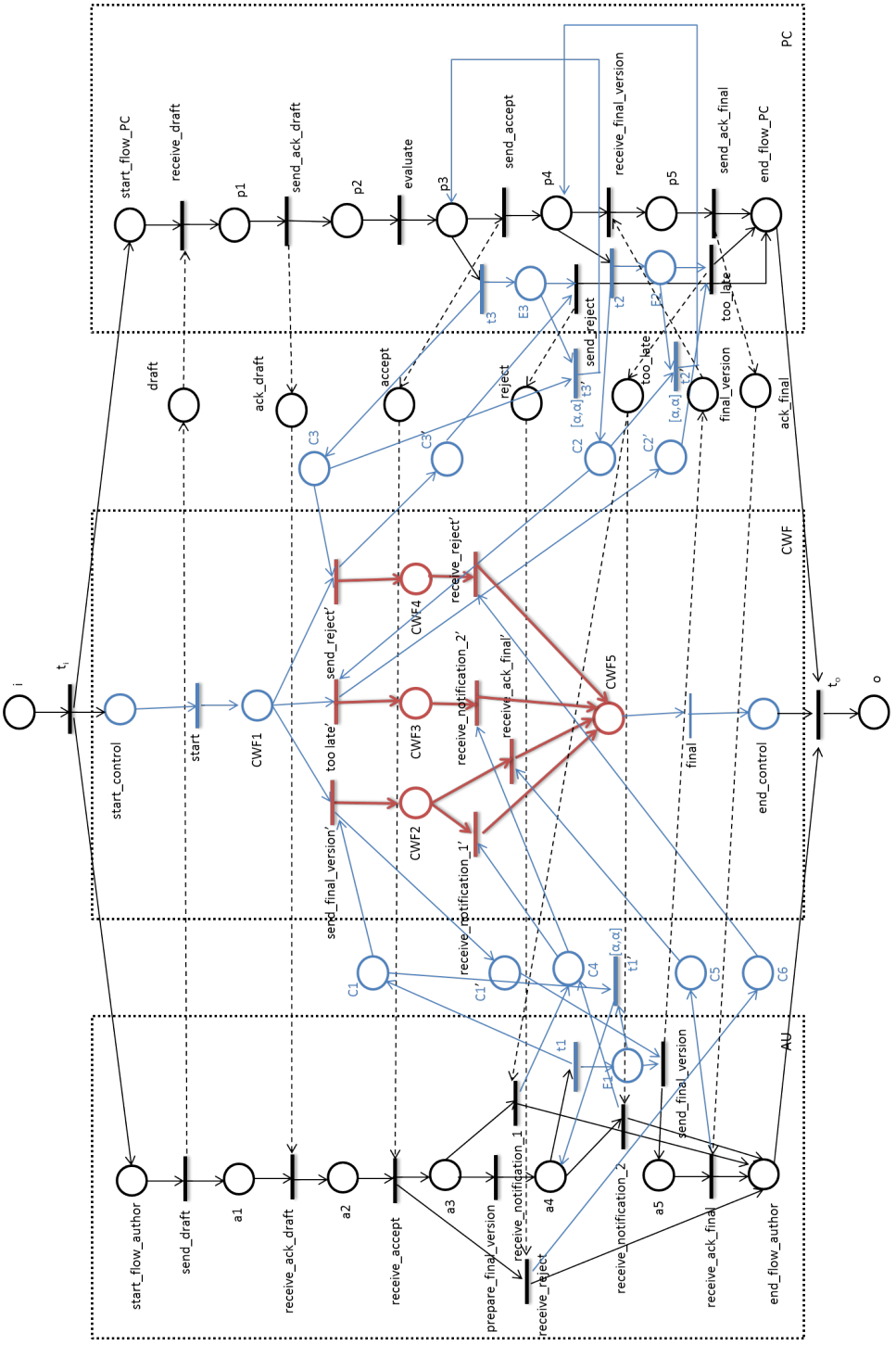

FIGURA 4.28: U(IOWF-net).

Sabendo que as *LWF-nets AU*, *PC* e *CWF* são localmente *Soundess* e que a *U(IOWF-net)* também é, finalmente, pode-se afirmar que a nova *IOWF-net* aumentada apresentada na Figura 4.27 se tornou localmente e globalmente *sound*, e o cenário que levava a situação de *deadlock* foi removido pela adição da nova *WorkFlow net CWF*. Como nenhum outro cenário de *deadlock* apareceu, podemos afirmar que este modelo aumentado da *IOWF-net* apresentado na Figura 4.2 está livre de situações de *deadlock*.

4.5 Resumo do método

Como primeiro passo do método foi encontrada a situação de *deadlock* e os sifões associados a esta situação. Em seguida, inseriu-se o controle supervisório nos sifões encontrados impedindo-os de ficarem vazios. Após isto, foi incluído o lugar de controle na *IOWF-net* original e validado se realmente controlou a situação de *deadlock* e se nenhuma outra situação foi gerada. Por fim, foi identificado o invariante de lugar gerado pelo acréscimo do lugar *CP2* para verificação de quais transições que fariam parte da nova arquitetura com a inserção da *WF-net* de controle no modelo inicial, transformando-o num modelo aumentado e controlado. Para garantir que tudo estivesse correto, ou seja, que a arquitetura fosse realmente livre de bloqueio, a propriedade *Soundness* e a vivacidade da rede foram verificadas.

Do método proposto, foi apresentado o procedimento de detecção de *deadlock* nas *WorkFlow nets* interorganizacionais e mostrado que estas situações de *deadlock* estão relacionadas às estruturas em forma de sifão que se esvaziam. Também foi apresentado o procedimento de controle supervisório de sifões para remover as situações de *deadlock* e a validação destes controladores quando inseridos no modelo inicial. Finalmente foi elaborado um modelo de arquitetura distribuída livre de *deadlock*.

Podemos concluir que a hipótese inicial foi validada. Ou seja, que as situações de *deadlock* estão relacionadas à presença de estruturas chamadas sifões que se esvaziam, ficando livre de fichas. Conclui-se também que o procedimento sistemático apresentado para a detecção e correção dessas situações nas *IOWF-nets* foi válido, e colaborou efetivamente para controlar a execução do modelo impedindo-o de alcançar os estados de total travamento. E por último, a construção da arquitetura distribuída por meio da inserção do controle com uma *WF-net* acrescentada ao modelo se mostrou eficiente e consideravelmente simples, o que facilita a utilização deste procedimento para outras *IOWF-net*.

Capítulo 5

Estudo de Caso

A ideia deste capítulo é apresentar a abordagem proposta no Capítulo 4 por meio de um estudo de caso aplicado a uma *WorkFlow net* interorganizacional (*IOWF-net*) com três processos locais.

5.1 Apresentação

A *IOWF-net* escolhida como estudo de caso é semelhante à utilizada para apresentar o método no capítulo anterior e se diferencia basicamente pelo acréscimo de uma terceira *WF-net* local (*LWF-net NP*), que se comunica por meio de lugares de comunicação assíncrona com a *LWF-net PC*. Outros lugares e transições também foram acrescentados no final de cada processo para receber a resposta da comunicação com *NP*. A *IOWF-net* que será abordada neste estudo de caso é apresentada na Figura 5.1.

Na Figura 5.1, a *LWF-net AU* representa o processo do autor ao realizar a submissão de um projeto para avaliação de um comitê avaliador. O processo do autor resume-se em enviar o projeto, aguardar a confirmação do recebimento do projeto pelo comitê avaliador e aguardar resposta da avaliação do projeto. Se aprovado, o autor deve preparar a versão final e enviar novamente ao comitê; se reprovado ou receber alguma notificação de rejeição, o autor deve aguardar até receber uma mensagem final do comitê e enfim, encerrar seu processo.

A *LWF-net PC* representa o processo do comitê avaliador. O processo do comitê compreende as atividades de receber o projeto, enviar o projeto para um avaliador externo, enviar uma notificação de recebimento ao autor, avaliar o projeto, comunicar ao autor e ao avaliador externo se o projeto foi aceito, rejeitado ou se não há mais

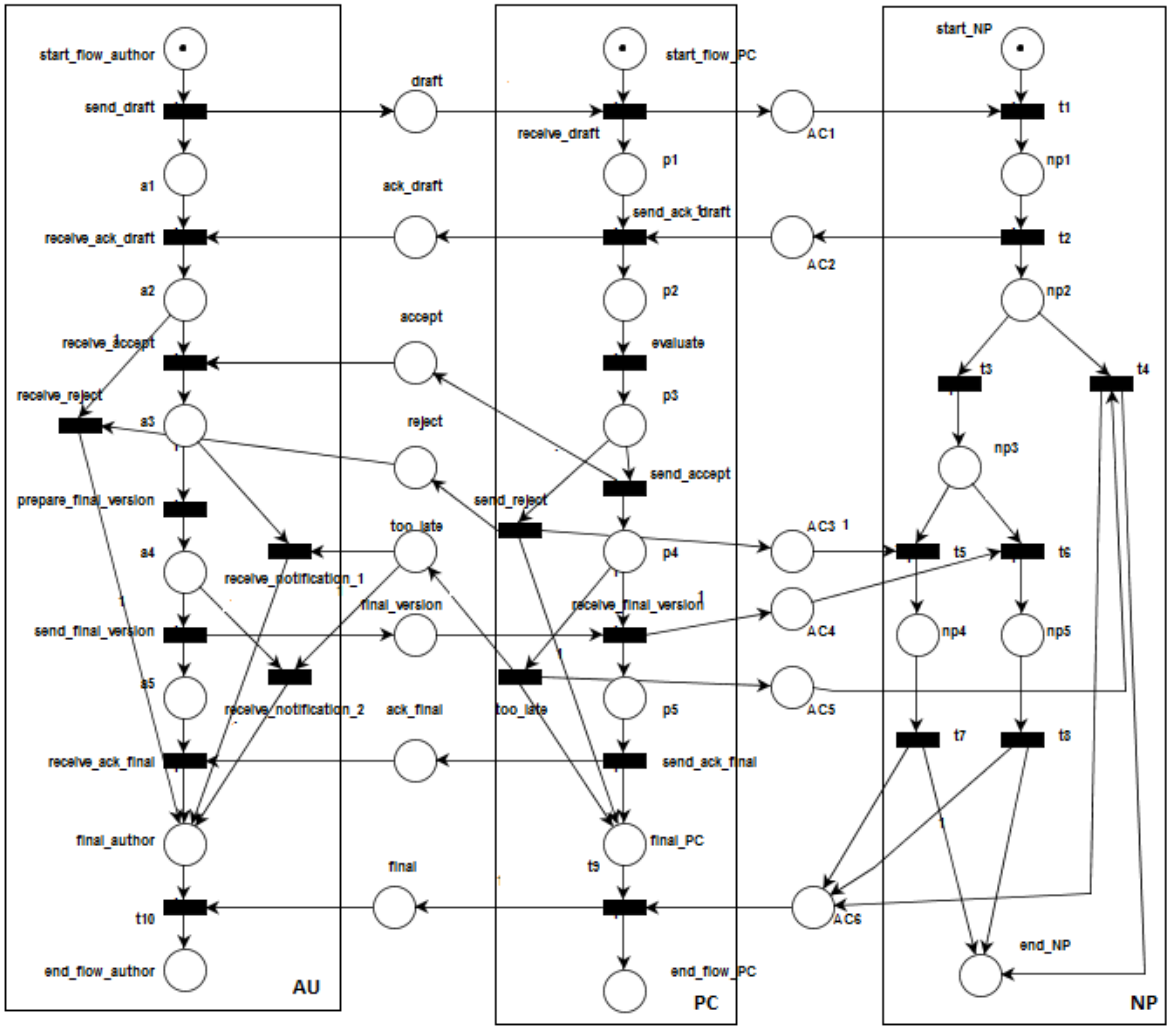

FIGURA 5.1: *IOWF-net* com três processos locais: *AU*, *PC* e *NP*.

tempo para publicá-lo, receber a versão final do autor, receber uma notificação final do avaliador externo e finalizar o processo.

O novo processo acrescentado ao se fazer uso da *LWF-net NP* pode ser interpretado como um complemento ao processo de avaliação. Por exemplo, uma avaliação externa contratada pelo comitê avaliador. Desta forma, quando o projeto é enviado por *AU* e recebido por *PC*, este o envia a *NP* por meio do lugar de comunicação assíncrono *AC*, que o analisa (*np1*) e devolve como resposta à *PC* a informação (*AC2*); por exemplo, uma informação que especifica se o projeto está nos padrões exigidos pela conferência. Após isso, *NP* fica aguardando o resultado da avaliação do projeto (*np2*) pelo comitê. Terminada a avaliação, se o projeto for rejeitado por *PC*, uma notificação é enviada ao autor (*reject*) e outra ao *NP* (*AC3*). Caso o projeto seja aceito e uma versão final for entregue ao *PC*, uma notificação é enviada a *NP* (*AC4*) junto à versão final do projeto. Se o projeto for recebido tardiamente e o comitê não for mais publicá-lo, *NP* é

notificado (*AC5*). *NP* confirma o recebimento das notificações a *PC* (*AC6*) e *PC* envia uma mensagem final a *AU* (*final*). O processo de *NP* se encerra então.

Neste estudo de caso, na IOWF-net abordada existem trocas de mensagens entre os três processos locais que a compõe e, além disto, existe uma dependência destas comunicações para que cada processo consiga executar seu fluxo. Trata-se de um cenário propício para a existência de situações de travamento, nas quais respostas aguardadas por um processo podem não chegar ou ficarem estagnadas dependendo da resposta de outro processo. Por exemplo, no nosso estudo de caso, o autor se encontra no estado *a1* onde aguarda uma confirmação de recebimento do projeto pelo comitê, porém o comitê (*PC*) só enviará essa resposta quando *NP* confirmá-lo mediante uma mensagem (*AC2*) informando que o projeto enviado pode ser avaliado. Portanto, o disparo da transição $receive\_ack\_draft$ está condicionado ao fato de *PC* receber primeiro uma resposta de *NP*, para que depois seja enviado por meio de ack_draft uma mensagem a *AU* permitindo o prosseguimento do fluxo de execução.

5.2 Verificação das propriedade Soundness e vivacidade

Considerando a *IOWF-net* apresentada na Figura 5.1, as *LWF-nets* (*AU*, *PC* e *NP*) que a constituem são *sound*, portanto podemos considerar o modelo apresentado como sendo localmente *sound*. Para verificar se a *IOWF-net* é globalmente *sound* vamos transformá-la em uma *WorkFlow net* acrescentando o lugar de início *i* com um arco saindo para a transição *ti* e um lugar de saída *o* com um arco de entrada vindo da transição *to*. A *IOWF-net* modificada, denominada em [Aalst 1998] como *U(IOWF-net)*, é apresentada na Figura 5.2.

Analisando o disparo das transições e a disposição das fichas na Figura 5.2 podemos perceber que ao disparar sequencialmente a transição *too_late* pertencente à *LWF-net PC* e a transição $send\_final\_version$ que pertence à *LWF-net AU*, o sistema estará em uma situação de *deadlock* na qual nenhuma outra transição poderá ser disparada. Assim, podemos concluir que a *U(IOWF-net)* não é *sound* e portanto a *IOWF-net* apresentada na Figura 5.1 não é globalmente *sound*. Outra situação que inviabiliza a *IOWF-net* de ser globalmente *sound* é mediante o disparo sequencial da transição *t3* pertencente à *LWF-net NP* e da transição *too_late* pertencente à *LWF-net PC*. Neste caso, teremos fichas em *np3*, mas com o disparo de *too_late* não teremos fichas em *AC3*

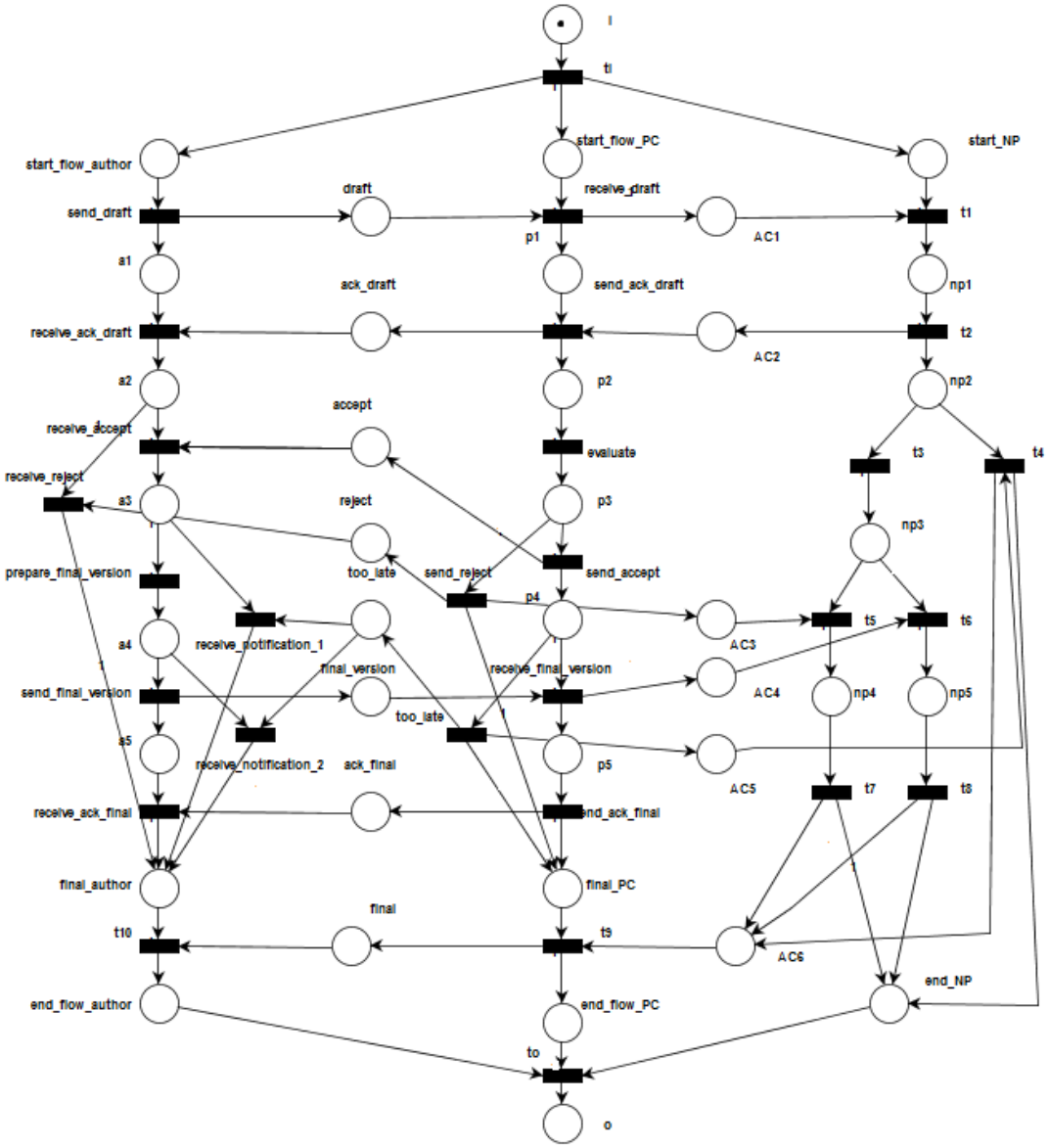

FIGURA 5.2: *U(IOWF-net)* da Figura 5.1.

e *AC4*, o que impedirá o disparo das transições *t5* e *t6*, levando a rede a outra situação de travamento.

Com a informação de que o modelo analisado não é globalmente *sound*, precisa-se verificar a vivacidade da rede. Caso a rede não seja viva, deve-se então identificar a presença da situação de *deadlock* por meio do grafo de alcançabilidade. A Figura 5.3 apresenta o grafo de alcançabilidade da *IOWF-net* da Figura 5.1.

Com: 1. $send\_draft$, 2. $receive\_ack\_draft$, 3. $receive\_accept$, 4. $prepare\_final\_version$, 5. $send\_final\_version$, 6. $receive\_ack\_final$, 7. $t10$, 8. $receive\_reject$, 9. $receive\_no- tification\_1$, 10. $receive\_notification\_2$,

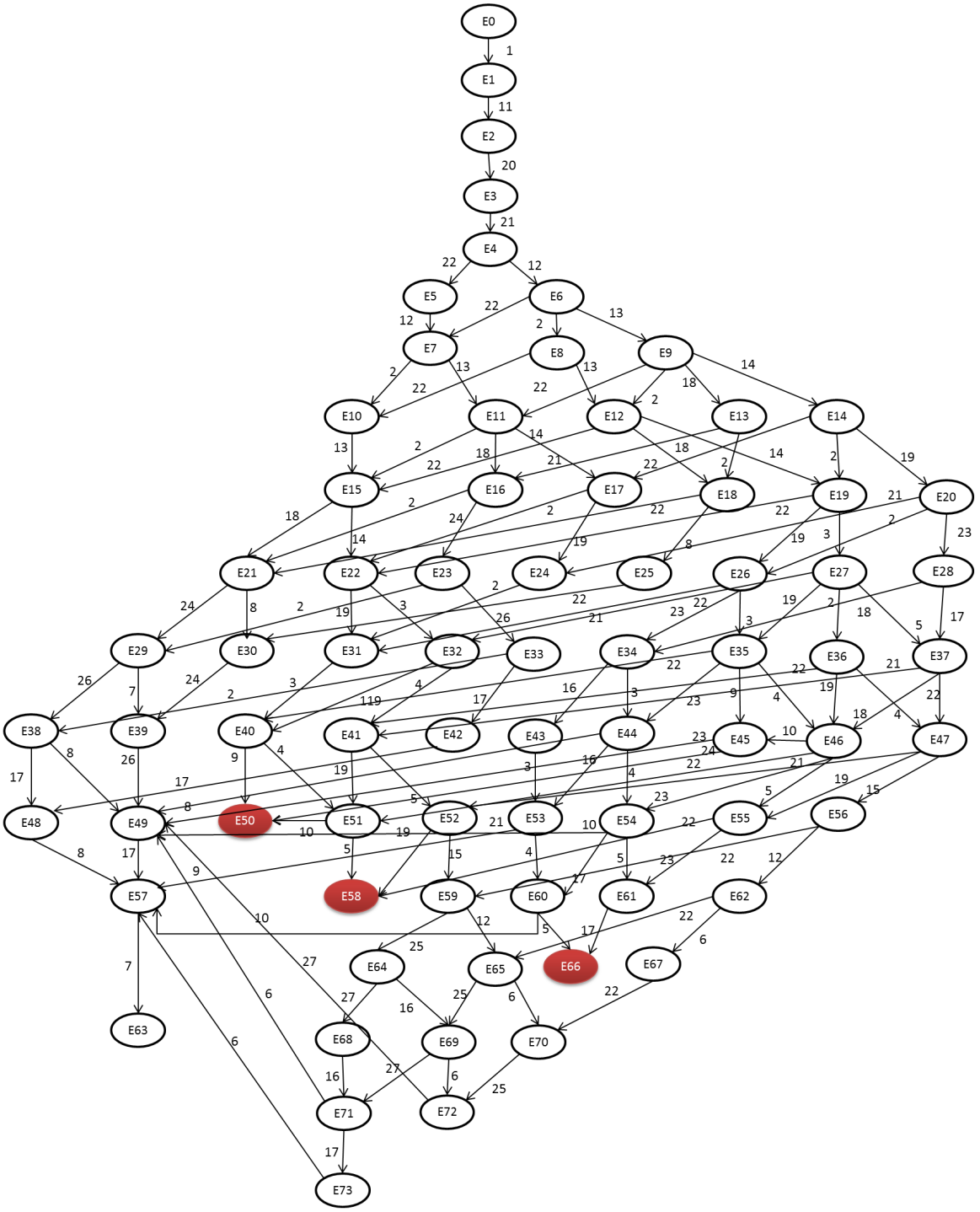

FIGURA 5.3: Grafo de alcançabilidade da Figura 5.1.

11. *receive_draft*, 12. *send_ack_draft*, 13. *evaluate*, 14. *send_accept*, 15. *receive_final_version*, 16. *send_ack_final*, 17. *t*9, 18. *send_reject*, 19. *too_late*, 20. *t*1, 21. *t*2, 22. *t*3, 23. *t*4, 24. *t*5, 25. *t*6, 26. *t*7, 27. *t*8.

Considerando o vetor da marcação inicial da rede gerado pelo software *PIPE* como sendo M_0 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0), que representa, respectivamente, a marcação nos lugares: *start_flow_author*, *a1*, *a2*, *a3*, *a4*, *a5*, *final_author*, *end_flow_author*, *draft*, *ack_draft*, *accept*, *reject*, *too_late*, *final_version*, *ack_final*, *final*, *start_flow_PC*, *p1*, *p2*, *p3*, *p4*, *p5*, *final_PC*, *end_flow_PC*, *AC1*, *AC2*, *AC3*, *AC4*, *AC5*, *AC6*, *start_NP*, *np1*, *np2*, *np3*, *np4*, *np5* e *end_NP*, ao analisar os estados alcançáveis pela rede é perceptível a situação de *deadlock* quando a rede alcança os estados *E50*, *E58* e *E66*. Quando o sistema alcança o estado *E50* sua marcação é M_{50} = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0), portanto, a situação de *deadlock* acontece quando há simultaneamente fichas em *AC5*, *final_author*, *final_PC* e *np3*. No estado *E58* temos o vetor marcação M_{58} = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0), assim caracterizando a situação de travamento com fichas em *a5*, *too_late*, *final_version*, *final_PC*, *AC5* e *np3*. Por fim, no estado *E66* temos o vetor marcação M_{66} = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) que apresenta a marcação da situação de *deadlock* na qual a rede está com fichas em *a5*, *too_late*, *final_version*, *final*, *end_flow_PC* e *end_NP*.

Identificadas as situações de *deadlock* e a marcação do sistema ao alcançar os estados de travamento, podemos concluir que:

- a rede fica em total travamento, caracterizando a situação de *deadlock*, quando alcança os estados *E50*, *E58* e *E66*. Portanto, o objetivo é controlar a rede para que estes estados não sejam alcançados;
- o disparo sequencial das transições *t3* e *too_late* é o que leva o sistema a alcançar o estado *E58*. Para alcançar o estado *E50* é necessário o disparo sequencial das transições *receive_notification_1* ou *receive_notification_2* e *t3*. Já o estado *E66* é alcançado por meio do disparo sequencial de *send_final_version* e *too_late*. Assim, devem ser encontrados sifões que se esvaziam mediante o disparo destas transições, caracterizando as situações de *deadlock* que necessitam de controle.

5.3 Detecção dos Sifões

Para encontrar os sifões, a rede da Figura 5.1 foi modificada de forma a se tornar repetitiva. Foi inserida à rede uma nova transição *SR*, que possui arcos de entrada que também são arcos de saída dos lugares end_flow_author, end_flow_PC e end_NP, e arcos de saída que são entradas nos lugares $start\_flow\_author$, $start\_flow\_PC$ e $start\_NP$. A *IOWF-net* repetitiva é apresentada na figura 5.4.

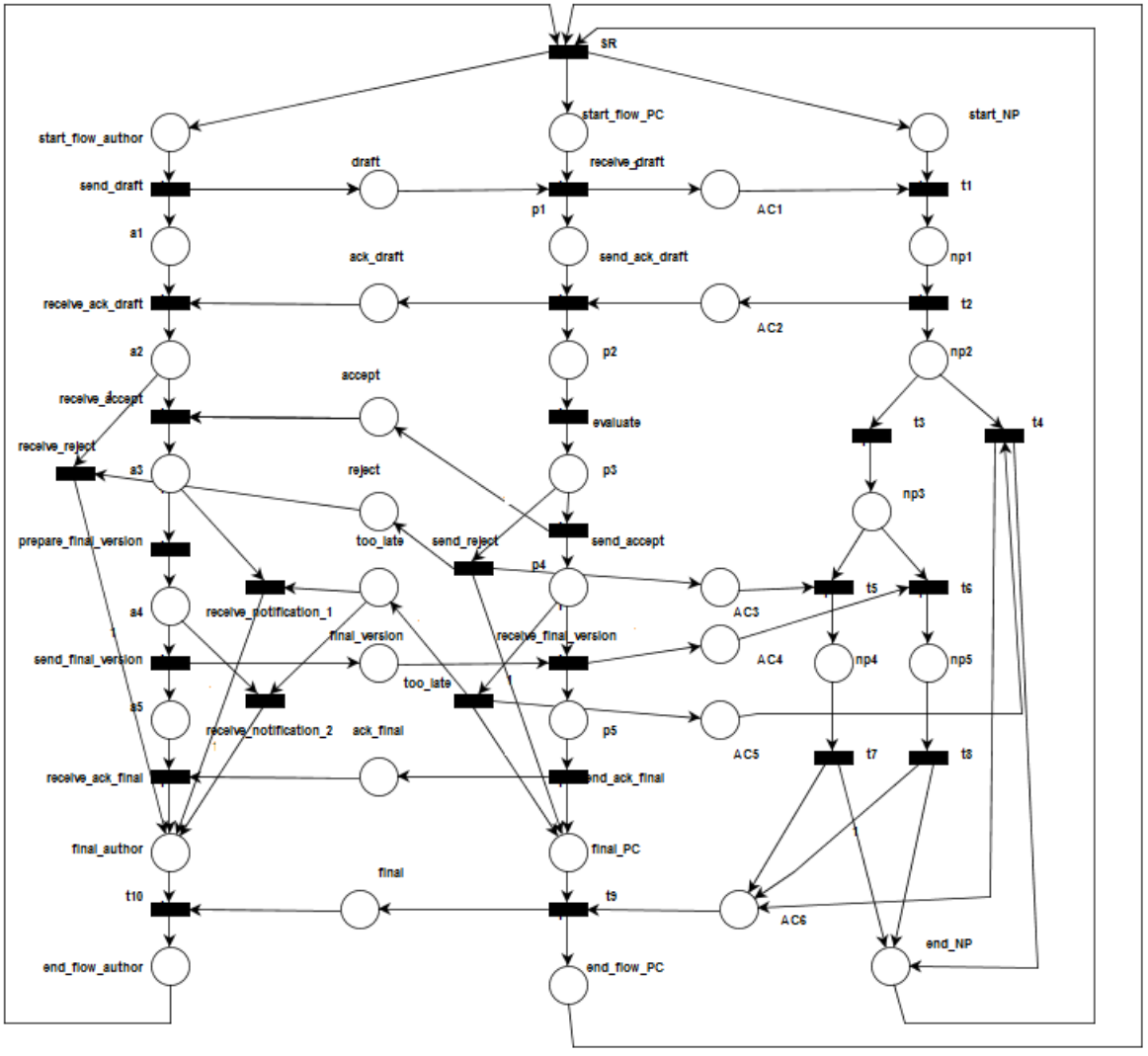

FIGURA 5.4: IOWF-net repetitiva.

Utilizando o *PIPE* e a rede apresentada na Figura 5.4, é possível encontrar os sifões apresentados na Tabela 5.1. A marcação com "x" na coluna *Trap* indica que o sifão possui uma *Trap*.

Tabela 5.1: Sifões e Traps da Figura 5.4.

ID	Sifão	Trap
S1	*end_NP, np1, np2, np3, np4, np5, start_NP*	x
S2	*end_flow_PC, final_PC, p1, p2, p3, p4, p5, start_flow_PC*	x
S3	*ack_final, end_flow_author, final_author, p1, p2, p3, p4, p5,* reject, *start_flow_PC, too_late*	x
S4	*a3, a4, a5,* accept, *end_flow_author, final_author, p1, p2, p3,* reject, *start_flow_PC*	x
S5	*end_flow_author,* final, *final_PC, p1, p2, p3, p4, p5, start_flow_PC*	x
S6	*AC6, end_flow_author,* final, *np1, np2, np3, np4, np5, start_NP*	x
S7	*AC3, AC4, AC5, AC6, end_flow_author,* final, *np4, np5, p1, p2, p3, p4, start_flow_PC*	x
S8	*AC3, AC4, AC5, AC6,* draft, *end_flow_author,* final, *np4, np5, p1, p2, p3, p4, start_flow_author*	x
S9	*AC2, AC3, AC4, AC5, AC6, end_flow_author,* final, *np1, np4, np5, p2, p3, p4, start_NP*	x
S10	*AC1, AC2, AC3, AC4, AC5, AC6, end_flow_author,* final, *np1, np4, np5, p2, p3, p4, start_flow_PC*	x
S11	*AC1, AC2, AC3, AC4, AC5, AC6,* draft, *end_flow_author,* final, *np1, np4, np5, p2, p3, p4, start_flow_author*	x
S12	*AC3, AC4, AC6, end_flow_author,* final, *np1, np2, np4, np5, p1, p2, p3, p4, start_flow_PC, start_NP*	
S13	*AC2, AC3, AC4, AC6, end_flow_author,* final, *np1, np2, np4, np5, p2, p3, p4, start_NP*	
S14	*AC1, AC2, AC3, AC4, AC6, end_flow_author,* final, *np1, np2, np4, np5, p2, p3, p4, start_flow_PC*	
S15	*AC1, AC2, AC3, AC4, AC6,* draft, *end_flow_author,* final, *np1, np2, np4, np5, p2, p3, p4, start_flow_author*	
S16	*AC3, AC4, AC6,* draft, *end_flow_author,* final, *np1, np2, np4, np5, p1, p2, p3, p4, start_flow_author, start_NP*	
S17	*AC1, AC3, AC4, AC6,* draft, *end_flow_author,* final, *np1, np2, np4, np5, p1, p2, p3, p4, start_flow_author*	
S18	*AC1, AC3, AC4, AC6, end_flow_author,* final, *np1, np2, np4, np5, p1, p2, p3, p4, start_flow_PC*	
S19	*AC1, AC6, end_flow_author,* final, *np1, np2, np3, np4, np5, start_flow_PC*	x
S20	*AC1, AC6,* draft, *end_flow_author,* final, *np1, np2, np3, np4, np5, start_flow_author*	x
S21	*AC2, end_flow_author,* final, *final_PC, np1, p2, p3, p4, p5, start_NP*	x

S22	*AC1, AC2, end_flow_author, final, final_PC, np1, p2, p3, p4, p5, start_flow_PC*	x
S23	*AC1, AC2, draft, end_flow_author, final, final_PC, np1, p2, p3, p4, p5, start_flow_author*	x
S24	*draft, end_flow_author, final, final_PC, p1, p2, p3, p4, p5, start_flow_author*	x
S25	*a1, a2, a3, a4, AC2, AC3, AC4, AC6, end_flow_author, final, final_version, np1, np2, np4, np5, p2, p3, start_flow_author, start_NP*	
S26	*a1, a2, a3, a4, AC1, AC2, AC3, AC4, AC6, end_flow_author, final, final_version, np1, np2, np4, np5, p2, p3, start_flow_author, start_flow_PC*	
S27	*a1, a2, a3, a4, AC1, AC2, AC3, AC4, AC6, draft, end_flow_author, final, final_version, np1, np2, np4, np5, p2, p3, start_flow_author*	
S28	*a1, a2, a3, a4, AC3, AC4, AC6, end_flow_author, final, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_flow_PC, start_NP*	
S29	*a1, a2, a3, a4, AC3, AC4, AC6, draft, end_flow_author, final, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S30	*a1, a2, a3, a4, AC1, AC3, AC4, AC6, draft, end_flow_author, final, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S31	*a1, a2, a3, a4, AC1, AC3, AC4, AC6, end_flow_author, final, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_flow_PC*	
S32	*a2, a3, a4, AC2, AC3, AC4, AC6, ack_draft, end_flow_author, final, final_version, np1, np2, np4, np5, p2, p3, start_NP*	
S33	*a2, a3, a4, AC1, AC2, AC3, AC4, AC6, ack_draft, end_flow_author, final, final_version, np1, np2, np4, np5, p2, p3, start_flow_PC*	
S34	*a2, a3, a4, AC1, AC2, AC3, AC4, AC6, ack_draft, draft, end_flow_author, final, final_version, np1, np2, np4, np5, p2, p3, start_flow_author*	
S35	*a2, a3, a4, AC3, AC4, AC6, ack_draft, end_flow_author, final, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC, start_NP*	
S36	*a2, a3, a4, AC3, AC4, AC6, ack_draft, draft, end_flow_author, final, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	

S37	*a2*, *a3*, *a4*, *AC1*, *AC3*, *AC4*, *AC6*, *ack_draft*, draft, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *start_flow_author*	
S38	*a2*, *a3*, *a4*, *AC1*, *AC3*, *AC4*, *AC6*, *ack_draft*, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *start_flow_PC*	
S39	*a3*, *a4*, *AC2*, *AC3*, *AC4*, *AC6*, accept, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_NP*	
S40	*a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, accept, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_PC*	
S41	*a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, accept, draft, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_author*	
S42	*a3*, *a4*, *AC3*, *AC4*, *AC6*, accept, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *start_flow_PC*, *start_NP*	
S43	*a3*, *a4*, *AC3*, *AC4*, *AC6*, accept, draft, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *start_flow_author*, *start_NP*	
S44	*a3*, *a4*, *AC1*, *AC3*, *AC4*, *AC6*, accept, draft, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *start_flow_author*	
S45	*a3*, *a4*, *AC1*, *AC3*, *AC4*, *AC6*, accept, *end_flow_author*, final, *final_version*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *start_flow_PC*	
S46	*a2*, *a3*, *a4*, *a5*, *ack_draft*, *end_flow_author*, *final_author*, *p1*, *start_flow_PC*	x
S47	*a1*, *a2*, *a3*, *a4*, *a5*, *end_flow_author*, *final_author*, *start_flow_author*	x
S48	*a2*, *a3*, *a4*, *a5*, *AC2*, *ack_draft*, *end_flow_author*, *final_author*, *np1*, *start_NP*	x
S49	*a2*, *a3*, *a4*, *a5*, *AC1*, *AC2*, *ack_draft*, *end_flow_author*, *final_author*, *np1*, *start_flow_PC*	x
S50	*a2*, *a3*, *a4*, *a5*, *AC1*, *AC2*, *ack_draft*, draft, *end_flow_author*, *final_author*, *np1*, *start_flow_author*	x
S51	*a2*, *a3*, *a4*, *a5*, *ack_draft*, draft, *end_flow_author*, *final_author*, *p1*, *start_flow_author*	x
S52	*a3*, *a4*, *a5*, *AC2*, accept, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *reject*, *start_NP*	x
S53	*a3*, *a4*, *a5*, *AC1*, *AC2*, accept, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *reject*, *start_flow_PC*	x
S54	*a3*, *a4*, *a5*, *AC1*, *AC2*, accept, draft, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *reject*, *start_flow_author*	x
S55	*a3*, *a4*, *a5*, accept, draft, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *reject*, *start_flow_author*	x

S56	*AC2*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_NP*, *too_late*	x
S57	*a1*, *a2*, *a3*, *a4*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *p5*, *start_flow_author*	x
S58	*a2*, *a3*, *a4*, *AC2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *np1*, *p5*, *start_NP*	x
S59	*a2*, *a3*, *a4*, *AC1*, *AC2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *np1*, *p5*, *start_flow_PC*	x
S60	*a2*, *a3*, *a4*, *AC1*, *AC2*, *ack_draft*, *ack_final*, *draft*, *end_flow_author*,*final_author*, *final_version*, *np1*, *p5*, *start_flow_author*	x
S61	*a3*, *a4*, *AC2*, *accept*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *np1*, *p2*, *p3*, *p5*, *reject*, *start_NP*	x
S62	*a3*, *a4*, *AC1*, *AC2*, *accept*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *np1*, *p2*, *p3*, *p5*, *reject*, *start_flow_PC*	x
S63	*a3*, *a4*, *AC1*, *AC2*, *accept*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *final_version*, *np1*, *p2*, *p3*, *p5*, *reject*, *start_flow_author*	x
S64	*a2*, *AC2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_NP*, *too_late*	
S65	*a1*, *a2*, *AC2*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *start_NP*, *too_late*	
S66	*a1*, *a2*, *AC1*, *AC2*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *start_flow_PC*, *too_late*	
S67	*a1*, *a2*, *AC1*, *AC2*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *too_late*	
S68	*a1*, *a2*, *a3*, *a4*, *AC1*, *AC2*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*	
S69	*a1*, *a2*, *a3*, *a4*, *AC1*, *AC2*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *start_flow_PC*	
S70	*a1*, *a2*, *a3*, *a4*, *AC2*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *start_NP*	
S71	*a2*, *AC1*, *AC2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_PC*, *too_late*	
S72	*a2*, *AC1*, *AC2*, *ack_draft*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *too_late*	
S73	*a2*, *a3*, *a4*, *AC1*, *AC2*, *ack_draft*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*	
S74	*a2*, *a3*, *a4*, *AC1*, *AC2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_flow_PC*	

S75	*a2*, *a3*, *a4*, *AC2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *start_NP*	
S76	*AC1*, *AC2*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_flow_PC*, *too_late*	x
S77	*AC1*, *AC2*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_flow_author*, *too_late*	x
S78	*a3*, *a4*, *AC1*, *AC2*, *accept*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_flow_author*	
S79	*a3*, *a4*, *AC1*, *AC2*, *accept*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_flow_PC*	
S80	*a3*, *a4*, *AC2*, *accept*, *ack_final*, *end_flow_author*, *final_author*, *np1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_NP*	
S81	*a2*, *a3*, *a4*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *p1*, *p5*, *start_flow_PC*	x
S82	*a2*, *a3*, *a4*, *ack_draft*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *final_version*, *p1*, *p5*, *start_flow_author*	x
S83	*a3*, *a4*, *accept*, *ack_final*, *end_flow_author*, *final_author*, *final_version*, *p1*, *p2*, *p3*, *p5*, *reject*, *start_flow_PC*	x
S84	*a3*, *a4*, *accept*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *final_version*, *p1*, *p2*, *p3*, *p5*, *reject*, *start_flow_author*	x
S85	*a2*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_PC*, *too_late*	
S86	*a1*, *a2*, *ack_final*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *start_flow_PC*, *too_late*	
S87	*a1*, *a2*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *too_late*	
S88	*a1*, *a2*, *a3*, *a4*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*	
S89	*a1*, *a2*, *a3*, *a4*, *ack_final*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *start_flow_PC*	
S90	*a2*, *ack_draft*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*, *too_late*	
S91	*a2*, *a3*, *a4*, *ack_draft*, *ack_final*, *draft*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_author*	
S92	*a2*, *a3*, *a4*, *ack_draft*, *ack_final*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *start_flow_PC*	
S93	*ack_final*, *draft*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, *reject*, *start_flow_author*, *too_late*	x

S94	*a3*, a4, accept, *ack_final*, draft, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, reject, *start_flow_author*	
S95	*a3*, a4, accept, *ack_final*, *end_flow_author*, *final_author*, *p1*, *p2*, *p3*, *p4*, *p5*, reject, *start_flow_PC*	
S96	*AC6*, *end_flow_PC*, *np1*, *np2*, *np3*, *np4*, *np5*, *start_NP*	x
S97	*AC3*, *AC4*, *AC5*, *AC6*, *end_flow_PC*, *np4*, *np5*, *p1*, *p2*, *p3*, *p4*, *start_flow_PC*	x
S98	*AC3*, *AC4*, *AC5*, *AC6*, draft, *end_flow_PC*, *np4*, *np5*, *p1*, *p2*, *p3*, *p4*, *start_flow_author*	x
S99	*AC2*, *AC3*, *AC4*, *AC5*, *AC6*, *end_flow_PC*, *np1*, *np4*, *np5*, *p2*, *p3*, *p4*, *start_NP*	x
S100	*AC1*, *AC2*, *AC3*, *AC4*, *AC5*, *AC6*, *end_flow_PC*, *np1*, *np4*, *np5*, *p2*, *p3*, *p4*, *start_flow_PC*	x
S101	*AC1*, *AC2*, *AC3*, *AC4*, *AC5*, *AC6*, draft, *end_flow_PC*, *np1*, *np4*, *np5*, *p2*, *p3*, *p4*, *start_flow_author*	x
S102	a*AC3*, *AC4*, *AC6*, *end_flow_PC*, *np1*, *np2*, *np4*, *np5*, *p1*, *p2*, *p3*, *p4*, *start_flow_PC*, *start_NP*	
S103	*AC2*, *AC3*, *AC4*, *AC6*, *end_flow_PC*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *p4*, *start_NP*	
S104	*a3*, *a4*, *AC2*, *AC3*, *AC4*, *AC6*, accept, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_NP*	
S105	*a2*, *a3*, *a4*, *AC2*, *AC3*, *AC4*, *AC6*, *ack_draft*, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_NP*	
S106	*a1*, *a2*, *a3*, *a4*, *AC2*, *AC3*, *AC4*, *AC6*, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_author*, *start_NP*	
S107	*a1*, *a2*, *a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_author*, *start_flow_PC*	
S108	*a1*, *a2*, *a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, draft, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_author*	
S109	*AC2*, *AC3*, *AC4*, *AC5*, *AC6*, *end_flow_PC*, *np1*, *np4*, *np5*, *p2*, *p3*, *p4*, *start_NPa2*, *a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, *ack_draft*, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_PC*	
S110	*a2*, *a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, *ack_draft*, draft, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_author*	
S111	*a3*, *a4*, *AC1*, *AC2*, *AC3*, *AC4*, *AC6*, accept, *end_flow_PC*, *final_version*, *np1*, *np2*, *np4*, *np5*, *p2*, *p3*, *start_flow_PC*	

S112	*a3, a4, AC1, AC2, AC3, AC4, AC6, accept, draft, end_flow_PC, final_version, np1, np2, np4, np5, p2, p3, start_flow_author*	
S113	*AC1, AC2, AC3, AC4, AC6, end_flow_PC, np1, np2, np4, np5, p2, p3, p4, start_flow_PC*	
S114	*AC1, AC2, AC3, AC4, AC6, draft, end_flow_PC, np1, np2, np4, np5, p2, p3, p4, start_flow_author*	
S115	*a3, a4, AC3, AC4, AC6, accept, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC, start_NP*	
S116	*a2, a3, a4, AC3, AC4, AC6, ack_draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC, start_NP*	
S117	*a1, a2, a3, a4, AC3, AC4, AC6, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_flow_PC, start_NP*	
S118	*a1, a2, a3, a4, AC3, AC4, AC6, draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S119	*a1, a2, a3, a4, AC1, AC3, AC4, AC6, draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S120	*a1, a2, a3, a4, AC1, AC3, AC4, AC6, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_flow_PC*	
S121	*a2, a3, a4, AC3, AC4, AC6, ack_draft, draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S122	*a2, a3, a4, AC1, AC3, AC4, AC6, ack_draft, draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S123	*a2, a3, a4, AC1, AC3, AC4, AC6, ack_draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC*	
S124	*a3, a4, AC3, AC4, AC6, accept, draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S125	*a3, a4, AC1, AC3, AC4, AC6, accept, draft, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S126	*a3, a4, AC1, AC3, AC4, AC6, accept, end_flow_PC, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC*	
S127	*AC3, AC4, AC6, draft, end_flow_PC, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_author, start_NP*	
S128	*AC1, AC3, AC4, AC6, draft, end_flow_PC, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_author*	
S129	*AC1, AC3, AC4, AC6, end_flow_PC, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_PC*	
S130	*AC1, AC6, end_flow_PC, np1, np2, np3, np4, np5, start_flow_PC*	x

S131	*AC1, AC6, draft, end_flow_PC, np1, np2, np3, np4, np5, start_flow_author*	x
S132	*AC2, end_flow_PC, final_PC, np1, p2, p3, p4, p5, start_NP*	x
S133	*AC1, AC2, end_flow_PC, final_PC, np1, p2, p3, p4, p5, start_flow_PC*	x
S134	*AC1, AC2, draft, end_flow_PC, final_PC, np1, p2, p3, p4, p5, start_flow_author*	x
S135	*draft, end_flow_PC, final_PC, p1, p2, p3, p4, p5, start_flow_author*	x
S136	*AC3, AC4, AC5, end_NP, np4, np5, p1, p2, p3, p4, start_flow_PC*	x
S137	*AC3, AC4, AC5, draft, end_NP, np4, np5, p1, p2, p3, p4, start_flow_author*	x
S138	*AC2, AC3, AC4, AC5, end_NP, np1, np4, np5, p2, p3, p4, start_NP*	x
S139	*AC1, AC2, AC3, AC4, AC5, end_NP, np1, np4, np5, p2, p3, p4,start_flow_PC*	x
S140	*AC1, AC2, AC3, AC4, AC5, draft, end_NP, np1, np4, np5, p2, p3, p4, start_flow_author*	x
S141	*AC3, AC4, end_NP, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_PC, start_NP*	
S142	*AC2, AC3, AC4, end_NP, np1, np2, np4, np5, p2, p3, p4, start_NP*	
S143	*a3, a4, AC2, AC3, AC4, accept, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_NP*	
S144	*a2, a3, a4, AC2, AC3, AC4, ack_draft, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_NP*	
S145	*a1, a2, a3, a4, AC2, AC3, AC4, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_author, start_NP*	
S146	*a1, a2, a3, a4, AC1, AC2, AC3, AC4, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_author, start_flow_PC*	
S147	*a1, a2, a3, a4, AC1, AC2, AC3, AC4, draft, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_author*	
S148	*a2, a3, a4, AC1, AC2, AC3, AC4, ack_draft, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_PC*	
S149	*a2, a3, a4, AC1, AC2, AC3, AC4, ack_draft, draft, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_author*	
S150	*a3, a4, AC1, AC2, AC3, AC4, accept, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_PC*	
S151	*a3, a4, AC1, AC2, AC3, AC4, accept, draft, end_NP, final_version, np1, np2, np4, np5, p2, p3, start_flow_author*	
S152	*AC1, AC2, AC3, AC4, end_NP, np1, np2, np4, np5, p2, p3, p4, start_flow_PC*	

S153	*AC1, AC2, AC3, AC4, draft, end_NP, np1, np2, np4, np5, p2, p3, p4, start_flow_author*	
S154	*a3, a4, AC3, AC4, accept, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC, start_NP*	
S155	*a2, a3, a4, AC3, AC4, ack_draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC, start_NP*	
S156	*a1, a2, a3, a4, AC3, AC4, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_flow_PC, start_NP*	
S157	*a1, a2, a3, a4, AC3, AC4, draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S158	*a1, a2, a3, a4, AC1, AC3, AC4, draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S159	*a1, a2, a3, a4, AC1, AC3, AC4, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_flow_PC*	
S160	*a2, a3, a4, AC3, AC4, ack_draft, draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S161	*a2, a3, a4, AC1, AC3, AC4, ack_draft, draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S162	*a2, a3, a4, AC1, AC3, AC4, ack_draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC*	
S163	*a3, a4, AC3, AC4, accept, draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author, start_NP*	
S164	*a3, a4, AC1, AC3, AC4, accept, draft, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_author*	
S165	*a3, a4, AC1, AC3, AC4, accept, end_NP, final_version, np1, np2, np4, np5, p1, p2, p3, start_flow_PC*	
S166	*AC3, AC4, draft, end_NP, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_author, start_NP*	
S167	*AC1, AC3, AC4, draft, end_NP, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_author*	
S168	*AC1, AC3, AC4, end_NP, np1, np2, np4, np5, p1, p2, p3, p4, start_flow_PC*	
S169	*AC1, end_NP, np1, np2, np3, np4, np5, start_flow_PC*	x
S170	*AC1, draft, end_NP, np1, np2, np3, np4, np5, start_flow_author*	x

Analisando a tabela acima, pode-se concluir que dos 170 sifões apresentados, 61 deles possuem uma *Trap* relacionada, portanto não se esvaziam e, consequentemente, não serão analisados. Os outros 109 sifões serão verificados para identificar quais deles, por meio do esvaziamento, caracterizam as situações de *deadlock*.

5.4 Identificação dos Sifões

Para identificar os sifões que estão relacionados com as situações de *deadlock* identificadas por meio do grafo de alcançabilidade, cada um dos 109 sifões apresentados na tabela acima foi analisado com auxílio do software *PIPE*. A análise realizada teve foco em identificar a estrutura e comportamento relacionado ao esvaziamento. Os sifões que não apresentaram nenhuma relação com os *deadlocks* apresentados na Seção 5.3 deste capítulo foram descartados. Os demais sifões foram agrupados de acordo com sua estrutura, que leva o esvaziamento das fichas e caracteriza o *deadlock*. Considerando isto, temos os seguintes grupos:

- os sifões que se esvaziam ao disparar as transições *too_late* e $send\_final\_version$ sequencialmente foram agrupados em $G1 = \{$S68, S69, S70, S73, S74, S75, S78, S79, S80, S88, S89, S91, S92, S9, S95$\}$;
- os sifões que se esvaziam ao disparar as transições *too_late* e *t3* sequencialmente pertencem ao grupo $G2 = \{$S12, S13, S14, S15, S16, S17, S18, S102, S103, S113, S114, S127, S128, S129, S141, S142, S152, S153, S166, S167, S168$\}$;
- os sifões que ficam vazios de fichas ao disparar sequencialmente as transições $receive\_notification\_1$, $receive\_notification\_2$ e *t3* foram agrupados em $G3 = \{$S39, S40, S41, S42, S43, S44, S45, S104, S111, S112, S115, S124, S125, S126, S143, S150, S151, S154, S163, S164, S165$\}$.

Sabendo que todos os sifões relacionados ao *deadlock* devem ser controlados, deve-se aplicar o processo de controle supervisório nos três grupos de sifões, ou seja, (1) nos sifões que se esvaziam ao disparar as transições *too_late* e $send\_final\_version$, (2) nos sifões que se esvaziam ao disparar sequencialmente as transições *too_late* e *t3* e, por fim, (3) nos sifões que se esvaziam quando disparadas as transições $receive\_notification\_1$, $receive\_notification\_2$ e *t3*.

5.5 Controle Supervisórios dos Sifões

Para encontrar o lugar de controle que restringe os disparos das transições que levam à situação de travamento do sistema, será aplicado o procedimento sistemático apresentado na Seção 4.3.1 do capítulo anterior.

Todos os sifões que caracterizam a situação de *deadlock* devem ser controlados e estes sifões, neste trabalho, foram agrupados (*G1*, *G2* e *G3*) tendo em comum o esvaziamento mediante o disparo das transições que levam a tais situações. Dentro de cada agrupamento, o que torna um sifão diferente do outro é a marcação inicial e a combinação de lugares que levam até as transições que necessitam de controle. Sabe-se que a marcação inicial do sifão é essencial para determinar a marcação inicial do lugar de controle. Dentro de cada um dos grupos, há sifões diferentes que podem possuir um único lugar ou dois lugares como marcação inicial. Na arquitetura proposta neste trabalho, cada lugar de controle é transformado em uma *WorkFlow net*, então, para atender a isso, é necessário que estes lugares de controle estejam inicialmente marcados. Logo, serão considerados inicialmente os sifões que possuem dois lugares como marcação inicial.

O primeiro grupo de sifões a ser controlado é o grupo *G1* que possui sifões que se esvaziam com o disparo das transições *too_late* e $send\_final\_version$. O sifão *S89* apresentado na Figura 5.5 é um dos sifões que caracterizam tal situação e é composto pelos lugares: $start\_flow\_author$, *a1*, *a2*, *a3*, *a4*, $final\_author$, end_flow_author, ack_final, $start\_flow\_PC$, *p1*, *p2*, *p3*, *p4* e *p5*. Podemos observar o esvaziamento de fichas pela sequência de disparo: $send\_draft$, $recive\_ack\_draft$, $receive\_accept$, $prepare\_final\_version$, $send\_final\_version$, $receive\_draft$, $send\_ack\_draft$, $evaluate$, *send_accept* e *too_late*. Executando esta sequência de disparo de transições, o sifão *S89* ficará completamente vazio de fichas e nenhuma outra transição poderá ser disparada. Neste sifão há também a possibilidade de esvaziamento de fichas ao disparar sequencialmente as transições $send\_final\_version$ e $send\_reject$. Apesar de $send\_reject$ não ser uma transição que caracteriza o *deadlock* que desejamos controlar, esta transição será considerada no controle supervisório − visto que neste trabalho o foco é controlar todas as transições do sifão que levam ao esvaziamento de fichas.

Portanto, deseja-se que as transições *too_late*, $send\_reject$ e $send\_final\_version$ não sejam disparadas sequencialmente e assim, para estabelecer as restrições de controle baseada nos invariantes de lugar, o sifão *S89* foi modificado. Esta modificação é apresentada na Figura 5.6. A transição $send\_final\_version$ foi transformada para acrescentar um lugar *m1* e uma transição $send\_final\_version'$, a transição $send\_reject$ foi transformada para acrescentar um lugar *m2* e uma transição $send\_reject'$ e a transição *too_late* foi alterada para se acrescentar um lugar m3 e uma transição too_late'. Desta maneira, é possível estabelecer a restrição, na qual não se podem ter fichas em *m1*, *m2* e *m3* simultaneamente, ou seja, $M(m1) + M(m2) + M(m3) \leq 1$.

Devido à estrutura sequencial dos lugares *p3* e *p4*, não é possível ter fichas ao mesmo tempo nos lugares *m2* e *m3* e a presença das fichas nos lugares *m1*, *m2* e *m3* ao mesmo

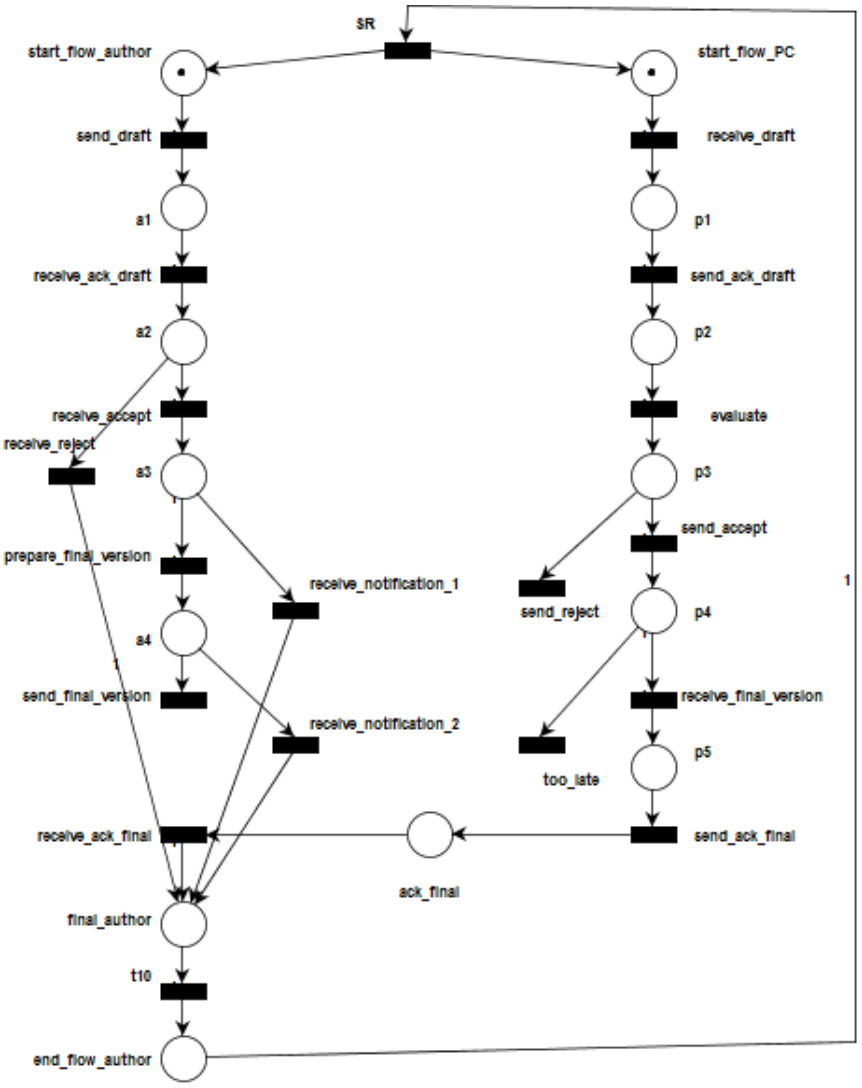

FIGURA 5.5: Sifão S89.

tempo significa o esvaziamento do sifão. Deste modo, a restrição foi estabelecida de modo que os três lugares possam ter no máximo uma ficha ao mesmo tempo, ou seja se houver uma ficha em *m1* não pode haver uma ficha em *m2* ou *m3*. Da mesma maneira, caso haja uma ficha em *m2*, por exemplo, não pode haver uma ficha em *m1*. Por isto, a marcação dos três lugares *m1*, *m2* e *m3* deve ser menor ou igual a 1.

Garantir que esta restrição seja respeitada irá garantir que as transições *too_late*, *send_reject* e $send\_final\_version$ não serão disparadas de modo a levar o esvaziamento total das fichas do sistema. Eventualmente uma ficha pode desaparecer da rede, ou seja, pode estar em *m1*, *m2* ou *m3*, porém a restrição estabelecida acima deve garantir que não haverá mais de um desses lugares com ficha e, por consequência, que outra ficha desapareça do sifão.

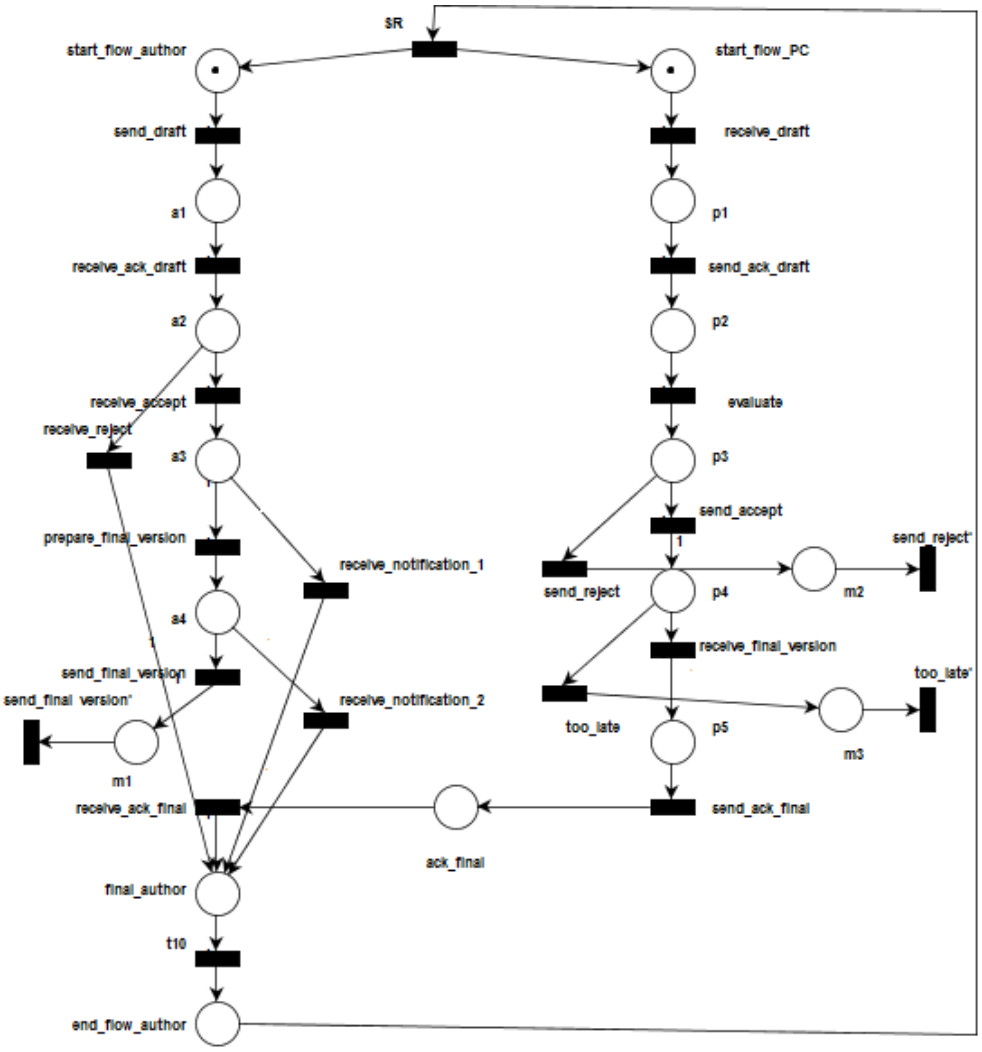

FIGURA 5.6: Sifão S89 modificado.

Dada a especificação de controle $M(m1) + M(m2) + M(m3) \leq 1$ e a sequência de lugares $start\_flow\_author$, $a1$, $a2$, $a3$, $a4$, $final\_author$, end_flow_author, ack_final, $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, $p5$, $m1$, $m2$ e $m3$, temos que a matriz de lugares que estão relacionados à restrição é dada por:

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

A matriz de incidência do sifão *S89* modificado é apresentada a seguir.

$$I_{S89} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Com:

- **Linhas:** *evaluate*, *prepare_final_version*, *receive_accept*, *receive_ack_draft*, *receive_ack_final*, *receive_draft*, *receive_final_version*, *receive_notification_1*, *receive_notification_2*, *receive_reject*, *send_accept*, *send_ack_draft*, *send_ack_ final*, *send_draft*, *send_final_version*, *send_final_version′*, *send_reject*, *send_ reject′*, *SR*, *t*10, *too_late* e *too_late′*.
- **Colunas:** *start_flow_author*, *a*1, *a*2, *a*3, *a*4, *final_author*, *end_flow_author*, *start_flow_PC*, *p*1, *p*2, *p*3, *p*4, *p*5, *ack_final*, *m*1, *m*2 e *m*3.

Tendo a matriz de incidência I_{S89}, o próximo passo para encontrar os arcos do lugar de controle é identificar a matriz I_C, que é calculada por meio da expressão:

$$I_C = -\text{L.C.}$$

Os cálculos para encontrar matriz I_C referente ao sifão *S89* são apresentados na Tabela 5.2.

Tabela 5.2: Transições de entrada e saída do lugar de controle do sifão S89.

ID	Transição	−L.I	Arco
1	$evaluate$	0	–
2	$prepare\_final\_version$	0	–
3	$receive\_accept$	0	–
4	$receive\_ack\_draft$	0	–
5	$receive\_ack\_final$	0	–
6	$receive\_draft$	0	–
7	$receive\_final\_version$	0	–
8	$receive\_notification\_1$	0	–
9	$receive\_notification\_2$	0	–
10	$receive\_reject$	0	–
11	$send\_accept$	0	–
12	$send\_ack\_draft$	0	–
13	$send\_ack\_final$	0	–
14	$send\_draft$	0	–
15	$send\_final\_version$	-1	saída
16	$send\_final\_version'$	1	entrada
17	$send\_reject$	-1	saída
18	$send\_reject'$	1	entrada
19	SR	0	–
20	$t10$	0	–
21	too_late	-1	saída
22	too_late'	1	entrada

Da Tabela 5.2 apresentada acima, temos que:

$$I_C = \Big[\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ -1\ \ 1\ \ -1\ \ 1\ \ 0\ \ 0\ \ -1\ \ 1\ \Big].$$

Ao obter a matriz I_C e sabendo que a marcação inicial do sifão *S89* é $M_0 = (1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)$, com duas fichas, uma em $start\_flow\_author$ e outra em $star\_flow\_PC$, a marcação do lugar de controle é dada por:

$$M_{CP} = M(start\_flow\_author) + M(start\_flow\_PC) - 1 = 1 + 1 - 1 = 1.$$

É possível verificar que o lugar de controle terá como entrada os arcos que sairão das transições $send\_final\_version'$, $send\_reject'$ e too_late' e como saída os arcos que serão entradas das transições $send\_final\_version$, $send\_reject$ e too_late.

Para confirmar que o lugar de controle inserido conseguiu controlar o disparo sequencial das transições *too_late*, *send_reject* e *send_final_version*, foi realizada a execução da rede no *PIPE* e, após a análise do comportamento da rede, foi comprovado que o sifão será controlado, ou seja, não ficará vazio em nenhuma execução a partir da marcação inicial. A Figura 5.7 apresenta a inserção do lugar de controle *CP1* no sifão *S89*.

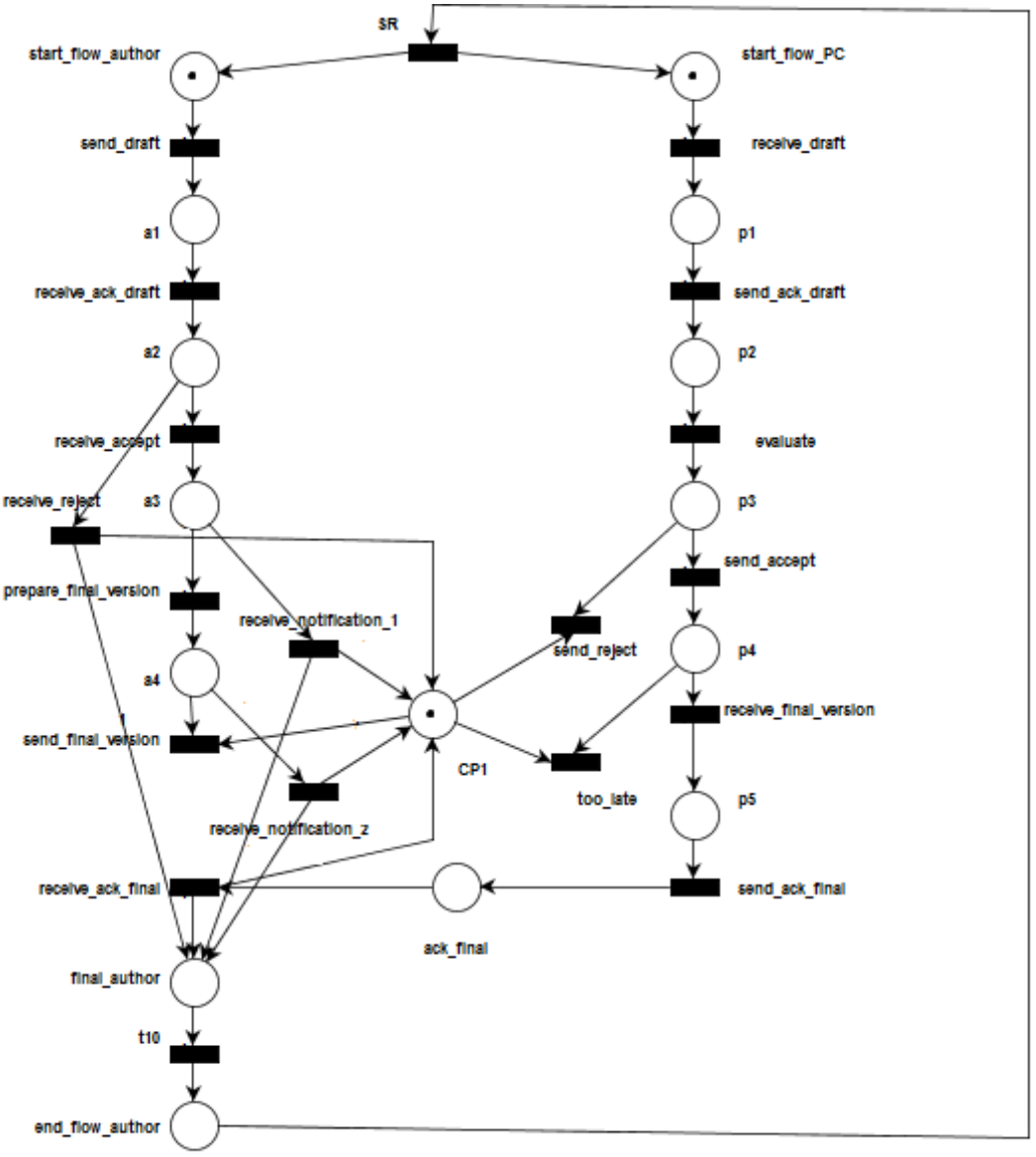

FIGURA 5.7: Sifão S89 com lugar de controle CP1.

Por exemplo, tendo a seguinte sequência de disparo: $send\_draft$, $receive\_ draft$, $send\_ack\_draft$, $receive\_ack\_draft$, $evaluate$, $send\_accept$, $receive\_accept$, $prepare\_final\_version$, após disparar $prepare\_final\_version$ haverá duas transições habilitadas para o disparo: $send\_final\_version$ e too_late. Ao disparar a transição too_late uma ficha será consumida do lugar $p4$ e outra ficha do lugar $CP1$ e, consequentemente,

a ficha irá desparecer da rede. Com isto, a transição *send_final_version* não estará mais habilitada para o disparo, visto que ela possui como lugar de entrada *CP1*, o qual se encontra vazio de fichas. Portanto, a transição que está habilitada para disparo é *receive_notification_2*, que ao efetuar o disparo irá consumir uma ficha do lugar *a4* e gerar uma ficha no lugar *final_author*, impedindo assim o disparo das outras transições que levam o esvaziamento do sifão. O disparo da transição *send_final_version* antes do disparo *too_late* tem um comportamento similar, impedindo que *too_late* seja disparada caso tenha sido consumida a ficha de *CP1* para o disparo de *send_final_version*. O mesmo pode ser dito em relação ao disparo de *send_reject*.

O próximo grupo de sifões a ser controlado é o grupo *G2* que contém os sifões que se esvaziam com o disparo das transições *too_late* e *t3*. Selecionado neste grupo, o sifão *S102* é apresentado na Figura 5.8.

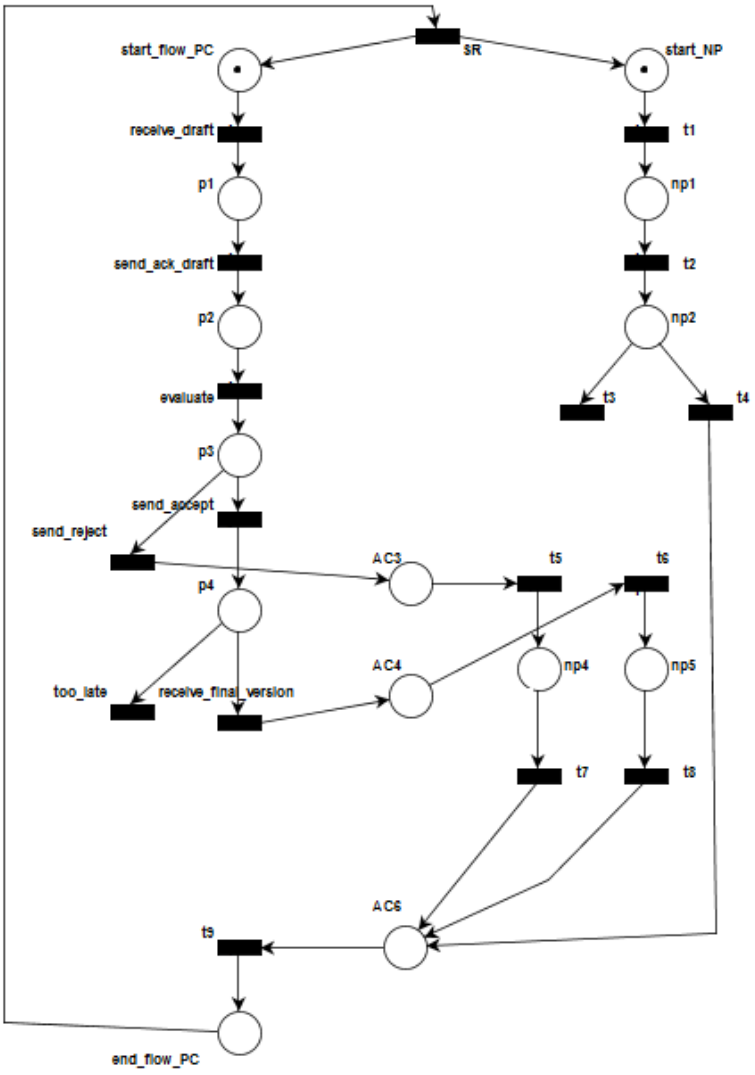

FIGURA 5.8: Sifão S102.

O sifão *S102* é um dos sifões que caracteriza a situação de *deadlock* e é composto pelos lugares: $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, end_flow_PC, $AC3$, $AC4$, $AC6$, $start\_NP$, $np1$, $np2$, $np4$ e $np5$. Podemos observar o esvaziamento de fichas pela sequência de disparo: $receive\_draft$, $send\_ack\_draft$, $evaluate$, $send\_accept$, too_late, ti, $t2$ e $t3$. Após a execução desta sequência de disparo de transições, o sifão *S102* ficará completamente vazio de fichas e, consequentemente, nenhuma outra transição poderá ser disparada.

Para estabelecer as restrições de controle baseadas nos invariantes de lugar e aplicar um controle tal que as transições *too_late* e *t3* não sejam disparadas sequencialmente, o sifão *S102* foi modificado e é apresentado na Figura 5.9.

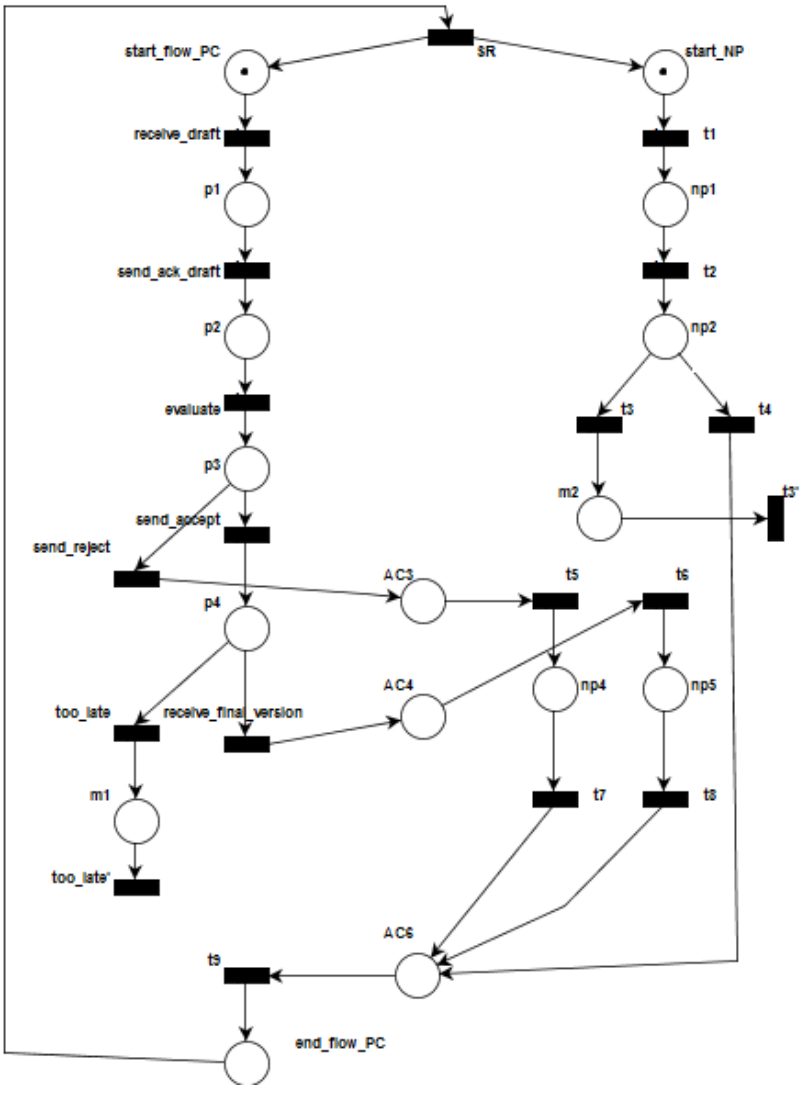

FIGURA 5.9: Sifão S102 modificado.

A transição *too_late* foi transformada para acrescentar um lugar *m1* e uma transição *too_late'* e a transição *t3* foi alterada para se acrescentar um lugar *m2* e uma transição

t3'. Portanto, é possível estabelecer a restrição na qual não se podem ter fichas em *m1* e *m2* simultaneamente, ou seja, $M(m1) + M(m2) \leq 1$.

Garantir que a restrição $M(m1) + M(m2) \leq 1$ seja respeitada irá assegurar que as transições *too_late* e *t3* não serão disparadas em uma mesma execução de modo a levar ao esvaziamento total das fichas do sistema. Eventualmente, uma ficha pode desaparecer da rede, ou seja, pode estar em *m1* ou *m2*, porém a restrição estabelecida acima deve impor que não haverá mais de um desses lugares com ficha. Por exemplo, se houver uma ficha em *m1*, a restrição deve impedir que a transição *t3* dispare, e assim, *m2* não terá fichas. Da mesma maneira se houver alguma ficha em *m2*, a restrição deve impedir que *too_late* fique habilitada ao disparo, não permitindo a existência de uma ficha em *m1*.

Dada a especificação de controle $M(m1) + M(m2) \leq 1$ e os lugares: $start\_\ flow\_PC$, $p1$, $p2$, $p3$, $p4$, end_flow_PC, $AC3$, $AC4$, $AC6$, $start\_NP$, $np1$, $np2$, $np4$, $np5$, $m1$ e $m2$, temos que a matriz de lugares que estão relacionados à restrição é dada por:

$$\mathrm{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

A matriz de incidência do sifão *S102* modificado é apresentada abaixo:

$$I_{S102} = \begin{bmatrix}
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0
\end{bmatrix}$$

Com:

- **Linhas:** $evaluate$, $receive\_draft$, $receive\_final\_version$, $send\_accept$, $send\_ack\_draft$, $send\_reject$, SR, $t1$, $t2$, $t3$, $t3'$, $t4$, $t5$, $t6$, $t7$, $t8$, $t9$, too_late e too_late'.
- **Colunas:** $start\_flow\_PC$, $p1$, $p2$, $p3$, $p4$, end_flow_PC, $AC3$, $AC4$, $AC6$, $start\_NP$, $np1$, $np2$, $np4$, $np5$, $m1$ e $m2$.

Os cálculos para encontrar a matriz I_C referente ao sifão *S102* são apresentados na tabela à seguir:

Tabela 5.3: Transições de entrada e saída do lugar de controle do sifão S102.

ID	**Transição**	**−L.I**	**Arco**
1	$evaluate$	0	–
2	$receive\_draft$	0	–
3	$receive\_final\_version$	0	–
4	$send\_accept$	0	–
5	$send\_ack\_draft$	0	–
6	$send\_reject$	0	–
7	SR	0	–
8	$t1$	0	–
9	$t2$	0	–
10	$t3$	-1	saída
11	$t3'$	1	entrada
12	$t4$	0	–
13	$t5$	0	–
14	$t6$	0	–
15	$t7$	0	–
16	$t8$	0	–
17	$t9$	0	–
18	too_late	-1	saída
19	too_late'	1	entrada

Da tabela 5.3 apresentada acima, temos que:

$$I_C = \left[\begin{array}{cccccccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{array} \right].$$

Obtida a matriz I_C e tendo como marcação inicial do sifão S102 e $M_0 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)$, com duas fichas, uma em $start\_flow\_PC$ e outra em $start\_NP$, a marcação do lugar de controle é dada por:

$$M_{CP} = M(start\_flow\_PC) + M(start\_NP) - 1 = 1 + 1 - 1 = 1.$$

Pode-se verificar que o lugar de controle terá como entrada os arcos que sairão das transições too_late' e $t3'$ e como saída os arcos que serão entradas das transições too_late e $t3$.

A Figura 5.10 apresenta a inserção do lugar de controle *CP2* no sifão *S102*. De maneira similar ao sifão apresentado anteriormente, para confirmar que o lugar de controle inserido conseguiu controlar o disparo sequencial das transições too_late e $t3$ foi realizada a execução no *PIPE* e, após analisar o comportamento da rede, foi observado que o sifão foi controlado da maneira esperada e que o sifão *S102* não ficará vazio em nenhuma execução a partir da marcação inicial.

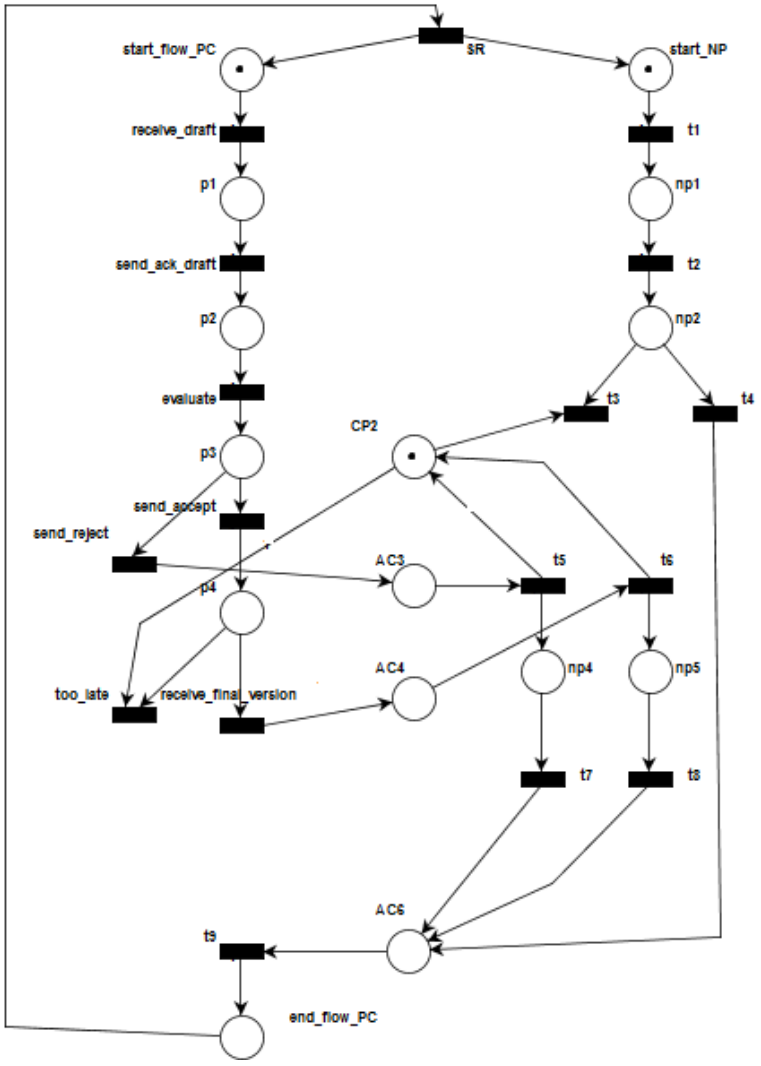

FIGURA 5.10: Sifão S102 com lugar de controle CP2.

O último grupo de sifões a ser controlado é o grupo *G3* que contém sifões que se esvaziam com o disparo das transições $receive\_notification\_1$, $receive\_notification\_2$

e $t3$. O sifão $S42$ apresentado na Figura 5.11 é um dos sifões que fica vazio de fichas por meio do disparo destas transições e é composto pelos lugares: $a3$, $a4$, end_flow_author, $accept$, $final\_version$, $final$, $start\_flow\_PC$, $p1$, $p2$, $p3$, $AC3$, $AC4$, $AC6$, $start\_NP$, $np1$, $np2$, $np4$ e $np5$.

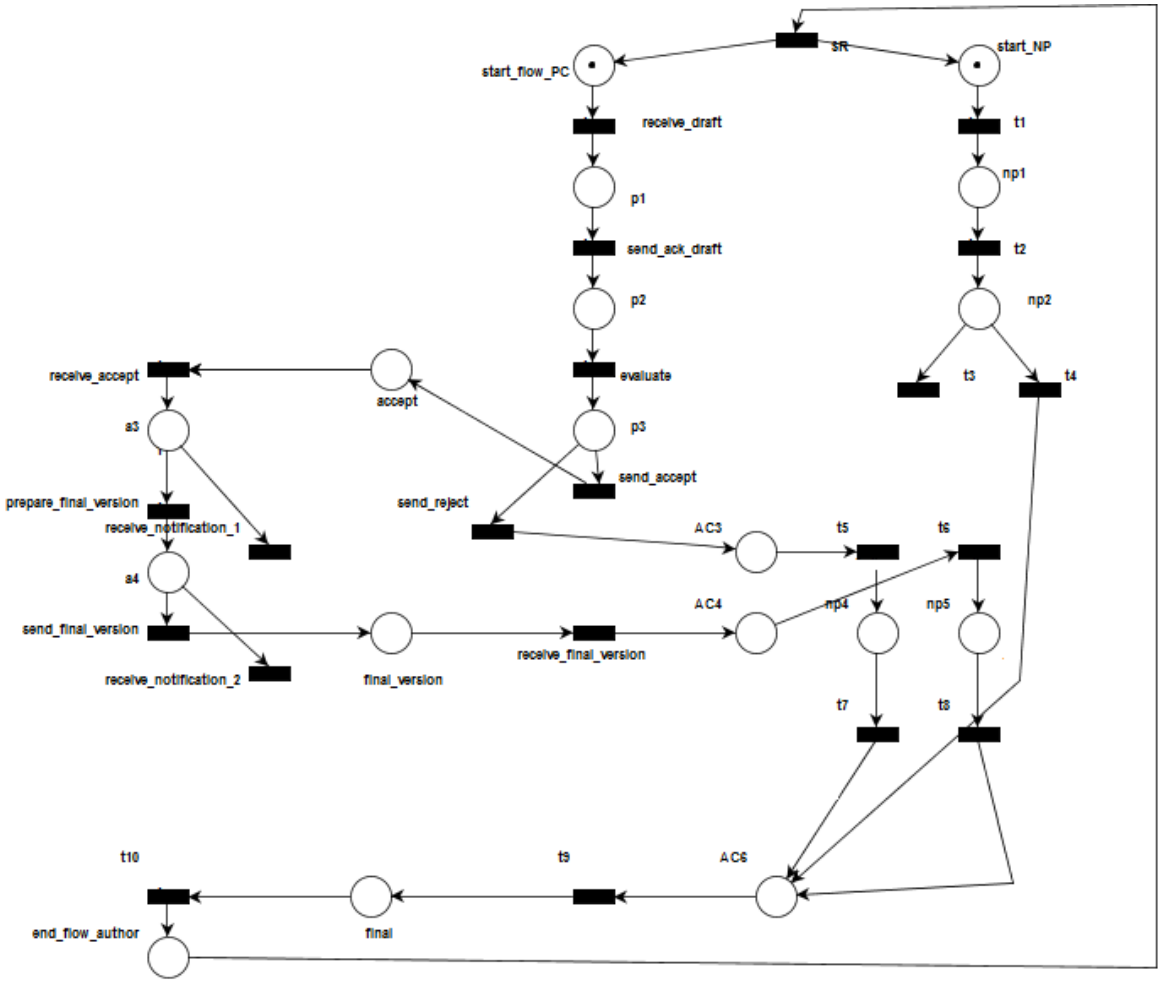

FIGURA 5.11: Sifão S42.

Tendo a seguinte sequência de disparo: $receive\_draft$, $send\_ack\_draft$, $evaluate$, $send\_accept$, $receive\_accept$, $receive\_notification\_2$, ti, $t2$ e $t3$, após o disparo desta sequência de transições, o sifão $S42$ ficará completamente vazio de fichas e, consequentemente, nenhuma outra transição poderá ser disparada.

Para estabelecer as restrições de controle baseadas nos invariantes de lugar e aplicar um controle tal que as transições $receive\_notification\_1$, $receive\_notification\_2$ e $t3$ não sejam disparadas sequencialmente, o sifão $S42$ foi modificado. A transição $receive\_notification\_1$ foi transformada com o acréscimo do lugar m1 e uma transição $receive\_notification\_1'$, a transição $receive\_notification\_2$ foi alterada com o acréscimo do lugar $m2$ e uma transição $receive\_notification\_2'$ e a transição $t3$ foi alterada

com o acréscimo do lugar *m3* e uma transição *t3'*. Portanto, é possível estabelecer a restrição, na qual não se podem ter fichas em *m1*, *m2* e *m3* simultaneamente, ou seja, $M(m1) + M(m2) + M(m3) \leq 1$.

A Figura 5.12 apresenta o sifão *S42* modificado.

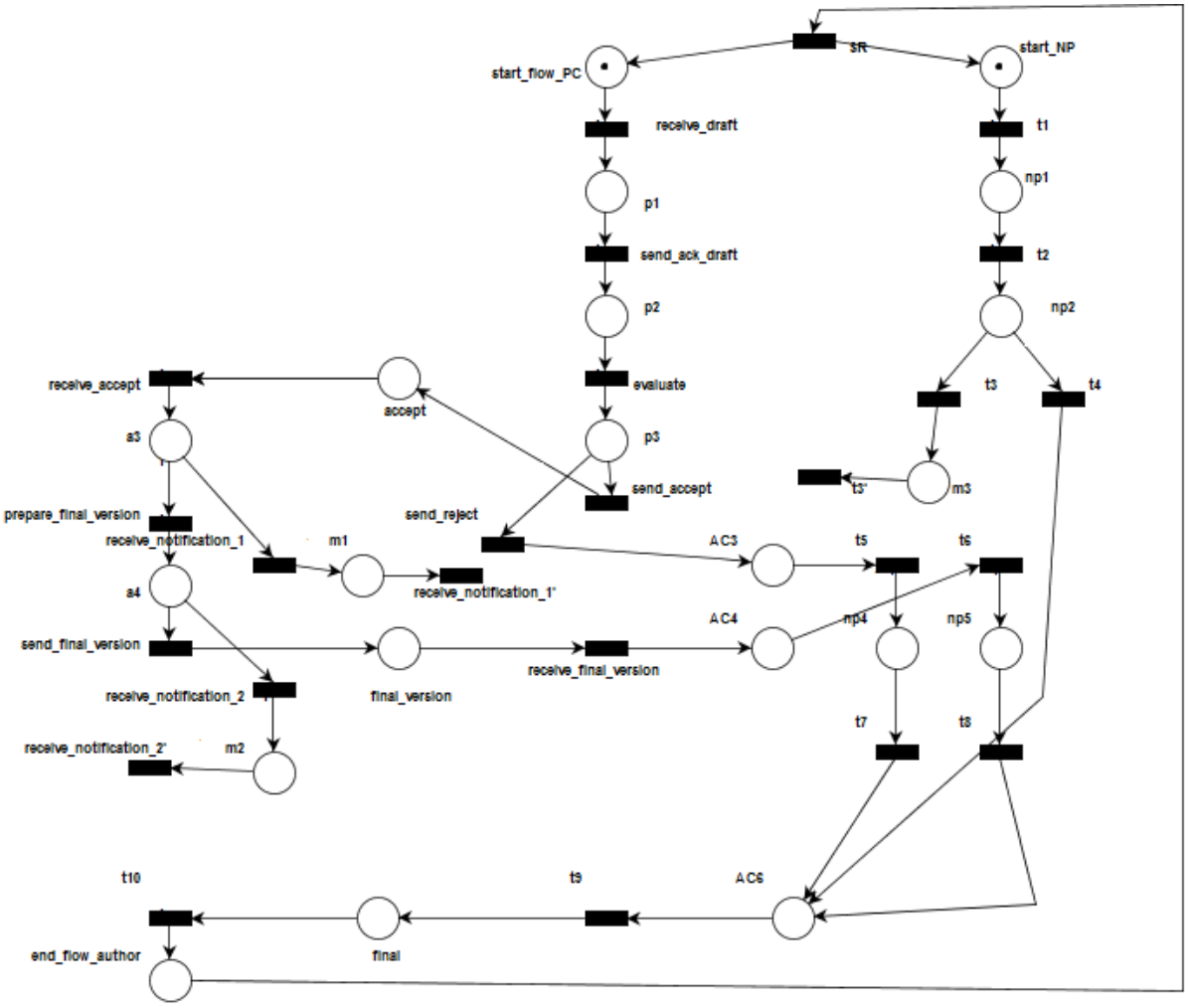

FIGURA 5.12: Sifão S42 modificado.

De forma similar ao sifão *S89*, devido à estrutura sequencial dos lugares *a3* e *a4*, não é possível ter fichas ao mesmo tempo nos lugares *m1* e *m2* e, sabendo que a presença das fichas nos lugares *m1*, *m2* e *m3* ao mesmo tempo significa o esvaziamento do sifão, a restrição foi estabelecida de modo que os três lugares possam ter no máximo 1 ficha ao mesmo tempo, ou seja, caso haja uma ficha em *m3* não pode haver uma ficha em *m1* ou *m2*. Da mesma forma, se houver uma ficha em m1, por exemplo, não pode haver uma ficha em *m3*. Por isto, a marcação dos três lugares *m1*, *m2* e *m3* deve ser menor ou igual a 1.

Dada a especificação de controle $M(m1) + M(m2) + M(m3) \leq 1$ e os lugares: $a3$, $a4$, end_flow_author, $accept$, $final\_version$, $final$, $start\_flow\_PC$, $p1$, $p2$, $p3$, $AC3$, $AC4$, $AC6$, $start\_NP$, $np1$, $np2$, $np4$, $np5$, $m1$ e $m2$, a matriz de lugares que estão relacionados à restrição é dada por:

$$\mathrm{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

A matriz de incidência I_{S42} do sifão *S42* modificado apresentado na Figura 5.12 é apresentada a seguir.

$I_{S42} =$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Com:

- **Linhas:** $a3$, $a4$, end_flow_author, $accept$, $final\_version$, $final$, $start\_flow\_PC$, $p1$, $p2$, $p3$, $AC3$, $AC4$, $AC6$, $start\_NP$, $np1$, $np2$, $np4$, $np5$, $m1$, $m2$ e $m3$.
- **Colunas:** $evaluate$, $prepare\_final\_version$, $receive\_accept$, $receive\_draft$, $receive\_final\_version$, $receive\_notification\_1$, $receive\_notification\_1'$, $receive\_notification\_2$, $receive\_notification\_2'$, $send\_accept$, $send\_ack\_draft$, $send\_final\_\ version$, $send\_reject$, SR, $t1$, $t10$, $t2$, $t3$, $t3'$, $t4$, $t5$, $t6$, $t7$, $t8$ e $t9$.

Tendo a matriz de incidência I_{S42}, o próximo passo para encontrar os arcos do lugar de controle é identificar a matriz I_C, que é calculada por meio da expressão: $I_C = -\text{L.C}$. Os cálculos para encontrar matriz I_C referente ao sifão *S42* são apresentados na Tabela 5.4.

TABELA 5.4: Transições de entrada e saída do lugar de controle do sifão S42.

ID	**Transição**	**−L.I**	**Arco**
1	$evaluate$	0	–
2	$prepare\_final\_version$	0	–
3	$receive\_accept$	0	–
4	$receive\_draft$	0	–
5	$receive\_final\_version$	0	–
6	$receive\_notification\_1$	-1	saída
7	$receive\_notification\_1'$	1	entrada
8	$receive\_notification\_2$	-1	saída
9	$receive\_notification\_2'$	1	entrada
10	$send\_accept$	0	–
11	$send\_ack\_draft'$	0	–
12	$send\_final\_version$	0	–
13	$send_reject$	0	–
14	SR	0	–
15	$t1$	0	–
16	$t2$	0	–
17	$t3$	-1	saída
18	$t3'$	1	entrada
19	$t4$	0	–
20	$t5$	0	–
21	$t6$	0	–
22	$t7$	0	–
23	$t8$	0	–
24	$t9$	0	–
25	$t10$	0	–

Da tabela acima, temos que:

$I_C =$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Após obter a matriz I_C e tendo como marcação inicial do sifão *S42* é $M_0 = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$, com duas fichas, uma em $start\_flow\_PC$ e outra em $star\_NP$ a marcação do lugar de controle é dada por:

$$M_{CP} = M(start\_flow\_PC) + M(start_NP) - 1 = 1 + 1 - 1 = 1.$$

Pode-se verificar que o lugar de controle terá como entrada os arcos que sairão das transições $receive\_notification\_1'$, $receive\_notification\_2'$ e $t3'$ e como saída os arcos que serão entradas das transições $receive\_notification\_1$, $receive\_notification\_2$ e $t3$.

O sifão *S42* com o lugar de controle *CP3* adicionado pode ser visto na figura 5.13. De maneira similar aos sifões apresentados anteriormente, para confirmar que o lugar de controle inserido conseguiu efetivamente assegurar o controle do disparo sequencial das transições $receive\_notification\_1$, $receive\_notification\_2$ e $t3$ foi realizada a execução no *PIPE*, sendo comprovado que o sifão *S42* será controlado e não ficará vazio em nenhuma execução a partir desta marcação inicial.

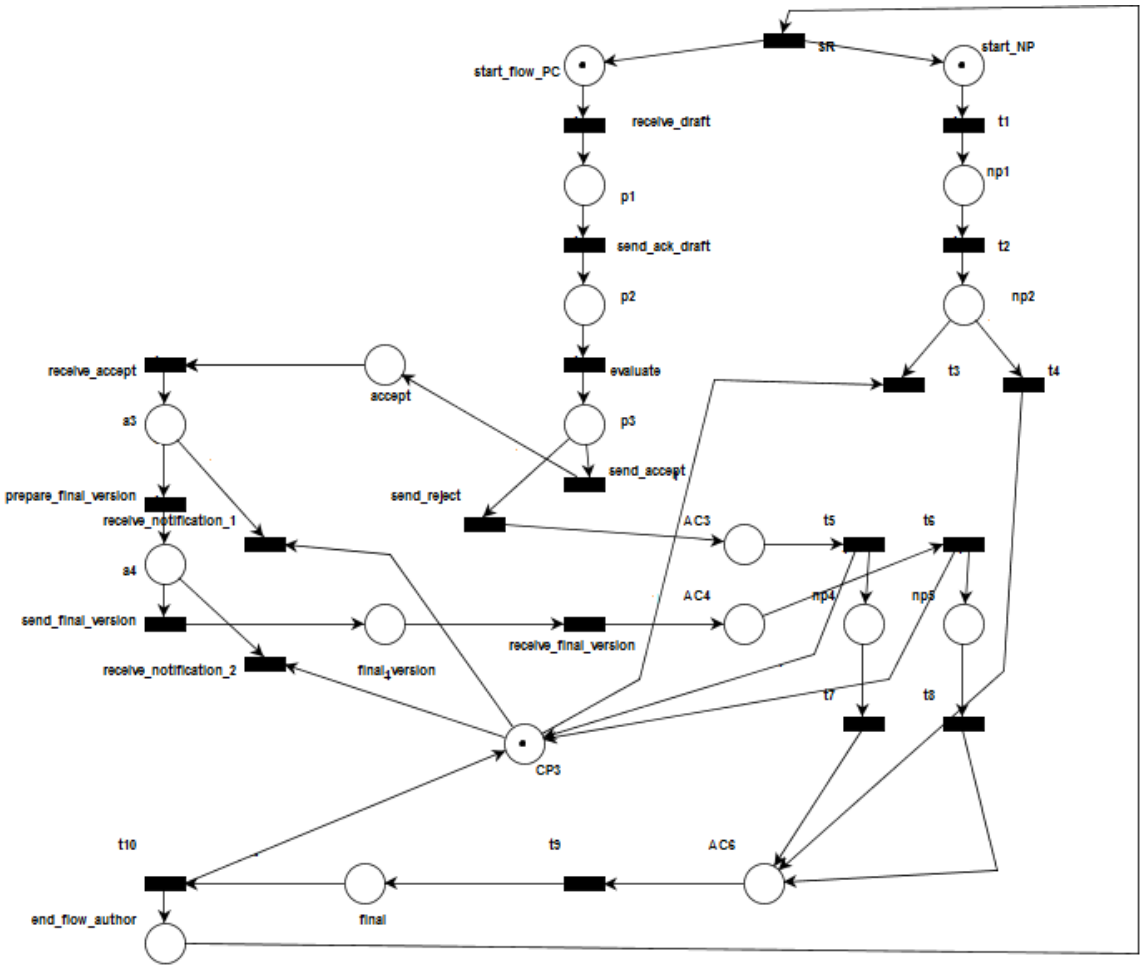

FIGURA 5.13: Sifão S42 com lugar de controle CP3.

Se observarmos a sequência de disparo: $receive\_draft$, *t1*, $send\_ack\_draft$, *t2*, *evaluate*, $send\_accept$ e $receive\_accept$, percebemos que a rede alcança neste momento o estado

com uma ficha no lugar *a3* e outra ficha no lugar *np2*. Desta forma, podemos sensibilizar as transições $prepare\_final\_version$ e $receive\_notification\_1$ que podem disparar consumindo a ficha que está em *a3* e as transições *t3* e *t4* que podem ser disparadas consumindo a ficha que está no lugar *np2*. Se a transição $receive\_notification\_1$ for disparada, uma ficha será consumida do lugar a3 e outra do lugar *CP3* e, como consequência do disparo, a ficha desaparecerá da rede. Contudo, a transição *t3* não poderá ser disparada nesta execução, na qual já aconteceu o disparo de $receive\_notification\_1$, visto que *CP3* é um lugar de entrada dessa transição e está vazio de fichas. O controle das transições também funciona para o disparo sequencial de *t3* e $receive\_notification\_2$. Sendo assim, pode-se concluir que o lugar de controle *CP3* conseguiu efetivamente cumprir o papel de controle de disparo sequencial das transições $receive\_notification\_1$, $receive\_notification\_2$ e *t3* no sifão *S42*.

5.6 Validação do modelo com Controle Supervisório

Após validar, com auxílio do *PIPE*, se a inclusão do lugar de controle em cada sifão está executando o que foi proposto para todos os sifões pertencentes ao grupo, ou seja, se está impedindo os sifões de ficarem completamente vazios de fichas, é necessário incluir os lugares de controle de cada grupo de sifão ao modelo inicial, apresentado na Figura 5.1.

Feita esta inclusão, o próximo passo é verificar se o controle também se aplica à rede completa e se nenhum outro *deadlock* surgiu com a inserção destes lugares. A inserção dos lugares de controle *CP1*, *CP2* e *CP3* na *IOWF-net* é apresentada na Figura 5.14.

Analisando a *IOWF-net* controlada da Figura 5.14, percebemos que o grafo de alcançabilidade que pode ser visualizado na Figura 5.15 não gera nenhum outro *deadlock* e que as situações de *deadlock* identificadas previamente foram completamente controladas. Portanto, podemos afirmar que a adição dos lugares de controle *CP1*, *CP2* e *CP3* identificados por meio do controle dos sifões *S89*, *S102* e *S42*, respectivamente, além de controlar as situações de *deadlock*, não geram nenhuma outra situação. Desta maneira, podemos concluir que a rede está livre de *deadlock*.

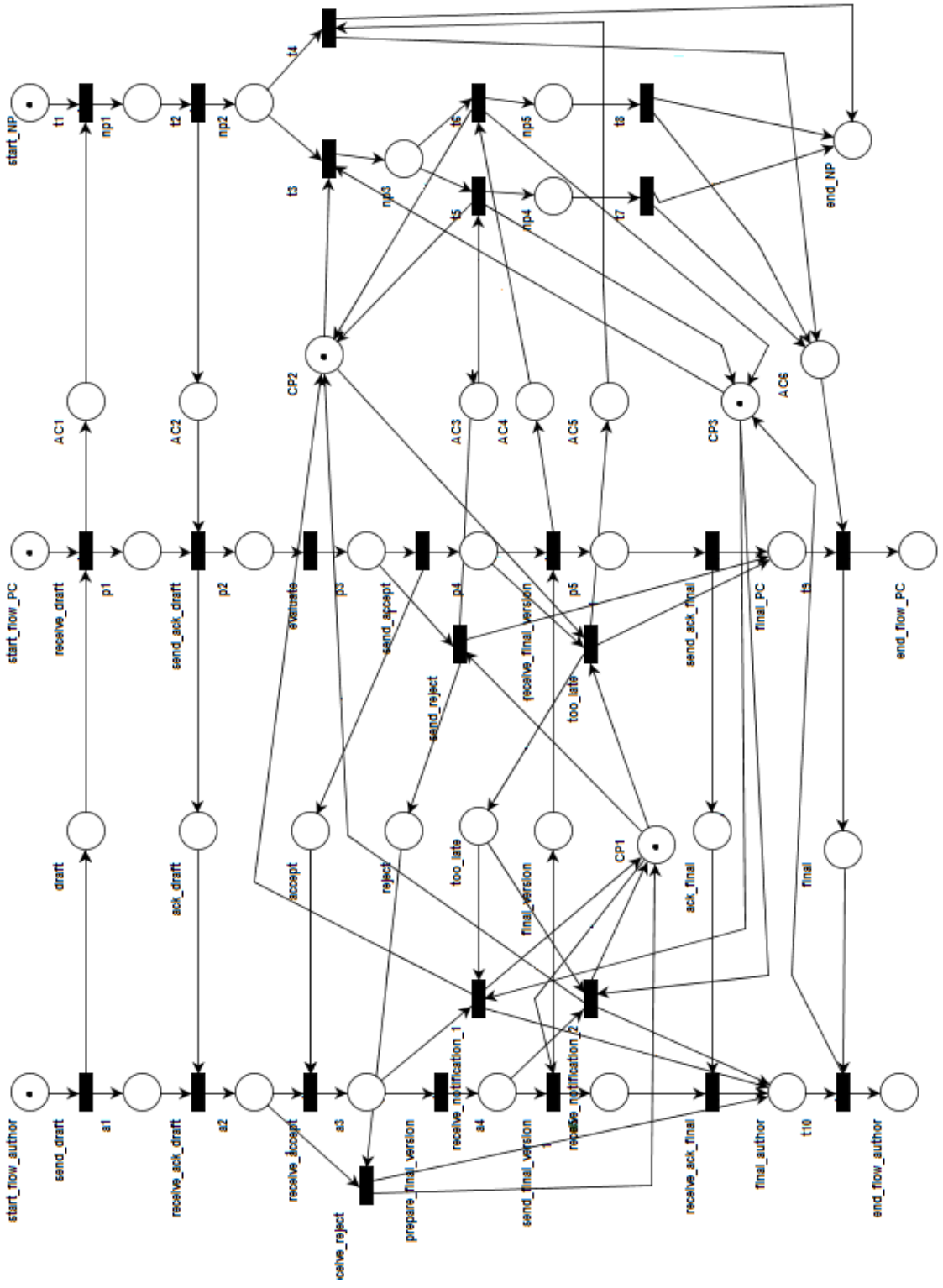

FIGURA 5.14: IOWF-net com lugares de controle CP1, CP2 e CP3.

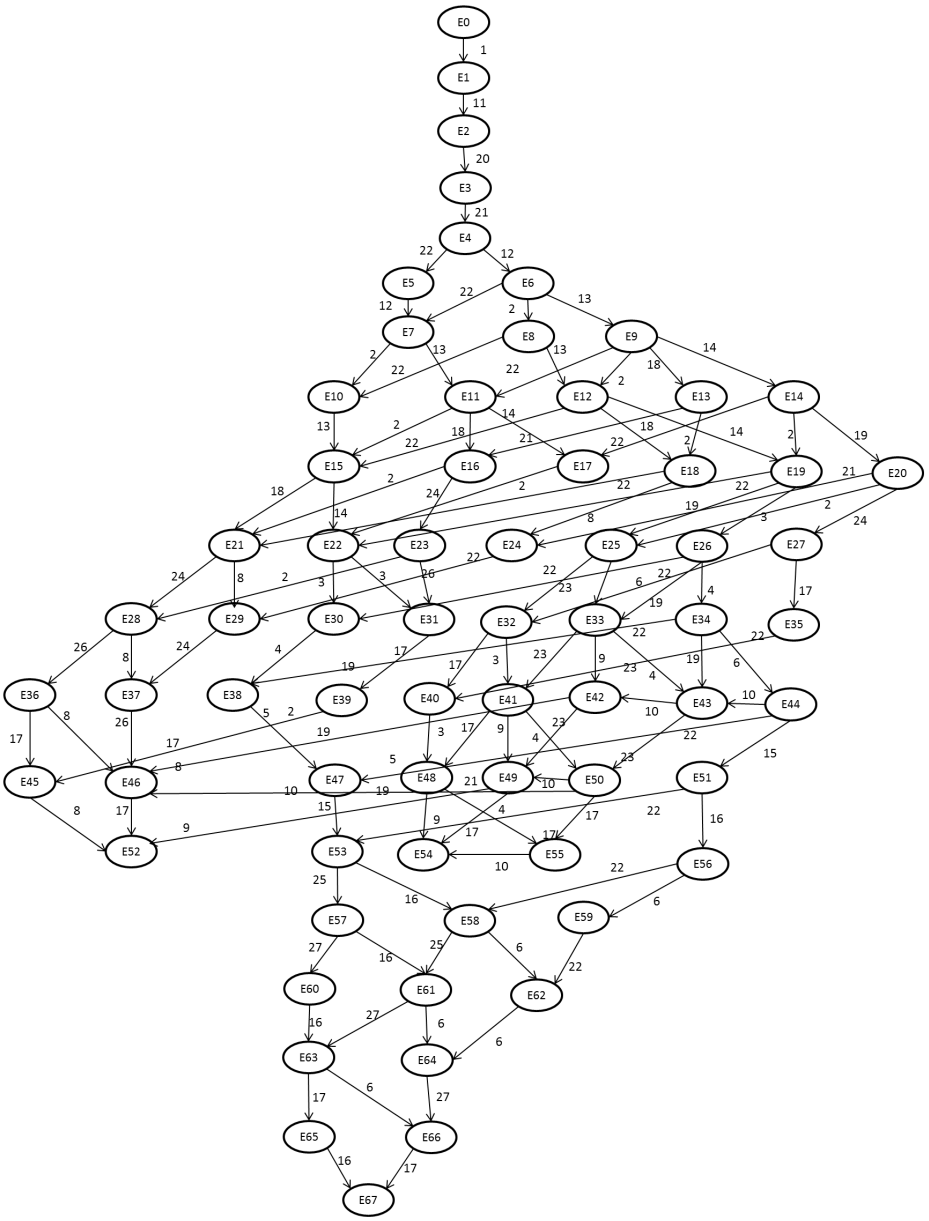

FIGURA 5.15: Grafo de alcançabilidade da Figura 5.14.

Com: 1. *send_draft*, 2. *receive_ack_draft*, 3. *receive_accept*, 4. *prepare_final_version*, 5. *send_final_version*, 6. *receive_ack_final*, 7. $t10$, 8. *receive_reject*, 9. *receive_notification_1*, 10. *receive_notification_2*, 11. *receive_draft*, 12. *send_ack_draft*, 13. *evaluate*, 14. *send_accept*, 15. *receive_final_version*, 16. *send_ack_final*, 17. $t9$, 18. *send_reject*, 19. *too_late*, 20. $t1$, 21. $t2$, 22. $t3$, 23. $t4$, 24. $t5$, 25. $t6$, 26. $t7$, 27. $t8$.

5.7 Modelo com arquitetura distribuída livre de deadlock

Os lugares de controle *CP1*, *CP2* e *CP3* apresentados na Figura 5.14 sempre serão inicialmente marcados, portanto não podem ser vistos apenas como lugares de comunicação assíncrona. Desta maneira, a abordagem proposta para a arquitetura distribuída livre de *deadlock* é de transformar os três lugares de controle em uma única *WorkFlow net* (*CWF*). Esta *WorkFlow net* terá o comportamento de uma *LWF-net* pertencente ao processo global, porém terá sua execução condicionada às solicitações das *LWF-nets* *AU*, *PC* e *NP* (tal processo será visto como um processo passivo).

Sabe-se que a estrutura de controle de *CWF* deverá reproduzir de forma acíclica o comportamento dado pelo invariante de lugar correspondente ao lugar de controle inserido. Portanto, no exemplo deste estudo de caso, deve-se encontrar três invariantes de lugar, cada um correspondente aos lugares de controle *CP1*, *CP2* e *CP3*. Por meio do software *PIPE* foram encontrados os invariantes de lugar apresentados nas Figuras 5.16, 5.17 e 5.18, representando respectivamente os invariantes de *CP1*, *CP2* e *CP3*.

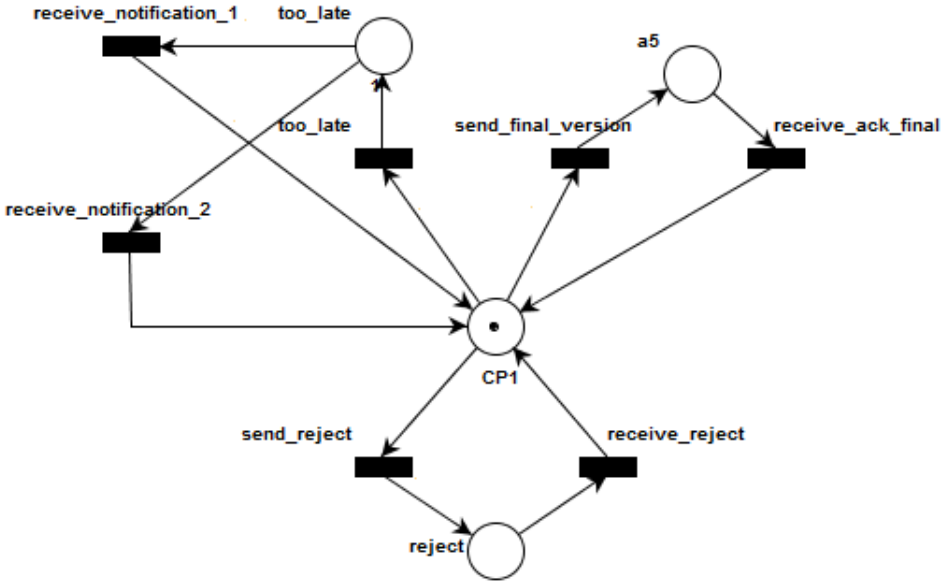

FIGURA 5.16: Invariante de lugar que contém o lugar de controle CP1.

O novo processo, composto pela *WorkFlow net CWF*, possui um lugar de entrada marcado nomeado *start_control*, o qual possui um único arco de saída que será direcionado para a transição *start*, e um lugar de saída nomeado *end_control* com um único arco de entrada, neste caso um arco de saída da transição $final\_control$. As outras transições da nova *WorkFlow net* serão as mesmas que aparecem nos invariantes de lugar das

Figuras 5.16, 5.17 e 5.18: *too_late′*, *receive_notification_1′*, *receive_notification_2′*, *send_final_version′*, *receive_ack_final′*, *send_reject′* e *receive_reject′*, *t3*, *t5'*, *t6'* e *t10'*.

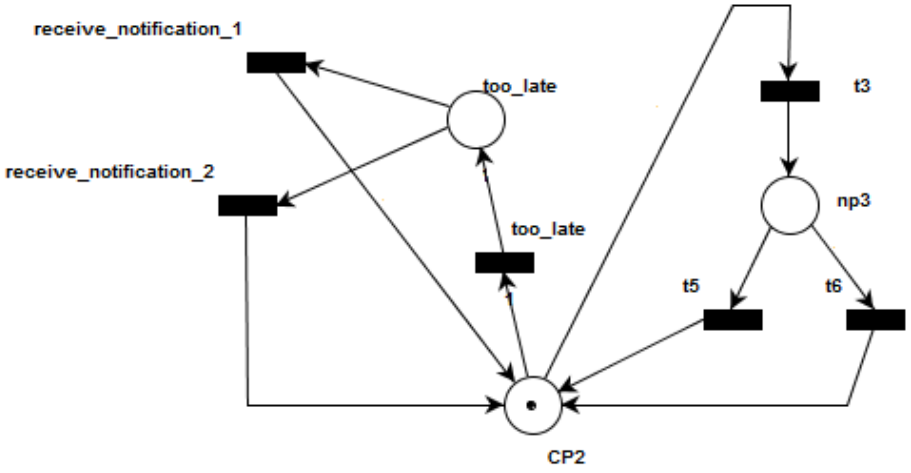

FIGURA 5.17: Invariante de lugar que contém o lugar de controle CP2.

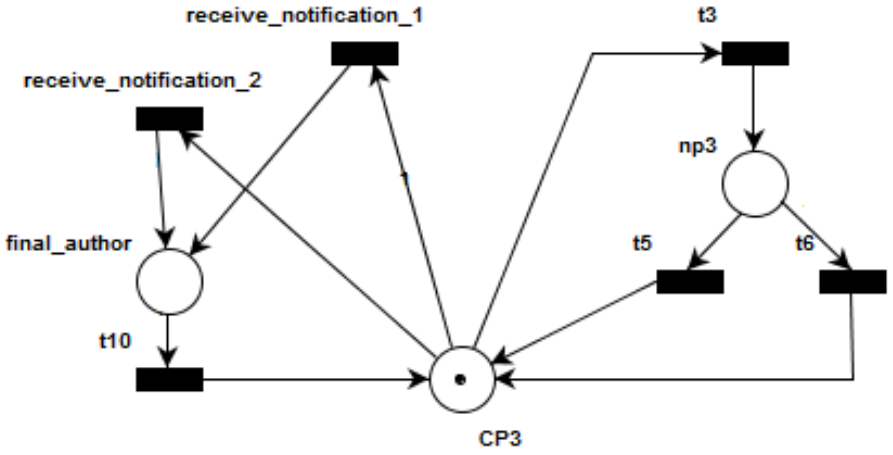

FIGURA 5.18: Invariante de lugar que contém o lugar de controle CP3.

A Figura 5.19 apresenta a arquitetura proposta para a *IOWF-net* da Figura 5.1. Como o controle da execução está associado às *LWF-nets AU*, *PC* e *NP*, o disparo das transições *too_late′*, *send_final_version′*, *send_reject′*, *t3'*, *receive_notification_1′* e *receive_notification_2′* está condicionado às solicitações vindas das *LWF-nets*, as quais requisitam o disparo de suas transições correspondentes.

Por exemplo, temos a seguinte sequência de disparo de transições: *send_draft*, *draft*, *t1*, *t2*, *send_ack_draft*, *receive_ack_draft*, *evaluate*, *send_accept*, *receive_accept* e *prepare_final_version*. Após o disparo de *send_final_version*, o sistema estará marcado com uma ficha nos lugares *a4*, *p4*, *np2* e *start_control*. Neste momento, se o

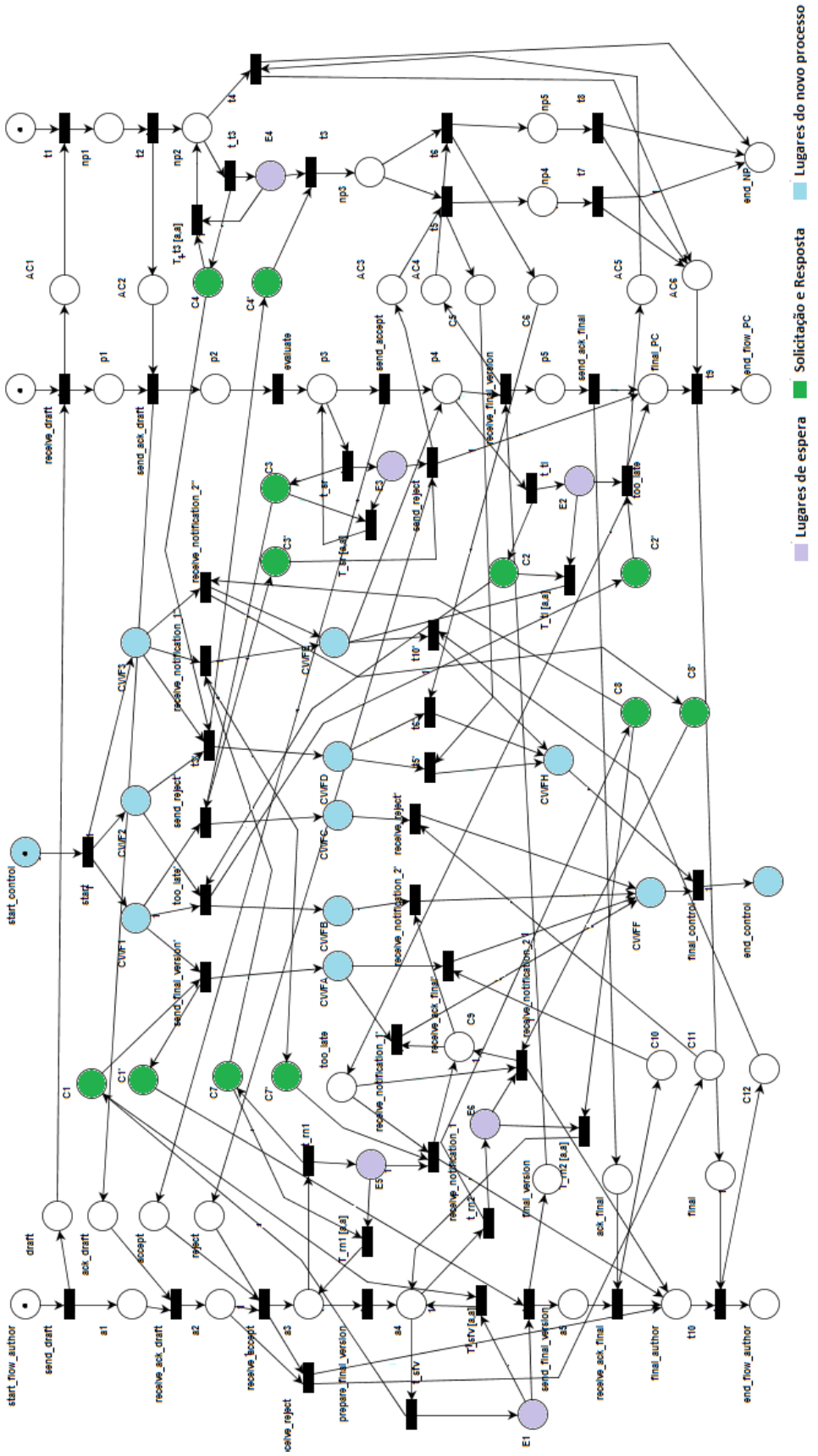

FIGURA 5.19: Proposta de Arquitetura em uma única *WorkFlow net*.

processo *PC* quiser disparar a transição *too_late* haverá o disparo da transição *t_tl* formando uma ficha no lugar de requisição *C2* e outra ficha no lugar de espera *E2*. O processo *CWF* de controle inicia sua execução mediante o disparo da transição *start* e irão surgir fichas nos lugares *CWF1*, *CWF2* e *CWF3*. A transição *too_late'* pertencente à *CWF* será disparada, consumindo uma ficha de *CWF1*, *CWF2* e outra de *C2*; logo uma ficha aparecerá no lugar *C2'* e outra em *CWFB*. Com uma ficha em *C2'* e outra no estado de espera *E2*, a transição *too_late* poderá ser disparada. Seu disparo irá consumir as fichas de *C2'* e *E2*, e formará três fichas: uma no lugar *too_late*, outra em $final\_PC$ e outra em *AC5*.

Agora, supõe-se que *NP* deseja disparar *t3*. A transição *t_t*3 será disparada consumindo uma ficha de *np2* e formando uma ficha *C4* e outra em *E4*. Como não existe uma ficha em *CWF2*, a transição *t3'* não poderá ser disparada. Assim, após um tempo de espera, a transição *T_tl* irá disparar consumindo uma ficha de *C4* e outra de *E4*, devolvendo a ficha para o estado *np2* de onde poderá disparar novamente *t_t*3 ou aguardar para o disparo de t4. Caso haja uma solicitação para o disparo de $send\_final\_version$ após um certo tempo de espera, a ficha será novamente devolvida em *a4*; o processo de controle não poderá disparar a transição $send\_final\_version'$ por não haver mais fichas em *CWF1*. Desta forma, será consumida a ficha de *C1* e outra de *E1* para o disparo de T_sfv que retornará a ficha para *a4*.

A próxima possibilidade é solicitar o disparo da transição $receive\_notification\_2$. Ao disparar a transição t_rn2 uma ficha será consumida do lugar *a4* e duas fichas serão formadas, uma no lugar *C8* e outra em *E6*. A transição $receive\_notification\_2'$ estará sensibilizada e o disparo poderá ser efetuado consumindo uma ficha de *C8* e outra de *CWF3*. Por consequência do disparo, uma ficha se formará no lugar *CWFE* e outra em *C8'*. Com uma ficha em *too_late*, outra em *C8'* e *E6*, a transição $receive\_notification\_2$ será disparada formando uma ficha em $final\_author$ e outra em *C9*. A marcação atual da rede apresenta uma ficha em cada um dos lugares: $final\_author$, *C9, CWFB, CWFE,* $final\_PC$, *AC5* e *np2*. Neste momento a transição *t4* será disparada consumindo as fichas de *np2* e *AC5* para formar fichas em *AC6* e end_NP. Sequencialmente a transição *t9* pertencente a *PC* será disparada, consumindo as fichas de $final\_PC$ e *AC6* para surgir fichas no lugar end_flow_PC e $final$.

Por fim será realizado o disparo da transição *t10*, consumindo as fichas de $final$ e $final\_author$ para formar uma ficha em end_flow_author e outra em *C12*. Neste momento os processos *AU*, *PC* e *NP* finalizarão sua execução contendo cada um deles uma ficha em seus lugares de término, respectivamente, end_flow_author, end_flow_PC e end_NP. Para finalizar a execução da *WorkFlow net* de controle *CWF*, uma ficha será

consumida do lugar *C9* e outra de *CWFB* formando uma ficha em *CWFF*, e a transição *t10'* será disparada consumindo uma ficha de *CWFE* e outra de *C12* e formando uma ficha em *CWFG*. E então, a transição $final\_control$ será disparada consumindo as fichas de *CWFF* e *CWFH* formando uma ficha no lugar *end_control*. Finalizada a execução, não haverá nenhuma outra ficha nos lugares da rede, exceto os lugares de término.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado um procedimento sistemático para detecção e correção de situações de *deadlock* em *Workflow nets* interorganizacionais seguida de uma proposta de abordagem para a arquitetura distribuída livre de *deadlock*.

No procedimento apresentado, inicialmente foi realizada a verificação da propriedade *Soundness* o modelo da *WorkFlow net* interorganizacional a fim de encontrar, por meio da análise desta propriedade algo que inviabilizou o modelo de ser *sound*. Um dos fatores que inviabiliza a propriedade *Soundness* é a presença de *deadlock* no modelo. Identificado que o modelo não é *sound*, o próximo passo foi gerar o grafo de alcançabilidade da rede para verificar a vivacidade da modelo. Percebeu-se por meio do grafo que o modelo não era vivo, foram então identificados quais os estados em que a rede se encontrava em total travamento a partir da marcação inicial e qual a sequência de disparo que levou o sistema a tal estado.

Posteriormente, tendo a informação das situações de *deadlock* existentes na rede, o software *PIPE* (*Plataform Independent Petri net Editor*) foi utilizado para encontrar os sifões e *Traps*. Sabe-se que a existência de situações de *deadlock* nas redes de Petri está relacionada à presença de sifões que se esvaziam [Barkaoui e Abdallah 1995b], [Ezpeleta et al. 1993]. Portanto, foi preciso analisar todos os sifões encontrados pelo *PIPE* para identificar quais deles se esvaziavam e quais eram os sifões que estavam diretamente relacionados com o disparo das transições que geraram o *deadlock*.

Após ter a relação de todos os sifões que podem ficar vazios de fichas e que estão diretamente relacionados à situação de *deadlock*, foi apresentado um procedimento de controle supervisório dos sifões, o qual utiliza, por meio da adição de lugares de controle, um modo de restringir o disparo das transições que levam a rede a alcançar estados indesejáveis. Para encontrar tais lugares, foi estabelecida uma restrição de controle baseada

nos invariantes de lugar, de modo que as transições que levam ao esvaziamento de fichas não possam ser disparadas em uma mesma execução ou de forma sequencial. Assim, se há duas transições que levam ao esvaziamento das fichas do sifão e se uma delas for disparada, a restrição de controle estabelecida deve impedir que a outra transição que também esvazia o sifão seja disparada e dessa forma garantir que o sifão não ficará insuficientemente marcado.

Em seguida, todos os lugares utilizados para controlar o esvaziamento de fichas dos sifões foram inseridos ao modelo inicial. Com a inclusão de novos lugares na rede podem aparecer outras situações de *deadlock*. Uma nova análise da estrutura e das propriedades deve ser feita para garantir que a inclusão destes lugares não gerou novos *deadlock* e que efetivamente restringiu as situações as quais foram propostos para controlar. Se novas situações de *deadlock* surgirem com a inclusão dos lugares de controle, os procedimentos de identificação do sifão e de controle supervisório devem ser feitos novamente até que não existam mais *deadlocks*.

Finalmente, tendo uma *WorkFlow net* interorganizacional livre de *deadlocks* foi feita uma proposta de arquitetura distribuída na qual todos os lugares de controle marcados adicionados à rede são transformados em uma *WorkFlow net* local que se comunicará com os processos locais já existentes. A estrutura da nova rede (com adição dos lugares de controle) reproduz de forma acíclica (a fim de respeitar a estrutura acíclica de *WorkFlow net*), o comportamento do invariante de lugar que contém o lugar de controle. Ou seja, para cada lugar de controle inserido existe um invariante de lugar que contém este lugar. A *WorkFlow net* local proposta na abordagem de arquitetura deste trabalho foi composta por um lugar de início e um lugar final que, respeitando a estrutura das *WorkFlow nets*, possuem apenas um arco de saída e um arco de entrada, respectivamente. As outras transições que compuseram esta *WorkFlow net* foram as mesmas que aparecem nos invariantes de lugar que contêm os lugares de controle.

A principal contribuição deste trabalho foi propor por meio de um procedimento sistemático a detecção e a correção de situações de *deadlock* em *WorkFlow nets* interorganizacionais, sem ter que modificar por inteiro a estrutura da rede original (as *WorkFlow nets* locais), apenas controlando a ocorrência de situações indesejadas.

A hipótese levantada sobre a presença de *deadlock* nas *WorkFlow nets* interorganizacionais estar relacionada à presença de estruturas chamadas sifões, que eventualmente esvaziavam das fichas que continham inicialmente, pode ser considerada como verdadeira. Foi verificado em particular mediante a aplicação de um procedimento sistemático, que as situações de *deadlock* nas redes interorganizacionais estão relacionadas à presença destes sifões e que o controle destas estruturas por meio de restrições impostas sobre o

disparo das transições que o esvaziam impede que as situações de *deadlock* identificadas aconteçam novamente na rede. Desta maneira podemos afirmar que existe e que foi possível estabelecer um procedimento sistemático para detectar as situações de *deadlock* que inviabilizam a boa propriedade *Soundness* nas *WorkFlow nets* interorganizacionais e também que este procedimento permite corrigir as estruturas que produzem os *deadlock* sem modificar as *WorkFlow nets* locais que componham o modelo de especificação das *WorkFlow nets* interorganizacionais.

Sobre este trabalho podem-se destacar os seguintes pontos de contribuição:

- como havia sido proposto inicialmente, foi apresentado que as situações de *deadlock* nas *WorkFlow nets* interorganizacionais estão relacionadas às estruturas chamadas sifões que se esvaziam. A principal vantagem nisto é que esta relação entre sifões e *deadlock* permite que o controle do esvaziamento destas estruturas estabeleça restrições que impeçam a rede de alcançar as situações de *deadlock*;
- foi apresentado também um método de detecção de sifão baseado em software livre de uso como *PIPE* que realiza de modo simples a detecção dos sifões. O *PIPE* também faz análise das propriedades estruturais da rede permitindo a localização, por meio do grafo de alcançabilidade, das situações de travamento do modelo;
- um procedimento de controle supervisório de sifão por meio do método de invariantes de lugar foi utilizado para impedir que o modelo alcançasse os estados de travamento. Este procedimento permite estabelecer restrições de controle que impedem o disparo das transições que levam ao esvaziamento de fichas no sifão e, consequentemente, elimina as situações de *deadlock* no modelo completo;
- foi apresentada uma proposta de arquitetura que possibilita a transformação dos lugares de controle inseridos em uma nova estrutura local apresentada por meio de uma *WorkFlow net* que se comunica de forma assíncrona com as demais *WorkFlow nets* locais já existentes no modelo inicial. Esta arquitetura proposta com a inserção desta nova *WorkFlow net* não modifica o modelo inicial (todos os lugares e transições da rede original se mantêm), apenas acrescentando os novos lugares e transições pertencentes à *WorkFlow net* de controle e os seus devidos lugares de comunicação com as outras *WorkFlow nets* locais.

Além da escrita da dissertação, este trabalho teve como resultado a publicação do artigo "*Siphon-Based Deadlock Prevention Policy for Interorganizational WorkFlow net Design*", na conferência *IEEEIRI'2013 - The 14th IEEE International Conference on Information Reuse and Integration* em São Francisco nos Estados Unidos da América.

Como proposta de trabalhos futuros será interessante analisar e eventualmente corrigir causas que levam ao descumprimento da propriedade *Soundness* para as *WorkFlow nets* interorganizacionais. Em particular, analisar situações quando o modelo não é *sound* mesmo sem a presença de *deadlock*. Geralmente, o modelo pode não ser *sound* devido a produção de fichas adicionais na estrutura de controle do processo que não são consumidas mesmo depois de ter completado o processo por parte de outras fichas. Este cenário pode levar a duplicação da informação e no caso interorganizacional pode, por exemplo, corresponder ao envio repetidamente de um mesmo documento a um mesmo conjunto de processos. Isto seria um problema, pois poderia, por exemplo, gerar a duplicação de uma operação de reembolso a um cliente de uma seguradora.

Referências Bibliográficas

[Aalst 1996] Aalst, W. (1996). Structural Characterizations of Sound Workflow Nets. *Computing Science Reports/23*.

[Aalst 1997] Aalst, W. (1997). Verification of Workflow Nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets (APN'97)*, ICATPN'97, pp. 407–426, London, UK, UK. Springer-Verlag.

[Aalst 1998] Aalst, W. (1998). Modeling and Analyzing Interorganizational Workflows. In *Proceedings of the International Conference on Application of Concurrency to System Design (ACSD'98)*, pp. 262–272, Washington, DC, USA.

[Aalst e Hee 2002] Aalst, W. e Hee, K. (2002). *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 1st edition.

[Aalst et al. 2000] Aalst, W., Hofstede, T., e Arthur, H. M. (2000). Verification of Workflow Task Structures: A Petri-net-based Approach. *Information Systems*, pp. 43–69.

[Abdallah e ElMaraghy 1998] Abdallah, I. B. e ElMaraghy, H. A. (1998). Deadlock prevention and avoidance in FMS: A Petri net based approach. *The International Journal of Advanced Manufacturing Technology*, pp. 704–715.

[Albus et al. 1987] Albus, J., McCain, H. G., e Lumia, R. (1987). NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). Technical report, National Institute of Standard and Technology.

[Awad 2008] Awad, A., P. F. (2008). Structural Detection of Deadlocks in Business Process Models. pp. 239–250.

[Balbo et al. 1994] Balbo, G., Bruell, S. C., e Sereno, M. (1994). Arrival Theorems for Product-form Stochastic Petri Nets. *Proceedings of the ACM SIGMETRICS Performance Evaluation Review*, pp. 87–97.

[Barkaoui e Abdallah 1995a] Barkaoui, K. e Abdallah, I. (1995a). Deadlock avoidance in FMS based on structural theory of Petri nets. In *Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation (ETFA'95)*, pp. 499–510.

[Barkaoui e Abdallah 1995b] Barkaoui, K. e Abdallah, I. (1995b). A deadlock prevention method for a class of FMS. In *Proceedings of the Systems, Man and Cybernetics. Intelligent Systems for the 21st Century. SMC'95. IEEE International Conference on*, pp. 4119–4124 vol.5.

[Barkaoui et al. 2008] Barkaoui, K., Ben Ayed, R., Boucheneb, H., e Hicheur, A. (2008). Verification of Workflow processes under multilevel security considerations. In *Proceedings of the 30th International Conference on Risks and Security of Internet and Systems (CRiSIS'08)*, pp. 77–84.

[Bass et al. 2012] Bass, L., Clements, P., e Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley, 3rd edition.

[Boer e Murata 1994] Boer, E. R. e Murata, T. (1994). Generating basis siphons and traps of Petri nets using the sign incidence matrix. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, pp. 266–271.

[Brown 2000] Brown, A. W. (2000). *Large-scale, component-based development*. Prentice Hall PTR, 1st edition.

[Cai et al. 2000] Cai, X., Lyu, M. R., Wong, K., e Ko, R. (2000). Component-based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes. In *Proceedings of the Seventh Asia-Pacific Software Engineering Conferenec (APSEC'00)*, Washington, DC, USA. IEEE Computer Society.

[Cardoso e Valette 1997] Cardoso, J. e Valette, R. (1997). *Redes de Petri*. Editora da Universidade Federal de Santa Catarina - UFSC.

[Chu e Xie 1997] Chu, F. e Xie, X.-L. (1997). Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation*, pp. 793–804.

[Coffman et al. 1971] Coffman, E. G., Elphick, M., e Shoshani, A. (1971). System Deadlocks. *ACM Computing Surveys*, pp. 67–78.

[Creswell 2011] Creswell, J. W. (2011). *Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research*. Pearson, Upper Saddle River, NJ, EUA, 4th edition.

[Desel e Erwin 2000] Desel, J. e Erwin, T. (2000). Modeling, Simulation and Analysis of Business Processes. pp. 129–141.

[Ezpeleta et al. 1995] Ezpeleta, J., Colom, J. M., e Martinez, J. (1995). A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, pp. 173–184.

[Ezpeleta et al. 1993] Ezpeleta, J., Couvreur, J., e Silva, M. (1993). A new technique for finding a generating family of siphons, traps and st-components. Application to colored Petri nets. In Rozenberg, G. (editor), *Advances in Petri Nets 1993*, Lecture Notes in Computer Science, pp. 126–147. Springer Berlin Heidelberg.

[Fanti et al. 2000] Fanti, M. P., Maione, B., e Turchiano, B. (2000). Comparing digraph and Petri net approaches to deadlock avoidance in FMS. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics (SMC'00)*, pp. 783–798.

[Fanti e Turchiano 2002] Fanti, M. P. e Turchiano, B. (2002). Deadlock analysis in automated manufacturing systems with conjunctive resource service. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'02)*, pp. 181–186.

[Fantinato et al. 2005] Fantinato, M., Toledo, M., e Gimenes, I. (2005). Arquitetura de Sistemas de Gerenciamento de Processos de Negócio Baseado em Serviços. Technical report, UNICAMP.

[Florin e Natkin 1985] Florin, G. e Natkin, S. (1985). *Les réseaux de Petri stochastiques: rapport de recherche informatique*. CNAM, Paris, França.

[Gang e Ming 2004] Gang, X. e Ming, Z. (2004). Systemic solutions to deadlock in FMS. In *Proceedings of the American Control Conference (ACC'04)*, pp. 5740–5745.

[Gang e Zhiming 2003] Gang, X. e Zhiming, W. (2003). A new method for FMS modeling and formal verification. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03)*, pp. 224–231.

[Garlan 2000] Garlan, D. (2000). Software Architecture: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*, pp. 91–101, New York, NY, USA. ACM.

[Genrich 1987] Genrich, H. J. (1987). Predicate/Transition Nets. In *Proceedings of the Advances in Petri Nets, Part I on Petri Nets: Central Models and Their Properties (APN'87)*, pp. 207–247, London, UK, UK.

[Gil 2002] Gil, A. C. (2002). *Como elaborar projetos de pesquisa*. Atlas S.A., São Paulo, SP, Brasil, 4th edition.

[Girault e Valk 2001] Girault, C. e Valk, R. (2001). *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Giua e DiCesare 1992] Giua, A. e DiCesare, F. (1992). On the existence of Petri net supervisors. In *Proceedings of the 31st IEEE Conference on Decision and Control (CDC'92)*, pp. 3380–3385.

[Hack 1972] Hack, M. H. T. (1972). Analysis of production schemata by Petri nets. Master's thesis, MIT.

[Hayes-Roth 1985] Hayes-Roth, B. (1985). A Blackboard Architecture for Control. *Artificial Intelligence*, pp. 251–321.

[Hofstede et al. 2009] Hofstede, A. H. M., Aalst, W., Adams, M., e Russell, N. (2009). *Modern Business Process Automation: YAWL and its Support Environment*. Springer.

[Hollingsworth 1994] Hollingsworth, D. (1994). Workflow Management Coalition: The Workflow Reference Model. Issue 1.1. Workflow Management Coalition.

[Holloway e Krogh 1994] Holloway, L. E. e Krogh, B. H. (1994). Controlled Petri nets: A tutorial survey. In *Proceedings of the 11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*, pp. 158–168. Springer Berlin Heidelberg.

[Holt 1972] Holt, R. C. (1972). Some Deadlock Properties of Computer Systems. *ACM Computing Surveys*, pp. 179–196.

[Ichikawa e Hiraishi 1988] Ichikawa, A. e Hiraishi, K. (1988). Analysis and Control of Discrete Event Systems Represented by Petri Nets. In *Proceedings of the Conference Sopron on Discrete Event Systems: Models and Applications (IIASA'88)*, pp. 115–134. Springer Berlin Heidelberg.

[Iordache et al. 2002] Iordache, M., Moody, J., e Antsaklis, P. (2002). Synthesis of deadlock prevention supervisors using Petri nets. *IEEE Transactions on Robotics and Automation*, pp. 59–68.

[Iordache 2003] Iordache, M. V. (2003). *Methods for the supervisory control of concurrent systems based on Petri net abstractions*. PhD thesis, Universidade de Notre Dame, South Bend, IN, EUA.

[Iordache e Antsaklis 2002] Iordache, M. V. e Antsaklis, P. J. (2002). Synthesis of supervisors enforcing general linear vector constraints in Petri nets. In *Proceedings of the American Control Conference (ACC'02)*, pp. 154–159.

[Iordache e Antsaklis 2006] Iordache, M. V. e Antsaklis, P. J. (2006). Supervision Based on Place Invariants: A Survey. *Discrete Event Dynamic Systems*, pp. 451–492.

[Jacobson 1997] Jacobson, I. (1997). *Software Reuse*. Pearson Education, 1st edition.

[Jeng e Peng 1996] Jeng, M. e Peng, M. (1996). Generating minimal siphons and traps for Petri nets. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'96)*, pp. 2996–2999.

[Jensen 1991] Jensen, K. (1991). Coloured petri nets: A high level language for system design and analysis. In Rozenberg, G. (editor), *Proceedings of the Advances in Petri Nets (APN'91)*, Lecture Notes in Computer Science, pp. 342–416. Springer Berlin Heidelberg.

[Jensen 1994] Jensen, K. (1994). Book Review: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. *Proceedings of the SIGOPS Operation Systems Review*, pp. 1–2. Reviewer-Nutt, Gary J.

[Julia et al. 2008] Julia, S., de Oliveira, F. F., e Valette, R. (2008). Real time scheduling of Workflow Management Systems based on a p-time Petri net model with hybrid resources. *Simulation Modelling Practice and Theory*, (4):462–482.

[Julia e Soares 2003] Julia, S. e Soares, M. S. (2003). Verification of Real Time UML specifications through a specialized inference mechanism based on a Token Player Algorithm and the sequent calculus of Linear Logic. In *Proceedings of the 15th European Simulation Symposium and Exhibition (EMSS'03)*, pp. 65–70.

[Kinuyama e Murata 1986] Kinuyama, M. e Murata, T. (1986). Generating siphons and traps by Petri Nets representation of logic equations. In *Proceedings of the 2nd Conference of the Net Theory (SIG-IECE'86)*, pp. 93–100.

[Lautenbach e Ridder 1996] Lautenbach, K. e Ridder, H. (1996). The Linear Algebra of Deadlock Avoidance: A Petri Net Approach. Technical report, Institute of Software Technology, University of Koblenz-Landau, Alemanha.

[Lawley e Ferreira 1994] Lawley, M. e Ferreira, P. (1994). An Automaton Based Framework For Analysis And Control Of Flexible Manufacturing Systems. In *Kansas State University Institute*, pp. 144–151.

[Lawley et al. 1998] Lawley, M., Ferreira, P., e Reveliotis, S. (1998). The application and evaluation of banker's algorithm for deadlock-free buffer space allocation in flexible manufacturing systems. *International Journal of Flexible Manufacturing Systems*, pp. 73–100.

[Leymann e Roller 1997] Leymann, F. e Roller, D. (1997). Workflow-based Applications. *IBM Systems Journal*, pp. 102–123.

[Li e Zhou 2003] Li, Z. W. e Zhou, M. (2003). A deadlock control method using elementary siphons of Petri nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'03)*, pp. 2716–2720.

[Li e Zhou 2004] Li, Z. W. e Zhou, M. (2004). 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 (SMC'04)*, pp. 38–51.

[Li e Zhou 2008] Li, Z. W. e Zhou, M. (2008). On Siphon Computation for Deadlock Control in a Class of Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans (SMC'08)*, pp. 667–679.

[López-Grao e Colom 2006] López-Grao, J. P. e Colom, J. M. (2006). Resource Allocation Systems: Some Complexity Results on the S4PR Class. In *Proceedings of the Formal Techniques for Networked and Distributed Systems (FORTE'06)*, pp. 323–338. Springer Berlin Heidelberg.

[Maranhão e Macieira 2004] Maranhão, M. e Macieira, M. E. B. (2004). *O processo nosso de cada dia: Modelagem de processos de trabalho*. Qualitymark, Rio de Janeiro, RJ, Brasil, 2nd edition.

[Marconi e Lakatos 2003] Marconi, M. A. e Lakatos, E. A. (2003). *Fundamentos de metodologia científica*. Atlas S.A., São Paulo, SP, Brasil, 5th edition.

[Maruta et al. 1998] Maruta, T., Onoda, S., Ikkai, Y., Kobayashi, T., e Komoda, N. (1998). A deadlock detection algorithm for business processes workflow models. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*, pp. 611–616.

[Medeiros 1997] Medeiros, A. A. D. (1997). *Contrôle d'exécution pour robots mobiles autonomes: architecture, specification et validation*. PhD thesis, Universidade Paul Sabatier, Toulouse, França.

[Merlin 1974] Merlin, P. M. (1974). *A Study of the Recoverability of Computing Systems*. PhD thesis, Irvine, CA, EUA.

[Merz et al. 1995] Merz, M., Moldt, D., Muller, K., e Lamersdorf, W. (1995). *Workflow Modeling and Execution with Coloured Petri Nets in COSM*.

[Minayo 1996] Minayo, M. C. S. (1996). *Pesquisa social : teoria, método e criatividade*. Vozes, Petrópolis, RJ, Brasil, 6th edition.

[Molloy 1981] Molloy, M. K. (1981). *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, University of California, LA, EUA.

[Moody e Antsaklis 1998] Moody, J. e Antsaklis, P. (1998). *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, Norwell, MA, USA.

[Moody et al. 1999] Moody, J., Iordache, M., e Antsaklis, P. J. (1999). Enforcement of event-based supervisory constraints using state-based methods. In *Proceedings of the 38th IEEE Conference on Decision and Control (CDC'99)*, pp. 1743–1748.

[Moody et al. 1994] Moody, J., Yamalidou, K., Lemmon, M., e Antsaklis, P. (1994). Feedback control of Petri nets based on place invariants. In *Proceedings of the 32th Conference on Decision and Control*, University of Notre Dame, USA.

[Moody e Antsaklis 1996] Moody, J. O. e Antsaklis, P. J. (1996). Supervisory control of Petri nets with uncontrollable/unobservable transitions. In *Proceedings of the 35th IEEE Conference on Decision and Control (CDC'96)*, pp. 4433–4438.

[Murata 1989] Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580.

[Norris 1998] Norris, J. R. (1998). *Markov Chains*. Cambridge University Press.

[Oberweis et al. 1997] Oberweis, A., Schatzle, R., Stucky, W., Weitz, W., e Zimmermann, G. (1997). INCOME/WF - A Petri Net Based Approach to Workflow Management. In *Proceedings of the 3rd Conference on Wirtschaftsinformatik (WI'97)*, pp. 557–580. Springer.

[Pádua et al. 2004] Pádua, S. I. D., Silva, A. R. Y., Inamasu, R. Y., e Porto, A. J. V. (2004). O potencial das redes de Petri em modelagem e análise de processos de negócios. *Revista Gestão e Produção*, pp. 1–11.

[Peterson 1981] Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[Petri 1962] Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Institut fur Instrumentelle Mathematik, Bonn.

[Pfleeger e Atlee 2009] Pfleeger, S. L. e Atlee, J. M. (2009). *Software Engineering: Theory and Practice*. Pearson Prentice Hall, 4th edition.

[PIPE 2003] PIPE (2003). PIPE - Plataform Independent Petri net Editor. http://pipe2.sourceforge.net/. Accessed: 2012-06-15.

[PnetLab 2006] PnetLab (2006). PnetLab 4.0. http://www.automatica.unisa.it/PnetLab.html. Accessed: 2012-05-11.

[Ramadge e Wonham 1987] Ramadge, P. J. e Wonham, W. M. (1987). Supervisory Control of a Class of Discrete Event Processes. *SIAM J. Control Optim.*, pp. 206–230.

[Ramchandani 1974] Ramchandani, C. (1974). Analysis of asynchronous concurrent systems by timed Petri nets. Technical report, Cambridge, MA, USA.

[Reisig 1985] Reisig, W. (1985). *Petri Nets: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA.

[Rillo 1988] Rillo, M. (1988). *Aplicacões de Redes de Petri em sistemas de Manufatura*. PhD thesis, Escola Politécnica da USP, São Paulo, SP, Brasil.

[Sanders 1998] Sanders, M. (1998). Constraint programming with object-oriented Petri nets. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*.

[Santos Filho 1998] Santos Filho, D. J. (1998). *Controle de Sistemas Antropocêntricos de Produção Baseado em Redes de Petri Interpretadas*. PhD thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil.

[Santos Filho 2000] Santos Filho, D. J. (2000). Aspectos do Projeto de Sistemas Produtivos. Technical report, Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil.

[Shim et al. 2002] Shim, J., Han, D., e Kim, H. (2002). Communication Deadlock Detection of Inter-organizational Workflow Definition. In *Proceedings of the Second International Workshop on Databases in Networked Information Systems (DNIS'02)*, pp. 43–57.

[Sibertin-Blanc 1985] Sibertin-Blanc, C. (1985). High Level Petri nets with Data Structure. In *Proceedings of the 6th European Workshop on Application and Theory of Petri Nets (APN'85)*, pp. 141–170, Espoo, Finlândia. Jensen, K.

[Sifakis 1977] Sifakis, J. (1977). Use of Petri Nets for Performance Evaluation. In *Proceedings of the Third International Symposium on Measuring, Modelling and Evaluating Computer Systems (MMB'77)*, pp. 75–93, Amsterdam, NL, Holanda.

[Silbertschatz et al. 2004] Silbertschatz, A., Galvin, P., e Gagne, G. (2004). *Sistemas Operacionais: Conceitos e Aplicações*. Campus, Rio de Janeiro, RJ, Brasil, 1st edition.

[Silva e Menezes 2005] Silva, E. L. e Menezes, E. M. (2005). *Metodologia da Pesquisa e elaboração de dissertação*. Atual, UFSC, Florianópolis, SC, Brasil, 4th edition.

[Simmons 1994] Simmons, R. G. (1994). Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, pp. 34–43.

[Soares Passos e Julia 2009] Soares Passos, L. M. e Julia, S. (2009). Qualitative analysis of WorkFlow nets using linear logic: Soundness verification. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'09)*, pp. 2843–2847.

[Sommerville 2010] Sommerville, I. (2010). *Software Engineering*. Pearson Education, Boston, USA, 9th edition.

[Symons 1978] Symons, F. J. W. (1978). *Modelling and Analysis of Communication Protocols Using Numerical Petri Nets*. PhD thesis, University of Essex, Irvine, Colchester, Essex, UK.

[Szyperski 2002] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman, Boston, MA, USA, 2nd edition.

[Tanenbaum 2010] Tanenbaum, A. S. (2010). *Sistemas Operacionais Modernos*. LTC, 3rd edition.

[Tigli et al. 1993] Tigli, J. Y., Occello, M., e Thomas, M. C. (1993). Toward a new intelligent reactive controller for autonomous mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'93)*, pp. 249–254.

[Tomiyama 2007] Tomiyama, M. N. (2007). Simulação e Modelagem de Processos Biológicos usando Redes de Petri Predicado Transição Diferenciais. Master's thesis, UFU, Universidade Federal de Uberlândia.

[Tricas e Ezpeleta 2003] Tricas, F. e Ezpeleta, J. (2003). Some results on siphon computation for deadlock prevention in resource allocation systems modeled with Petri nets. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03)*, pp. 322–329.

[Tricas e Martinez 1995] Tricas, F. e Martinez, J. (1995). An extension of the liveness theory for concurrent sequential processes competing for shared resources. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century (SMC'95)*, pp. 3035–3040.

[Villani 2000] Villani, E. (2000). Abordagem Híbrida para Modelagem de Sistemas de Ar Condicionado em Edifícios Inteligentes. Master's thesis, USP, Universidade de São Paulo.

[Wang et al. 2009] Wang, Y., Lafortune, S., Kelly, T., Kudlur, M., e Mahlke, S. (2009). The Theory of Deadlock Avoidance via Discrete Control. *SIGPLAN Notices*, pp. 252–263.

[Wazlawick 2008] Wazlawick, R. S. (2008). *Metodologia de Pesquisa para Ciência da Computação*. Elsevier, Rio de Janeiro, RJ, Brasil, Rio de Janeiro, RJ, Brasil, 4th edition.

[Weske 2012] Weske, M. (2012). *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Secaucus, NJ, USA, 2nd edition.

[WfMC 1996] WfMC (1996). Workflow Management Coalition glossary and terminology. Documento: WFMC-TC-1011, http://www.wfmc.org/.

[Wu e Zhou 2003] Wu, N. e Zhou, M. (2003). AGV routing for conflict resolution in AGV systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'03)*, pp. 1428–1433.

[Wu e Zhou 2004] Wu, N. e Zhou, M. (2004). Modeling and deadlock control of automated guided vehicle systems. *IEEE/ASME Transactions on Mechatronics*, pp. 50–57.

[Wu e Zhou 2005] Wu, N. e Zhou, M. (2005). Modeling and deadlock avoidance of automated manufacturing systems with multiple automated guided vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, pp. 1193–1202.

[Wu e Zhou 2007] Wu, N. e Zhou, M. (2007). Shortest Routing of Bidirectional Automated Guided Vehicles Avoiding Deadlock and Blocking. *IEEE/ASME Transactions on Mechatronics*, pp. 63–72.

[Xiong et al. 2009] Xiong, P., Zhou, M., e Pu, C. (2009). A Petri Net Siphon Based Solution to Protocol-Level Service Composition Mismatches. In *Proceedings of the IEEE International Conference on Web Services (ICWS'09)*, pp. 952–958.

[Xu et al. 2009] Xu, L., Liu, H., Wang, S., e Wang, K. (2009). Modelling and analysis techniques for cross-organizational workflow systems. *Systems Research and Behavioral Science*, pp. 367–389.

[Yamalidou e Kantor 1991] Yamalidou, E. C. e Kantor, J. C. (1991). Modeling and optimal control of discrete-event chemical processes using petri nets. *Computers and Chemical Engineering*, pp. 503–519.

[Yin 2008] Yin, R. K. (2008). *Case Study Research: Design and Methods (Applied Social Research Methods)*. SAGE Publications, Thousand Oaks, CA, EUA, 4th edition.