ROGÉRIO A. RONDINI

UMA ARQUITETURA BASEADA EM ESPAÇO DE TUPLAS PARA REDES IMS

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

São Paulo
2012

ROGÉRIO A. RONDINI

UMA ARQUITETURA BASEADA EM ESPAÇO DE TUPLAS PARA REDES IMS

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de Concentração:

Sistemas Digitais

Orientador:

Dra. Graça Bressan

São Paulo
2012

FICHA CATALOGRÁFICA

Rondini, Rogério Augusto
Uma Arquitetura Baseada em Espaço de Tuplas para Redes IMS/ R. A. Rondini. São Paulo, 2012.
127 p.

Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo. Engenharia de Computação e Sistemas Digitais.

1. Redes de Computadores 2. Sistemas Distribuídos I. Universidade de São Paulo. Escola Politécnica. Engenharia de Computação e Sistemas Digitais. II. t.

Dedico a meu Pai, falecido meses antes da conclusão, e à minha Mãe que se manteve forte nesse ano tão difícil; ambos, fundamentais na minha formação.

Dedico à minha esposa Cristina e meus filhos Lorraine e Luccas.

"... Já não sonho, hoje faço com meu braço
o meu viver."
Fernando Brant & Milton Nascimento

AGRADECIMENTOS

Primeiramente à Deus; à toda minha família; à minha esposa Cristina e meus filhos Lorraine e Luccas pela compreensão e apoio, principalmente durante este ano em que dediquei grande parte do meu tempo à conclusão do trabalho.

À minha orientadora, Profa. Dra. Graça Bressan, pela paciência, por acreditar no trabalho, pela orientação nos momentos mais críticos, e, principalmente, pela fundamental sugestão para utilização de Redes de Petri Coloridas para demonstração dos resultados, sem a qual certamente o trabalho não seria concluído.

Aos professores Dra.Regina de Melo Silveira e Dr. Francisco Reverbel, pelas valiosas contribuições durante o processo de qualificação.

Ao amigo Leonardo Gonçalves (Léo) pelas discussões sobre espaço de tupla.

À Profa. Cristina Borba pelo excelente trabalho de revisão de artigos em Língua Inglesa, e pelas valiosas dicas e técnicas de apresentação oral.

Ao Prof. Dr. Paulo Barreto por um dia ter dedicado seu tempo na adaptação da macro ABNTex para os padrões da Poli, e ao grupo PoliGNU por manter esse trabalho atualmente. Isso facilitou muito a minha vida durante a confecção deste texto.

Aos professores das disciplinas que cursei, Prof. Luiz Barco, Prof. Dr. Alfredo Goldman (IME), Profa. Dra. Liria Matsumoto Sato, Prof. Dr. Denis Gabos e Prof. Dr. Wilson Vicente Ruggiero, os quais foram de grande importância para a formulação do trabalho.

Por fim, à todos que porventura deixei de mencionar mas contribuíram direta ou indiretamente para que eu pudesse ter êxito na conclusão desta tese de doutorado.

RESUMO

A arquitetura *IP Multimedia Subsystem*, proposta pelo consórcio *3rd Generation Partnership Project* como base para o suporte à convergência entre telefonia móvel e a Internet, define uma série de elementos arquiteturais, entre os quais, o componente *Call Session Control Function* e o protocolo *Session Initiation Protocol*. *Session Initiation Protocol* é um protocolo da camada de aplicação utilizado para estabelecer, modificar e terminar sessões multimídia entre dispositivos. Em redes baseadas na arquitetura *IP Multimedia Subsystem*, o *Session Initiation Protocol* é o responsável pela comunicação entre dispositivos e a rede, e entre os componentes responsáveis pelo gerenciamento de sessão. Nos últimos anos, estudos detectaram degradação de desempenho em redes baseadas na arquitetura *IP Multimedia Subsystem* em função das características centralizadas do *Session Initiation Protocol* e dos componentes de gerenciamento de sessão. Este trabalho apresenta uma arquitetura distribuída para redes baseadas em *IP Multimedia Subsystem*, tendo como fundamento o paradigma de computação paralela baseado em espaço de tuplas onde os servidores são organizados em uma rede P2P, com objetivo de prover uma infraestrutura escalável e tolerante a falhas. A validação da arquitetura em termos de desempenho e escalabilidade se deu através de modelagem formal e simulação com Redes de Petri Coloridas.

Palavras-chave: Espaço de Tuplas. Redes P2P. Redes IMS. Protocolo SIP. Redes de Petri Coloridas. Modelagem e Simulação.

ABSTRACT

The IP Multimedia Subsystem architecture, proposed by the 3rd Generation Partnership Project consortium as basis to support the convergence between mobile networks and the Internet, defines a set of architectural elements, among them, the Call Session Control Function and the Session Initiation Protocol. The Session Initiation Protocol is an application layer protocol used to establish, modify and terminate sessions between devices. On the IP multimedia subsystem based network, the Session Initiation Protocol play a key role on the communication between devices and the network, and between session management components. In the last years, studies have detected a performance bottleneck on IP multimedia subsystem networks due to centralized characteristic of the Session Initiation Protocol and in Session Control components. This work shows a distributed architecture for IP Multimedia Subsystem networks based on the tuple space paradigm, and the servers structured in a P2P network, aiming to achieve a scalable and fault-tolerant infrastructure. The validation of the architecture on the performance and scalability took place through the Coloured Petri Net formal modeling and simulation.

Keywords: Tuple Space. P2P Network. IMS Network. SIP Protocol. Coloured Petri Nets. Modeling and Simulation.

SUMÁRIO

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

Lista de Símbolos e Variáveis

1 Introdução p. 15

1.1 Motivação . . . . . p. 16

1.2 Justificativas . . . . . p. 17

1.3 Objetivos . . . . . p. 18

1.4 Metodologia . . . . . p. 19

1.5 Contribuições . . . . . p. 20

1.6 Estrutura da Tese . . . . . p. 20

2 Arquitetura IMS p. 22

2.1 Componentes de Controle de Sessão . . . . . p. 23

2.2 Bases de dados de registro de usuário . . . . . p. 24

2.3 Protocolos de Comunicação . . . . . p. 24

2.4 Requisitos . . . . . p. 27

2.5 Considerações . . . . . p. 27

3 Espaço de Tuplas p. 29

3.1 Fundamentos de Espaço de Tuplas e a linguagem Linda . . . . . p. 29

3.2 JavaSpace™ . . . . . p. 31

3.3 L^2imbo . . . . . p. 33
3.4 Lime e LighTS . . . . . p. 34
3.5 Considerações . . . . . p. 36

4 Trabalhos Relacionados p. 37
4.1 Decentralização da Arquitetura IMS . . . . . p. 37
4.2 Peer to Peer SIP (P2PSIP) . . . . . p. 37
4.3 Espaço de Tuplas . . . . . p. 39
4.3.1 Execution time prediction in DSM-based mobile grids . . . p. 39
4.3.2 Comet: A Scalable Coordination Space for Decentralized Distributed Environments . . . . . p. 39
4.4 Modelagem Formal de redes IMS/SIP com CPN . . . . . p. 40
4.4.1 Análise funcional do protocolo SIP . . . . . p. 40
4.5 Considerações . . . . . p. 40

5 Uma arquitetura baseada em Espaço de Tuplas para redes IMS p. 42
5.1 Definição da Arquitetura . . . . . p. 42
5.1.1 SIPSpace: SIP baseado em espaço de tuplas . . . . . p. 44
5.1.2 Espaço de tuplas sobre redes P2P . . . . . p. 45
5.1.3 Aspectos Operacionais . . . . . p. 48
5.1.4 Aspectos de Implementação . . . . . p. 49
5.2 Modelagem Formal . . . . . p. 52
5.2.1 Modelo CPN para o SIPSpace . . . . . p. 52
5.2.2 Modelo CPN para a Rede IMS Baseada em Espaço de Tuplas p. 60

6 Análise Formal baseada em Redes de Petri Coloridas p. 76
6.1 Análise Funcional do SIPSpace . . . . . p. 76
6.1.1 Análise de Espaço de Estado . . . . . p. 76

6.1.2 Considerações . . . . . p. 79
6.2 Análise de Desempenho da Arquitetura baseada em Espaço de Tuplas p. 79
6.2.1 Métricas e Coleta de Dados . . . . . p. 79
6.2.1.1 Taxa de utilização do CSCF . . . . . p. 79
6.2.1.2 Tempo médio de residência nas operações de Registro e Início de Sessão . . . . . p. 81
6.2.1.3 Tempo médio de espera no LTS . . . . . p. 82
6.2.1.4 Tempo médio de espera no DTS . . . . . p. 83
6.2.1.5 Média de tuplas em espera no LTS . . . . . p. 84
6.2.1.6 Média de tuplas em espera no DTS . . . . . p. 85
6.2.1.7 Vazão do sistema . . . . . p. 86
6.2.1.8 Escalabilidade . . . . . p. 86
6.2.2 Cenário Base . . . . . p. 89
6.2.3 Resultados . . . . . p. 91
6.2.3.1 Simulação 01 - Variação do número de Servidores IMS habilitados . . . . . p. 91
6.2.3.2 Simulação 02 - Variação da Carga . . . . . p. 96
6.2.3.3 Simulação 03 - Comparação com arquitetura padrão p. 102
6.2.4 Considerações . . . . . p. 104

7 Conclusões p. 106
7.1 Considerações Finais e Principais Contribuições . . . . . p. 106
7.2 Limitações . . . . . p. 107
7.3 Trabalhos Futuros . . . . . p. 107

Referências p. 109

Apêndice A – Introdução à Modelagem e Simulação com Redes de Petri

Coloridas p. 113

A.1 Definição Formal . . . . . p. 113

A.2 Exemplo de Modelo CPN . . . . . p. 114

A.3 Simulação e Ferramentas de Análise . . . . . p. 117

A.3.1 Espaço de Estado . . . . . p. 117

A.3.2 Análise de Desempenho . . . . . p. 117

Apêndice B – Funções de Monitoração e Coleta de Dados p. 119

Apêndice C – Modelo CPN para Arquitetura IMS padrão p. 126

LISTA DE ILUSTRAÇÕES

1 Visão geral da arquitetura IMS (KHLIFI; GRÉGOIRE, 2008) . . . . . p. 22

2 Visão Geral da arquitetura do protocolo SIP (PEREA, 2008) . . . . . p. 25

3 Visão geral do JavaSpaces™(FREEMAN; HUPFER; ARNOLD, 1999) . . p. 31

4 Arquitetura L^2imbo (WADE, 1999) . . . . . p. 33

5 Visão geral da arquitetura baseada em espaço de tuplas. . . . . p. 42

6 Visão geral da Arquitetura do SIPSpace . . . . . p. 44

7 Topologia da arquitetura baseada em espaço de tuplas . . . . . p. 46

8 Modo de operação do servidor IMS (simplificado) . . . . . p. 48

9 Arquitetura JAIN-SIP (O'DOHERTY; RANGANATHAN, 2003) . . . . . p. 50

10 Arquitetura de Implementação . . . . . p. 52

11 Modelo CPN para Transação INVITE - Validação Funcional . . . . p. 56

12 Modelo CPN Hierárquico representando a rede IMS baseada em espaço de tuplas . . . . . p. 68

13 Modelo CPN Hierárquico representando um Servidor IMS da rede . p. 69

14 Modelo CPN representando a chegada de requisições (*Arrival Node*) p. 70

15 Modelo CPN representando os dispositivos conectados à rede (*Devices*) . . . . . p. 71

16 Modelo CPN para o espaço de tuplas local (LTS) . . . . . p. 72

17 Modelo CPN para o CSCF e o Event Processor . . . . . p. 73

18 Cenário para análise de desempenho e construção dos modelos CPN p. 90

19 Simulação 01 - Vazão X Número de Servidores IMS . . . . . p. 93

20 Simulação 01 - Tempos de Resposta X Número de Servidores IMS . p. 93

21 Simulação 01 - Escalabilidade e Eficiência com base no SpeedUp . p. 94

22 Simulação 01 - Escalabilidade com base na equação de produtividade p. 95
23 Simulação 01 - Média de Tuplas no DTS . . . . . p. 96
24 Simulação 01 - Tempo de Espera no DTS . . . . . p. 96
25 Simulação 01 - Taxa de utilização do CSCF por Servidor IMS Habilitado . . . . . p. 97
26 Simulação 02 -Tempos de resposta por taxa de chegada . . . . . p. 98
27 Simulação 02 - Tempos de resposta por taxa de chegada (95%) . . . p. 99
28 Simulação 02 - Média de tupla no espaço por taxa de chegada . . . p. 99
29 Simulação 02 - Tempo de espera no espaço por taxa de chegada . . p. 100
30 Simulação 02 - Vazão por taxa de chegada . . . . . p. 100
31 Simulação 02 - Índice de escalabilidade com base na vazão . . . . . p. 101
32 Simulação 02 - Taxa de carga do CSCF . . . . . p. 101
33 Simulação 03 - Comparação entre os tempos de resposta e vazão . . p. 102
34 Simulação 03 - Comparação ente as taxas de carga nos servidores CSCF . . . . . p. 103
35 Modelo CPN Exemplo . . . . . p. 115
36 Modelo CPN para a Arquitetura IMS padrão . . . . . p. 127

LISTA DE TABELAS

1 Protocolo SDP - Descrição de Sessão . . . . . p. 26

2 Protocolo SDP - Descrição de Tempo . . . . . p. 26

3 Protocolo SDP - Descrição da Mídia . . . . . p. 26

4 Declarações para o modelo CPN da Figura 11 . . . . . p. 57

5 Conjunto de cores para os modelos CPN da arquitetura baseada em espaço de tuplas . . . . . p. 61

6 Declarações de variáveis para os modelos CPN da arquitetura baseada em espaço de tuplas . . . . . p. 62

7 Declarações globais para os modelos CPN da arquitetura baseada em espaço de tuplas . . . . . p. 62

8 Parâmetros utilizados durante as simulações . . . . . p. 92

9 Resultados da Simulação 01 (a) . . . . . p. 92

10 Resultados da Simulação 01 (b) . . . . . p. 92

11 Resultados da Simulação 02 (a) . . . . . p. 97

12 Resultados da Simulação 02 (b) . . . . . p. 97

13 Resultados da Simulação 02 (c) . . . . . p. 98

14 Resultados da Simulação 02 (d) . . . . . p. 98

15 Resultados da Simulação 03 (Arquitetura Padrão) . . . . . p. 102

16 Resultados da Simulação 03 (Espaço de Tuplas) . . . . . p. 102

17 Resultados obtidos em (LUO et al., 2009) . . . . . p. 103

LISTA DE ABREVIATURAS E SIGLAS

3GPP	3rd Generation Partnership
CSCF	Call Session Control Function
P-CSCF	Proxy Call Session Control Function
S-CSCF	Serving Call Session Control Function
I-CSCF	Interrogate Call Session Control Function
CPN	Coloured Petri Net
CTS	Centralized Tuple Space
DHT	Distributed Hash Table
DNS	Domain Name Service
DTS	Distributed Tuple Space
HSS	Home Subscriber Server
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
LTS	Local Tuple Space
QoE	Quality of Experience
QoS	Quality of Service
QPN	Queuing Petri Nets
NGN	Next Generation Network
PDF	Policy Decision Function
RTP	Real-time Transport Protocol
SCC	Strongly-Connected-Component Graph
SDP	Session Description Protocol
SIP	Session Initiation Protocol

SLF	Subscriber Location Function
UA	User Agent
UE	User Equipment
URI	Universal Resource Identifier

LISTA DE SÍMBOLOS E VARIÁVEIS

λ	Taxa de chegada de requisições
ψ	Escalabilidade
U_{CSCF_i}	Taxa de utilização dos CSCF
mr	Média de requisições atendidas pelo CSCF
Rr	Tempo de residência para operações de registro
Rs	Tempo de residência para operações de estabelecimento de sessão
V	Vazão (*Throughput*) do sistema
$E(w)_{LTS_i}$	Tempo médio de espera no LTS
$E(w)_{DTS}$	Tempo médio de espera no TTS
$E(n_q)_{LTS_i}$	Média de tuplas no LTS
$E(n_q)_{DTS}$	Média de tuplas no DTS
mt	Tempo global do simulador (*Model Time*)
Sp	SpeedUp
Ep	Eficiência
$F(p)$	Produtividade
$f_{QoS_p}(\sigma_1, .., \sigma_n)$	Função de qualidade de serviço

1 INTRODUÇÃO

A união entre telefonia móvel e a Internet abriu um novo cenário para o mercado de serviços móveis. Nesse novo cenário, o modelo de negócio passou a considerar a participação de novos atores na cadeia de valor, como por exemplo, provedores de serviço e provedores de conteúdo, (PASCOTTO et al., 2008; OH et al., 2007).

Operadoras de telefonia são as responsáveis pela infraestrutura e tecnologias de comunicação, enquanto novos serviços e entrega de conteúdo são oferecidos pelos parceiros, sendo, cada parceiro, especializado em um tipo de serviço ou conteúdo, tais como, jogos, entretenimento, notícias, entre outros.

Para suportar esse novo modelo de negócio, o *3rd Generation Partnership* (3GPP) criou a arquitetura *IP Multimedia Subsystem* (IMS), (CAMARILLO; GARCÍA-MARTÍN, 2004).

A Arquitetura IMS define uma série de entidades e funcionalidades, tais como gerenciamento de sessão e roteamento, base de dados com informações dos usuários, servidores de aplicação, funções de interconexão, suporte e tarifação. Adicionalmente, a arquitetura IMS define um conjunto de protocolos, tais como *Session Initiation Protocol* (SIP), *Session Description Protocol* (SDP), *Real-time Transport Protocol* (RTP), *Diameter*, entre outros, cada um com características e finalidades específicas. O principal protocolo utilizado no controle de sessão é o SIP, (ROSENBERG et al., 2002), responsável pelo estabelecimento, manutenção e finalização de sessões multimídia.

Apesar de estabelecer responsabilidades bem definidas em termos de elementos e suas funcionalidades, a arquitetura IMS apresenta algumas características centralizadas, as quais afetam o desempenho em cenários de larga escala.

O objeto de pesquisa desta tese é identificar tecnologias de rede e sistemas distribuídos que possibilitem a decentralização da arquitetura IMS, em especial, seus componentes responsáveis pelo gerenciamento de sessão, e propor uma arquitetura

escalável e tolerante a falhas, sem que haja perda das funcionalidades especificadas pelo 3GPP.

1.1 Motivação

Conforme apresentado nas especificações ([TS 23.218], 2008), ([TS 23.228], 2008) e ([TS 24.229], 2012), e discutido em (CAMARILLO; GARCÍA-MARTÍN, 2004), (POIKSELKA et al., 2006), (KHLIFI; GRÉGOIRE, 2008) e (QADEER et al., 2009), um dos principais componentes da arquitetura IMS é o *Call Session Control Function*(CSCF), responsável pelo gerenciamento de sessão. O CSCF é composto por três entidades, o *Serving Call Session Control Function* (S-CSCF), o *Proxy Call Session Control Function* (P-CSCF) e o *Interrogate Call Session Control Function* (I-CSCF), os quais se comunicam através do protocolo SIP.

Conforme discutido em (ABHAYAWARDHANA; BABBAGE, 2007), as operações de registro e estabelecimento de sessão na arquitetura IMS são muito complexas, e ampliam as deficiências do protocolo SIP, uma vez que outros elementos são adicionados à rede e trocam mensagens SIP entre si.

Nos últimos anos, alguns estudos detectaram degradação de desempenho no protocolo SIP quando utilizado em redes IMS, o qual ocorre em função na natureza centralizada do servidor de registro e da carga no elemento S-CSCF causada pelo mecanismo de renovação de registro utilizados por dispositivos em redes IMS (PANDEY et al., 2007) (VINEEL, 2007) (KANG et al., 2007) (MKWAWA; KOUVATSOS, 2008) (LUO et al., 2009).

(PANDEY et al., 2007) concluiu que o atraso no processamento de requisições ocorre no S-CSCF, e ainda argumenta que para otimizar o processamento nesse elemento é necessário um projeto de rede IMS eficiente, diferente do tradicional modelo centralizado.

Em (VINEEL, 2007) as requisições SIP foram separadas em duas categorias: (i) *system-triggered*, requisições geradas automaticamente pelos dispositivos - *User Equipments* (UE), e.g, requisições de registro na rede; e (ii) *user-triggered*, requisições iniciadas a partir da interação do usuário, e.g, requisição de estabelecimento de sessão. A conclusão foi que o principal problema de degradação de desempenho ocorre nas operações de registro em função da sobrecarga no S-CSCF.

O mesmo problema descrito anteriormente foi estudado por (KANG et al., 2007),

o qual identificou que 60% das requisições recebidas pelos servidores SIP são requisições de registro na rede. Adicionalmente, identificou-se que esse comportamento ocorre em função do envio de requisições periódicas de renovação registro.

(MKWAWA; KOUVATSOS, 2008) utilizou o modelo de filas *central server queuing model* para medir a taxa de utilização dos servidores na arquitetura IMS. Os resultados mostraram que a taxa de utilização do S-CSCF foi de 98%, enquanto o P-CSCF e o I-CSCF permaneceram abaixo dos 37%, ou seja, uma alta taxa de utilização do servidor responsável pelo S-CSCF e uma subutilização dos servidores responsáveis pelo I-CSCF e P-CSCF. Esse comportamento causa uma queda acentuada de desempenho durante o estabelecimento de sessões.

(LUO et al., 2009) utilizou *Queuing Petri Nets* (QPN) para analisar o comportamento do protocolo SIP em redes IMS. Os resultados para uma taxa de chegada em torno de 50.000 requisições de estabelecimento de sessão por hora e 5.000 requisições de registro por hora, mostraram que a taxa de utilização de cada servidor permaneceu abaixo dos 35%, sendo o S-CSCF com a maior delas, 34,2%. Na simulação com taxas de chegada acima de 140.000 requisições por hora, o S-CSCF atingiu acima de 90% de utilização, resultando em uma queda significativa de desempenho. Adicionalmente, foi observado que o I-CSCF permaneceu abaixo de 15% de utilização, ou seja, uma sobrecarga no S-CSCF e uma subutilização de recurso no I-CSCF.

Considerando o crescente cenário de utilização de serviços em dispositivos móveis conectados a redes IP, é fundamental que haja uma arquitetura descentralizada e escalável para as redes IMS, bem como uma arquitetura de software flexível para implementação de suas funcionalidades.

1.2 Justificativas

Conforme já mencionado, a convergência entre telefonia móvel e a Internet abriu um novo cenário para o mercado de serviços móveis, agregando os provedores de serviços e conteúdo à cadeia de valor.

Um dos pontos principais desse novo modelo é o gerenciamento de transações em tempo real. Além das operadoras de telefonia, provedores de serviços e conteúdo podem definir suar próprias regras de acesso, por exemplo, operadoras de telefonia podem oferecer desconto a determinados grupos de usuários quando acessando conteúdo

de provedores específicos, enquanto oferecem banda diferenciada para outro grupo de usuários durante um determinado período.

Esse cenário complexo de execução e gerenciamento requer uma infraestrutura de rede capaz de suportar, sem perda de desempenho, a demanda crescente por serviços móveis baseados em redes IP.

A arquitetura proposta neste trabalho prevê a decentralização total dos elementos responsável pelo gerenciamento de sessão definidos na arquitetura IMS, utilizando como base o modelo de programação paralela denominado Espaço de Tuplas.

Em ambientes distribuídos clássicos, a comunicação entre processos ocorre através de canais formados por pares de elementos, (COULOURIS; DOLLIMORE; KINDBERG, 2005). O paradigma de espaço de tuplas é fundamentalmente diferente, já que a comunicação entre processos ocorre unicamente através do espaço de tuplas, ou seja, tuplas são produzidas por um processo produtor para um processo consumidor identificado, encapsulando-se a informação do destino dentro da própria tupla, (GELERNTER, 1985).

Adicionalmente, a arquitetura adota redes P2P estruturadas baseadas em *Distributed Hash Table* (DHT) como base para o espaço de tuplas. Essa abordagem permitira adicionar novos Servidores IMS na rede de forma simples, além de prover um eficiente protocolo para localização de tuplas.

1.3 Objetivos

O objetivo principal deste trabalho é a definição de uma arquitetura descentralizada para redes IMS, mais especificamente para implementação dos componentes responsáveis pelo controle e gerenciamento de sessão, seguindo o paradigma baseado em espaço de tuplas, construído sobre redes P2P. Adicionalmente, validar tal arquitetura sobre aspectos de desempenho e escalabilidade através de modelagem e simulação por redes de petri coloridas - *Colored Petri Nets* (CPN).

Do objetivo geral, podemos desmembrar em três objetivos específicos, detalhados a seguir.

Definição de uma arquitetura distribuída para redes IMS. Criar uma arquitetura descentralizada para redes IMS com base no paradigma de espaço de tuplas, construído sobre redes P2P. Definir a arquitetura de software para implementa-

ção dos componentes IMS com base no modelo descentralizado.

Definição de um modelo de execução distribuído do protocolo SIP. Transformar um modelo baseado no paradigma requisição/resposta com controle centralizado, característico do protocolo SIP, para um modelo descentralizado, onde a comunicação entre elementos (processos) pode ocorrer de forma desconectada e independente do tempo.

Validação Funcional e Análise de Desempenho. Criar modelos utilizando Redes de Petri Coloridas para análise funcional baseada em espaço de estado e analise de desempenho através de simulação.

1.4 Metodologia

A **primeira etapa** da pesquisa consistiu na análise da arquitetura IMS e dos requisitos através de suas especificações, ([TS 23.218], 2008), ([TS 23.228], 2008), ([TS 24.229], 2012), ([TS 22.228], 2008) e ([TS 23.221], 2011), e de referências bibliográficas (CAMARILLO; GARCÍA-MARTÍN, 2004), (POIKSELKA et al., 2006), (KHLIFI; GRÉGOIRE, 2008) e (QADEER et al., 2009).

A **segunda etapa** consistiu de estudos aprofundados sobre os potenciais problemas de desempenho da arquitetura IMS e o protocolo SIP em ambientes de larga escala, conforme discutido em (PANDEY et al., 2007), (VINEEL, 2007), (KANG et al., 2007), (MKWAWA; KOUVATSOS, 2008) e (LUO et al., 2009)

A **terceira etapa** consistiu na definição da arquitetura descentralizada e da proposta de mudança de implementação do protocolo SIP. A ideia de adotar o modelo de espaço de tuplas se deu a partir do conhecimento da especificação JavaSpace™(FREEMAN; HUPFER; ARNOLD, 1999) e (WALDO, 2000), e do trabalho (BALLETTE; LIOTTA; RAMZY, 2005) que utiliza espaço de tuplas para implementar grades computacionais móveis.

A **quarta etapa** consistiu em um estudo aprofundado de modelagem e simulação baseada em redes de petri coloridas, e na definição de modelos CPN para a arquitetura proposta.

Por fim, a **quinta etapa** consistiu na definição de métricas, simulação dos modelos CPN e análise dos resultados.

1.5 Contribuições

Considerando a utilização de espaço de tuplas como uma nova abordagem para implementação de protocolos de rede, este trabalho apresenta a seguintes contribuições inéditas:

1. mudança de paradigma na implementação do protocolo SIP, saindo do modelo tradicional baseado em requisição/resposta para o modelo de comunicação generativa baseado em espaço de tuplas;
2. definição de uma arquitetura distribuída para redes IMS com base no paradigma de espaço de tuplas sobre redes P2P.
3. modelagem e simulação da rede IMS, modelo tradicional e arquitetura proposta, através de redes de petri coloridas, uma ferramenta excepcional para validação de desempenho em sistemas distribuídos.

1.6 Estrutura da Tese

O texto desta tese está estruturado da seguinte forma: o **capítulo 1** apresenta a introdução da tese; o **capítulo 2** apresenta a arquitetura IMS, detalhando os componentes e protocolos responsáveis pelo controle de sessão; o **capítulo 3** apresenta os fundamentos de espaço de tuplas. o **capítulo 4** apresenta os trabalhos relacionados; o **capítulo 5** apresenta a arquitetura baseada em espaço de tuplas para redes IMS e a modelagem formal através de redes de petri coloridas; o **capítulo 6** faz análise funcional através de espaço de estado e a análise de desempenho através de simulações dos modelos CPN; por fim, o **capítulo 7** discute os resultados e benefícios do trabalho, e indica tópicos de pesquisa em aberto para potenciais trabalhos futuros.

Em complemento ao texto da tese, foram adicionados três apêndices: o **apêndice A** apresenta uma introdução à modelagem e simulação com redes de petri coloridas; o **apêndice B** apresenta as funções de monitoração utilizadas para coleta de dados das simulações; por fim, o **apêndice C** ilustra o modelo CPN representando a arquitetura IMS padrão, utilizado para simulação e comparação com a arquitetura baseada em espaço de tuplas.

Em função da publicação de trabalho internacionais, algumas ilustrações e os modelos CPN estão em Inglês.

2 ARQUITETURA IMS

A arquitetura IMS define uma série de entidades e funcionalidades, cada uma com responsabilidades bem definidas, ([TS 23.218], 2008), ([TS 23.228], 2008), ([TS 24.229], 2012), (CAMARILLO; GARCÍA-MARTÍN, 2004), (POIKSELKA et al., 2006), (KHLIFI; GRÉGOIRE, 2008) e (QADEER et al., 2009). A Figura 1 apresenta uma visão geral dessa arquitetura.

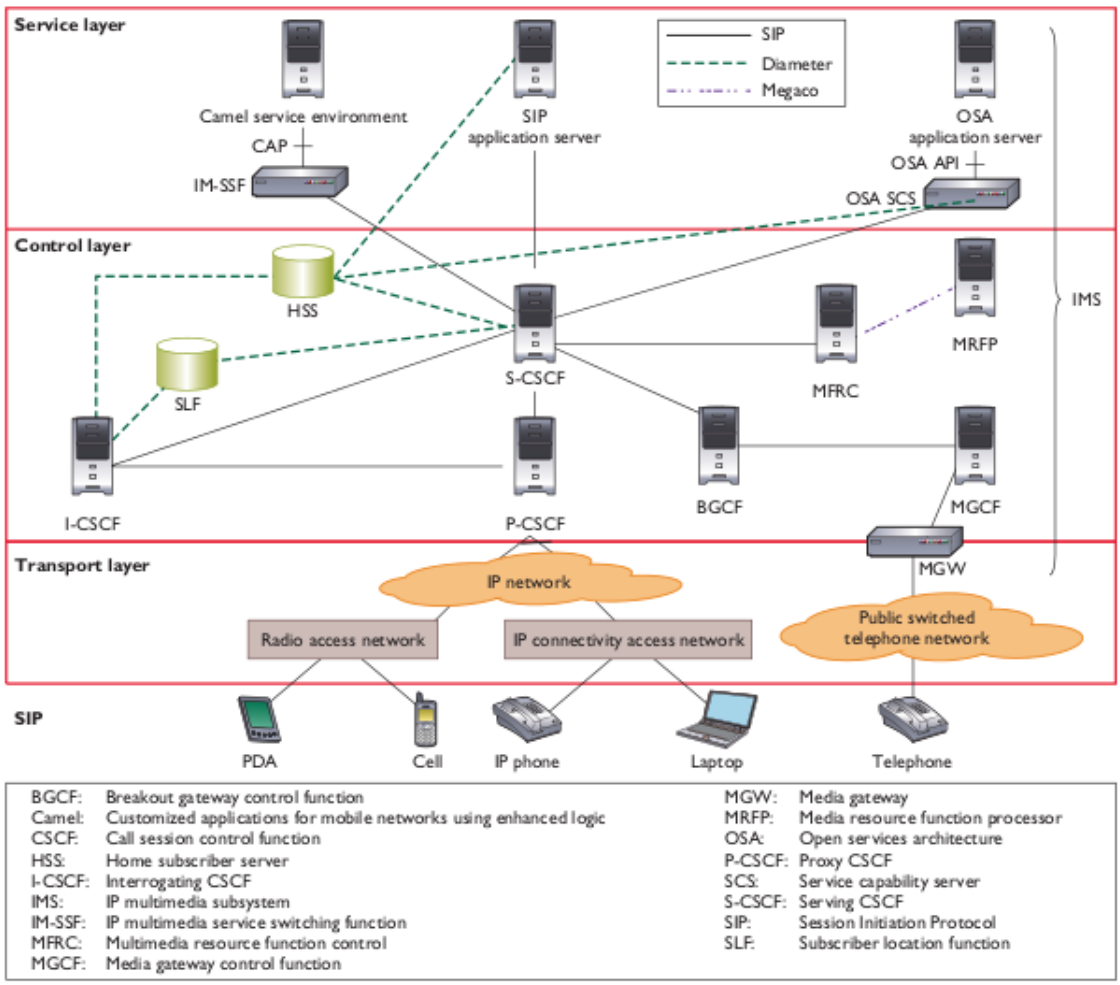

Figura 1: Visão geral da arquitetura IMS (KHLIFI; GRÉGOIRE, 2008)

A arquitetura é subdividida em três camadas principais: (i) camada de transporte, responsável pela conectividade entre a rede IMS e os dispositivos e entre a rede IMS e redes utilizando outras tecnologias; camada de controle, responsável pelo controle de sessão e acesso aos serviços; e (iii) camada de serviço responsável pela execução dos serviços móveis oferecidos pelos provedores.

Considerando os objetivos deste trabalho, serão detalhados apenas os componentes e protocolos relacionados à camada de controle, mais especificamente os componentes responsáveis pelo controle de sessão.

2.1 Componentes de Controle de Sessão

O controle de sessão é parte fundamental da arquitetura IMS. O componente responsável por essa tarefa é o CSCF, onde a troca de informações é implementada através do protocolo SIP. Existem três tipos de componente CSCF, P-CSCF, S-CSCF e I-CSCF, cada um executando tarefas específicas, conforme descrito a seguir.

Proxy-CSCF - O P-CSCF é o primeiro ponto de contato entre um terminal e uma rede IMS. Isso significa que toda requisição partindo de um UE será enviado ao P-CSCF, assim como as mensagens de finalização partindo da rede IMS. Existem quatro atividades designadas ao P-CSCF:

1. Compactação das mensagens SIP, uma vez que o SIP é um protocolo baseado em texto com um número considerável de informações e parâmetros no cabeçalho.
2. Associação de informações de segurança padrão IPSec.
3. Interação com o componente de autorização de acesso *Policy Decision Function* (PDF)
4. Detecção de chamadas emergenciais.

Interrogate-CSCF O I-CSCF é o ponto de entrada para chamadas executadas de redes externas. Ele é exposto através de *Domain Naming Service* (DNS), para que servidores remotos como um P-CSCF em um domínio de rede visitado ou um S-CSCF em um domínio estrangeiro possam utilizá-lo como ponto de entrada.

Serving-CSCF O S-CSCF é um servidor SIP que atua como um servidor de registro (*Registar Server*), tomando decisões de roteamento de requisições, mantendo

estado de sessão e armazenando informações de serviços e usuários. Quando o usuário envia uma requisição de registro, a mensagem é roteada para o S-CSCF o qual obtém as informações de autenticação. Após o registro ser aceito, o S-CSCF obtém as informações do usuário associadas com sua identidade pública, as quais serão utilizadas durante a manutenção da sessão multimídia.

2.2 Bases de dados de registro de usuário

Todos os dados dos usuários registrados são armazenados em um servidor central chamado *Home Subscriber Server* (HSS). O HSS armazena informações de identificação, segurança e localização. Adicionalmente, o HSS provê informações em qual S-CSCF o usuário está registrado.

Em caso de haver necessidade de mais de um HSS, a arquitetura IMS prevê a existência de um *Subscription Locator Function* (SLF) para mapeamento de informações de usuário e servidor HSS. Nesse caso, o SLF sabe informar em qual HSS está armazenada a informação de um determinado usuário.

2.3 Protocolos de Comunicação

SIP é um protocolo da camada de aplicação, utilizado para criar, modificar e finalizar uma sessão (e.g chamada telefônica, distribuição de conteúdo multimedia, etc.) entre um ou mais participantes (ROSENBERG et al., 2002). As principais funções do protocolo SIP são, localização de dispositivos na rede, contactar dispositivos para determinar as características de uma sessão multimídia, troca de informações para o estabelecimento da sessão e gerenciamento de uma sessão em curso.

A especificação do protocolo SIP define um conjunto de entidades, conforme mostra a Figura 2: UA, *Location Server*(*Registar*), *Proxy Server* e *Redirect Server*. UAs são entidades lógicas que podem atuar como cliente ou servidor; por exemplo, um dispositivo móvel é um UA cliente quando envia uma requisição `INVITE` e um UA Servidor quando recebe uma requisição `BYE`. *Registrar Server* é a entidade que recebe as requisições de registro de usuários e as armazena em um serviço de localização. *Proxy Server* é uma entidade intermediária responsável pelo roteamento de requisições. Por fim, o *Redirect Server* gera respostas direcionando os clientes a contactarem

URIs alternativas.

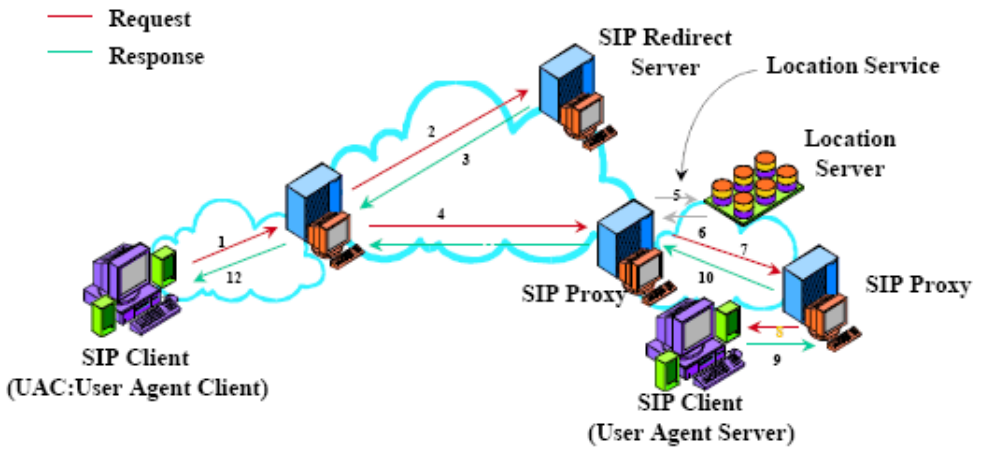

Figura 2: Visão Geral da arquitetura do protocolo SIP (PEREA, 2008)

Além dos elementos físicos, o protocolo SIP oferece um conjunto de operações.

- INVITE - utilizado para estabelecer sessão multimídia entre duas UAs.
- REGISTER - utilizado pelo usuário para notificar a sua entrada na rede com suas informações de localização.
- BYE - utilizado para encerrar uma sessão.
- ACK - utilizado para aceitar uma requisição de abertura de sessão.
- CANCEL - utilizado para cancelar uma requisição de abertura de sessão.
- OPTIONS - utilizado para consultar uma UA sobre sua capacidade e disponibilidade.

Em complemento ao protocolo SIP, o *Internet Engineering Task Force* (IETF) recomenda a utilização do protocolo SDP. O SDP é um protocolo da camada de aplicação utilizado para descrever sessões multimídia, utilizando uma representação padronizada (HANDLEY; JACOBSON; PERKINS, 2006).

Uma mensagem SDP contém três níveis de informação, e aparecem na ordem apresentada a seguir:

1. *Descrição de sessão* - inclui informações como identificador de sessão, endereço IP, informações de contato, entre outros.

2. *Descrição de tempo* - informa sobre data/hora de início e término da sessão.

3. *Formato e tipo de mídia* - informa sobre protocolo de transporte utilizado, informações de conexão, largura de banda, chave de criptografia, entre outros.

As Tabelas 1, 2 e 3 resumem as informações contidas em uma mensagem SDP.

Tabela 1: Protocolo SDP - Descrição de Sessão

Atributo	**Descrição**
v=	versão do protocolo
o=	origem e ID da sessão
s=	nome da sessáo
i=*	informações
u=*	URI
e=*	endereço de e-mail
p=*	número telefone
c=*	informação de conecção
b=*	zero ou mais informações de largura de banda
z=*	time zone
k=*	chave de criptografia
a=*	zero ou mais linhas de descrição de mídia

Tabela 2: Protocolo SDP - Descrição de Tempo

Atributo	**Descrição**
t=	tempo de atividade da sessão
r=*	zero ou mais repetições

Tabela 3: Protocolo SDP - Descrição da Mídia

Atributo	**Descrição**
m=	endereço de transporte
i=*	título
c=*	informação de conexão
b=*	largura de banda
k=*	chave de criptografia
a=*	zero ou mais linhas de descrição da mídia

2.4 Requisitos

A arquitetura IMS foi definida com base nos requisitos arquiteturais definidos em ([TS 23.221], 2011).

1. Conectividade através de redes IP (IPv4 e IPv6)
2. Independência de acesso
3. Qualidade de serviço - *Quality of Service* (QoS) - no acesso a serviços
4. Comunicação segurança
5. Suporte a *roaming*
6. Integração com outras tecnologias
7. Estilo arquitetural baseado em camadas

Adicionalmente, (AGRAWAL et al., 2008) apresenta alguns requisitos de escalabilidade que devem ser considerados nos projetos de rede IMS.

O protocolo SIP é o elemento chave para possibilitar comunicação em redes IMS. Através do SIP, usuários podem estabelecer comunicação independente do tipo de conteúdo e número de participantes, além da heterogeneidade de dispositivos, tais como telefones celulares, computadores, TV digital, entre outros.

Conforme já mencionado, um dos principais problemas de comunicação é a descoberta do nó na qual um usuário será encontrado. O protocolo SIP endereça essa questão através do mapeamento entre AoR e URI de contato do usuário. Na arquitetura padrão do SIP, o elemento responsável por esse mapeamento é o Servidor de Registro, um elemento totalmente centralizado.

Nesse sentido, considerando a perspectiva de demanda crescente por serviços baseados em IMS, um protocolo com característica centralizada, tal como o SIP, poderá inviabilizar o acesso ubíquo à Internet proposto pelo 3GPP.

2.5 Considerações

A arquitetura IMS apresenta uma série de entidades, funcionalidades e protocolos com responsabilidades bem definidas. No entanto, é importante ressaltar que a arqui-

tetura IMS é agnóstica em termos de implementação, ou seja, ela **define O QUE** é necessário em termos de elementos e funcionalidades, e quais protocolos devem ser utilizados, **mas não diz COMO** devem ser implementados.

O principal problema é que as implementações tendem a seguir à risca o modelo arquitetural proposto, o que torna a arquitetura IMS centralizada e com os problemas de desempenho apresentados anteriormente.

Nesse sentido, cabe a definição de uma arquitetura de implementação descentralizada e escalável, e não a mudança da arquitetura IMS.

3 ESPAÇO DE TUPLAS

3.1 Fundamentos de Espaço de Tuplas e a linguagem Linda

O conceito inicial de espaço de tuplas foi proposta por David Gelernter em meados dos anos 80, a partir do desenvolvimento da linguagem Linda (GELERNTER, 1985; CARRIERO; GELERNTER, 1989).

A linguagem Linda implementa o modelo de programação paralela denominado *Generative Communication*[1], e foi desenvolvida tendo como base o paradigma de Memória Associativa[2]

O modelo *Generative Communication* surgiu como uma alternativa aos três principais modelos existentes à época, variáveis compartilhadas (*shared memory*), passagem de mensagem (*message passing*) e operações remotas (*remote operations*). A principal diferença do modelo generativo é o desacoplamento entre processos (CARRIERO; GELERNTER; MATTSON, 1992). Isso ocorre em função da natureza de comunicação indireta entre processos, ao contrário dos três modelos apresentados anteriormente, onde a comunicação entre processos ocorre de maneira direta. Isso significa dizer que se dois processos precisam se comunicar, eles não trocam mensagens ou compartilham variáveis; em vez disso, o processo produtor insere uma tupla no espaço, e o processo consumidor consome a tupla do espaço (CARRIERO; GELERNTER, 1989).

De maneira geral, podemos considerar um espaço de tuplas como sedo um "saco"de tuplas onde qualquer máquina pertencente a um *cluster* pode realizar operações, sejam operações síncronas ou assíncronas. Formalmente, um espaço de tuplas é caracterizado como uma abstração de um ambiente computacional baseado no conceito de uma única memória virtual compartilhada, responsável por armazenar as tu-

[1] Ao longo do texto, esse termo não será traduzido

[2] Memória que é endereçada em função do conteúdo, tendo uma valor associado a uma chave.

plas a serem processadas. Suponhamos a existência de dois processos, P1 e P2 e um espaço de tuplas T1; para enviar dados a P2, o processo P1 gera tuplas e as coloca em T1; para consumir a tupla gerada por P1, o processo P2 faz acesso ao espaço T1 e recupera a tupla.

Existem dois tipos de túplas, tuplas de processos (*process tuples*) e tuplas de dados (*data tuples*). As tuplas de processos referem-se às tarefas em execução; estas, trocam dados gerando e consumindos as túplas de dados. As túplas de dados são estruturas de dados formadas por uma coleção de dados (atributos) de tipos distintos. Essa estrutura pode ser associada facilmente a objetos de linguagens Orientadas a Objetos como Java/C++ ou a estruturas (*struct*) da linguagem C.

Existem três operações fundamentais que podem ser executadas em um espaço de tuplas, *out()*, *in()* e *read()*. Tais operações devem ser executadas de forma atômica.

1. out(): A operação *out* pode ser formalizada como sendo:

   $$out(N, P2,...Pn) \tag{3.1}$$

   onde, `P2,..,Pn` é uma lista de parâmetros que podem ou não ter valor, e `N` é a chave que identifica a tupla no espaço. A execução da operação *out* causa inserção da tupla em um espaço de tuplas.

2. in(): A operação *in*, semelhante à operação *out*, é formalizada como sendo:

   $$in(N, P2,...Pn) \tag{3.2}$$

   onde, `P2,..,Pn` é uma lista de parâmetros que podem ou não ter valor, e `N` é a chave que identifica a túpla no espaço. Se a tupla `N` existir no espaço, esta será removida e o processo continua. Caso não exista a tupla `N` no espaço, o processo será suspenso até que uma tupla seja incluída.

3. read(): A operação *read* é identica à operação *in*, exceto pelo fato de que a execução desta operação não remove a tupla do espaço.

As operações de leitura (e.g *in(..)* e *read(..)*) realizadas no espaço podem utilizar o conceito de *match*. Nesse caso, um processo consumidor lê as tuplas de acordo com

critérios passados na operação de leitura. Por exemplo, a execução da operação `in("a string", ?f, ?i, "b string")` causa uma busca no espaço por tuplas de quatro elementos; o primeiro `"a string"` e o último `"b string"`, e os dois parâmetros do meio como variáveis `f` e `i`. Caso não exista nenhuma tupla no espaço com as características informadas, a operação *in(..)* é bloqueada até aparecer uma tupla (CARRIERO; GELERNTER, 1989).

3.2 JavaSpace™

Após a popularização da linguagem Linda no meio acadêmico nos anos 80, o espaço de tuplas ficou esquecida, ressurgindo no final dos anos 90 com a especificação JavaSpaces™pela Sun Microsystems, (WELLS; CHALMERS; CLAYTON, 2004) (FREEMAN; HUPFER; ARNOLD, 1999), com o aval da equipe responsável pela linguagem Linda da Universidade de Yale. A especificação JavaSpaces™foi produzida pela Sun como parte do projeto Jini (WALDO, 2000).

JavaSpaces™é uma especificação para implementação de aplicações distribuídas, baseada em espaço de tupla (FREEMAN; HUPFER; ARNOLD, 1999). Diferente dos tradicionais modelos de programação distribuída baseados em passagem de mensagem ou RMI, o modelo do *JavaSpaces* utiliza uma coleção de processos que cooperam através de fluxos de objetos que são inseridos/removidos de um ou mais espaço, conforme mostra a Figura 3.

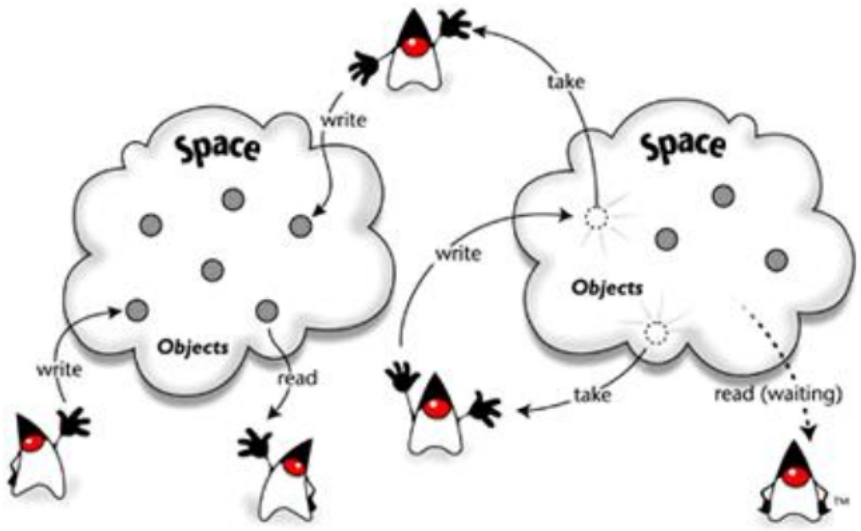

Figura 3: Visão geral do JavaSpaces™(FREEMAN; HUPFER; ARNOLD, 1999)

Assim como na especificação básica da linguagem Linda, os objetos não se comunicam diretamente. Aplicações e processos interagem com o espaço através de um conjunto simples contendo quatro operações:

1. *write()* Insere objetos no espaço
2. *take()* Recupera objetos do espaço. Existe uma variante desta operaçõa chamada *takeIfExists()*.
3. *read()* Lê objetos do espaço, sem removê-los. Também possui uma variante chamada *readIfExists()*.
4. *notify()* Notifica um objeto específico quando tuplas inseridas no espaço atendem a um determinado critério.

É importante notar que processos não modificam objetos no espaço ou acessam seus métodos diretamente. Para tal, um processo precisa recuperar um objeto do espaço através das operações *take()* ou *read()* e, se necessário, inserí-lo novamente no espaço.

Para que um objeto Java se torne uma tupla, é necessário implementar a interface `net.jini.core.entry.Entry` disponibilizada pela especificação Jini™(WALDO, 2000), conforme o fragmento de código a seguir.

```
public class EntryImpl implements Entry {
  public String nome;
  public String id;
  public DummyEntry() {
  }
  public DummyEntry(String nome) {
      this.nome = nome;
  }
}
```

O código a seguir ilustra algumas operações no espaço.

```
1   Entry template = new EntryImpl("Hello Word");
2   Transaction.Created myTxnC = TransactionFactory.create(theTxnMgr, 5000);
```

```
3   Transaction myTxn = myTxnC.transaction;

   // tenta obter uma tupla do espaco baseado no template
6   Entry tupla = space.take(template, myTxn, timeout);
7   if (tupla != null) {
        // faz alguma coisa
        // devolve tupla para o espaco
10        space.write(myResult, myTxn, Lease.FOREVER);
11  }
12  myTxn.commit();
```

Conforme observado, inicialmente um template de um objeto `Entry` é criado com o nome "Hello Word"(linha 1). A operação *take(..)* recupera uma tupla do espaço baseado no template cujo nome é "Hello Word"(linha 6). Se existir uma tupla, pode-se realizar operações sobre essa tupla; após o processamento a tupla é devolvida no espaço através da operação *write(..)* (linha 10).

3.3 L²imbo

Em (WADE, 1999), foi proposta uma plataforma para aplicações móveis distribuídas chamada L²imbo.

A plataforma L²imbo apresenta uma série de extensões em relação ao modelo original proposto na linguagem Linda. Sua principal característica é a definição de múltiplos protocolos para implementação de espaço de tuplas, cada um deles responsável pela implementação adequada das operações passíveis de utilização na plataforma, e.g. *in* (inclusão), *out* (remoção) e *rd* (leitura). Os protocolos são classificados como Distribuído, Centralizado e Local, conforme a Figura 4.

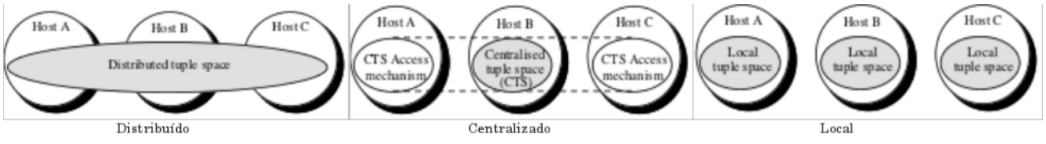

Figura 4: Arquitetura L²imbo (WADE, 1999)

Distribuído - *Distributed Tuple Space* (DTS) Projetado para oferecer escalabilidade e maximizar a disponibilidade de tuplas em aplicações distribuídas. Apesar

dessa característica ser a base do conceito de espaço de tuplas, a novidade na plataforma L^{2}imbo, pelo menos à época em que foi criado, é o fato de existir uma mecanismos para garantir a retirada única de tuplas do espaço, ou seja, apesar da tupla estar replicada em diversos servidores, será lida apenas por um deles.

Centralizado - *Centralized Tuple Space* (CTS) Em contraste ao modelo Distribuído, no modelo Centralizado um dos nós é escolhido para armazenar as tuplas. Os outros nós utilizam um mecanismo de acesso ao CTS para ler e gravar tuplas.

Local - *Local Tuple Space* (LTS) O modelo Local é o mais simples de todos. Cada nó possui um espaço de tupla local. Nesse modelo não existe acesso à rede. O objetivo do LTS é compartilhar informações entre aplicações localizadas em um mesmo nó.

A interface de programação da plataforma L^{2}imbo segue o padrão original da API Linda, com algumas extensões.

3.4 Lime e LighTS

Lime (Linda In a Mobile Environment), (MURPHY; PICCO; ROMAN, 2006), criou uma nova abordagem para desenvolvimento de aplicações móveis baseadas em espaço de tuplas. O modelo proposto no Lime assume um conjunto de nós denominados *containers*, onde agentes são executados. Os agentes podem se mover de um nó para outro.

Cada agente pode possuir múltiplos espaços de tuplas, que por sua vez, podem ser compartilhados com outros agentes em uma mesma faixa de comunicação.

A interface de programação do Lime segue o mesmo padrão definido pelo modelo proposta inicialmente em Linda, porém, as operações extendem o modelo original acrescentando um parâmetro representando o identificador do agente responsável por receber a tupla.

LighTS, (PICCO; BALZAROTTI; COSTA, 2005), é um *framework* que pode ser utilizado como base para o desenvolvimento de aplicações baseadas em espaço de tuplas. Ele não é um espaço de tupla propriamente dito, e sim um conjunto de elementos (objetos Java) que podem ser implementados ou estendidos.

Inicialmente, o LighTS foi desenvolvido para ser o núcleo do Lime, porém, percebeu-se que ele pode ser utilizado em outros modelos de espaço de tuplas. As principais interfaces oferecidas pelo LighTS são `ITupleSpace`, `ITuple` e `IField`, representadas a seguir.

```
public interface ITupleSpace {
  String getName ();
  void out ( ITuple tuple );
  void outg ( ITuple [] tuples );
  ITuple in ( ITuple template );
  ITuple inp ( ITuple template );
  ITuple [] ing ( ITuple template );
  ITuple rd ( ITuple template );
  ITuple rdp ( ITuple template );
  ITuple [] rdg ( ITuple template );
  int count ( ITuple template );
}

public interface ITuple {
  ITuple add ( IField field );
  ITuple set ( IField field , int index );
  IField get ( int index );
  ITuple insertAt ( IField field , int index );
  ITuple removeAt ( int index );
  IField [] getFields ();
  int length ();
  boolean matches ( ITuple tuple );
}

public interface IField {
  Class getType ();
  IField setType ( Class classObj );
  boolean matches ( IField field );
}
```

3.5 Considerações

Este capítulo apresentou os conceitos fundamentais do paradigma de espaço de tuplas.

O modelo de sistema distribuído baseado em espaço de tuplas surgiu como uma alternativa aos três principais modelos existentes na época, variáveis compartilhadas (*shared memory*), passagem de mensagem (*message passing*) e operações remotas (*remote operations*). A principal diferença é o desacoplamento entre processos, obtido em função da natureza de comunicação indireta, ao contrário dos modelos tradicionais, onde a comunicação entre processos ocorre de maneira direta.

Existem várias pesquisas acadêmicas propondo modelos de implementação de espaço de tuplas. Conforme será apresentado mais adiante, para este trabalho foi adotada a abordagem de espaço de tuplas sobre redes P2P, mantendo a semântica da linguagem Linda.

4 TRABALHOS RELACIONADOS

4.1 Decentralização da Arquitetura IMS

(CAO et al., 2011) propõe um algoritmo paralelo para implementação da operação de estabelecimento de sessão, mais especificamente, para implementação do S-CSCF. O algoritmo está baseado na separação do processamento de requisições de estabelecimento de sessão entre dispositivos localizados em um mesmo domínio de rede, consequentemente processadas pelo mesmo S-CSCF, e dispositivos localizados em domínios de rede diferentes.

Adicionalmente, o algoritmo proposto considera a utilização de cache de informações do S-CSCF dos domínios externos. Conforme discutido pelos autores, a existência de cache de informações reduz o acesso ao I-CSCF, pois elimina alguns passos do fluxo de troca de mensagens, reduzindo também o tempo gasto para estabelecimento de sessão.

(BAILLY; DEBEAU; JESTIN, 2006) propõe uma abordagem baseada em P2P para o protocolo SIP, visando a decentralização dos servidores para implementação de redes IMS. Sua ideia básica é distribuir os elementos funcionais, tais como o HSS, o CSCF e os servidores de aplicação.

Adicionalmente, o trabalho discute uma abordagem híbrida, onde existem alguns elementos na rede P2P denominados super-peers que formam a base da rede; os demais servidores são conectados à rede P2P através dos super-peers.

4.2 Peer to Peer SIP (P2PSIP)

Em (BRYAN; LOWEKAMP, 2007) e (BRYAN et al., 2011) é apresentado o P2PSIP, uma iniciativa do IETF para especificar uma nova versão do protocolo SIP baseado em

redes P2P.

O protocolo SIP conforme foi especificado, é um protocolo que segue o modelo requisição/resposta, com características centralizadas, principalmente pela existência do servidor de registro.

Já o P2PSIP é uma coleção de nós organizados em uma rede P2P. A base do funcionamento do P2PSIP é a especificação do protocolo RELOAD (REsource LOcation And Discovery), (JENNINGS et al., 2011), o qual é responsável pela implementação de um serviço de localização distribuído, eliminando a necessidade de um servidor de registro centralizado.

A localização dos recursos no RELOAD é feita através de implementações de DHT. Cada nó da rede P2P executa um algoritmo distribuído que possibilita o armanzenamento de dados em mais de um nó, além de prover busca eficiente na localização de recursos. No caso da busca eficiente, o P2PSIP assume que sendo utilizado qualquer implementação de DHT,a busca será feita em $log(n)$, onde n representa o número de nós, conforme demonstrado em (STOICA et al., 2003).

A utilização propsota pelo P2PSIP envolve duas funções básicas:

- *Registro*: UAs podem utilizar a funcionalidade de persistência de dados do RELOAD para armazenar o mapeamento entre seu AoR e seu identificador de nó da rede, e obter os identificadores de nós de outros UAs.
- *Rendezvous*[1]: Uma vez que um determinado UA descobre o identificador de nó do dispositivo ao qual deseja se conectar, este usará o sistema de roteamento de mensagem do RELOAD para iniciar uma conexão direta com o UA destino para troca de mensagens SIP.

Em (JENNINGS et al., 2011) é apresentado um exemplo simples. Supondo que Bob faça o registro na rede com o Identificador de Nó [1234] mapeado para sua AoR [bob@dht.example.com]. Quando Alice desejar se comunicar com Bob, utilizará o RELOAD para realizar uma busca na rede pelo AoR de Bob, [bob@dht.example.com]. Nesse momento, o RELOAD retornará para Alice o Identificador de Nó do Bob, [1234]. De posse do Identificador de Nó do Bob, Alice pode iniciar a troca de mensagens SIP padrão (INVITE, CANCEL, etc.) através de uma conexão direta.

[1] Optei pelo termo original, sem tradução

4.3 Espaço de Tuplas

4.3.1 Execution time prediction in DSM-based mobile grids

Em (BALLETTE; LIOTTA; RAMZY, 2005), foi proposta uma abordagem baseada em espaço de tuplas para execução de tarefas em grades computacionais móveis. Nesse contexto, o espaço de tuplas atua como um repositório para código executável e dados das tarefas a serem processadas, para informações de utilização de recursos e para os resultados dos processamentos.

A vantagem da utilização de espaço de tuplas neste caso, conforme descrito em (BALLETTE; LIOTTA; RAMZY, 2005), é que o modelo de comunicação desacoplada fornece meios para tratar indisponibilidade temporária de recursos, ao passo que nos modelos tradicionais a necessidade de comunicação permanente entre dispositivos não é uma característica aceitável com computação móvel.

A proposta desse trabalho foi implementada utilizando o protocolo Jini™, (WALDO, 2000), para o serviço de descoberta, e o JavaSpaces™, (FREEMAN; HUPFER; ARNOLD, 1999), como implementação de espaço de tuplas.

4.3.2 Comet: A Scalable Coordination Space for Decentralized Distributed Environments

O projeto Comet, (LI; PARASHAR, 2005), implementa um espaço de tuplas baseado em conteúdo para ambientes P2P de grande escala. Sua arquitetura provê um espaço de memória global compartilhado onde tuplas podem ser acessadas por todos os nós da rede, independente da localização física.

O modelo é construído sobre uma rede P2P utilizando uma implementação de DHT. O DHT é tradicionalmente conhecido em redes P2P por suportar entrada e saída de nós na rede dinamicamente, com excelente desempenho.

O detalhe interessante do Comet é que a busca de tuplas no espaço pode ser feita por aproximação. Em sistemas P2P baseados em DHT, a localização de um elemtno ocorre baseado em um hash gerado para uma determinada informação. O Comet amplia esse modelo utilizando uma técnica chamada *Hilbert Space-Filling Curve* para mapear as tuplas utilizando um espaço semântico de informação. Nesse caso, ao invés de uma tupla estar associada a uma única chave, como no modelo tradicional, pode ser

associada a k chaves, selecionadas a partir do seu conteúdo.

4.4 Modelagem Formal de redes IMS/SIP com CPN

4.4.1 Análise funcional do protocolo SIP

No trabalho (BAI; YE; MA, 2011) foi criado um modelo CPN para a especificação *Invite Transaction*, considerando canais de comunicação. A principal característica do trabalho é a validação funcional do protocolo SIP utilizando análise de espaço de estado. Para a modelagem foram adotados quatro cenários: (i) a transação INVITE é transmitida sobre um canal confiável; (ii) a transação INVITE é transmitida sobre um canal não confiável, e apenas o cliente faz retransmissões de mensagem; (iii) a transação INVITE é transmitida sobre um canal não confiável, e apenas o servidor faz retransmissões; e (iv) a transação INVITE é transmitida sobre um canal não confiável e ambos, cliente e servidor, realizam retransmissões de mensagens.

Em (BARAKOVIC; JEVTIC; HUSIC, 2012) o protocolo SIP também foi analisado através de um modelo CPN para a especificação *Invite Transaction*, porém, considerando as recentes modificações na máquina de estado, conforme especificado na RFC 6026 (SPARKS; ZOURZOUVILLYS, 2010). O objetivo do trabalho foi assegurar a corretude nas modificações, através da análise de espaço de estado.

(KIZMAZ; KIRCI, 2011) utilizou um modelo CPN temporizado (Timed CPNs) para análise do protocolo, também pela criação de modelo CPN para a *Invite Transaction*.

4.5 Considerações

Este capítulo apresentou os principais trabalhos relacionados à proposta desta tese. Em (BAILLY; DEBEAU; JESTIN, 2006) e (BRYAN et al., 2011), ficou clara a necessidade de descentralizar o protocolo SIP e os elementos da arquitetura IMS. Ambos utilizam uma abordagem baseada em redes P2P. Em (BALLETTE; LIOTTA; RAMZY, 2005) foi proposta uma abordagem baseada em espaço de tuplas para implementação de grade computacional móvel. A justificativa pelo espaço de tuplas é justamente sua natureza de comunicação indireta, que favorece cenários onde existe desconexão temporária dos dispositivos. (LI; PARASHAR, 2005) apresenta uma abordagem de construção de espaço de tuplas sobre redes P2P. Os demais trabalhos utilizaram redes de petri coloridas para

modelagem e validação funcional do protocolo SIP.

É importante observar que todos eles apresentam algumas características que serviram de base para este trabalho, no entanto, nenhum propõe a utilização de espaço de tuplas para implementação de protocolos de comunicação ou para a definição de arquitetura de redes IMS.

5 UMA ARQUITETURA BASEADA EM ESPAÇO DE TUPLAS PARA REDES IMS

5.1 Definição da Arquitetura

A definição da arquitetura pode ser separada em duas etapas: (i) a reestruturação das camadas do protocolo SIP, tradicionalmente baseado no modelo requisição/resposta, para um modelo baseado em espaço de tuplas (padrão *publish/subscribe*); (ii) a definição de uma infraestrutura distribuída também com base no paradigma de espaço de tuplas, construído sobre uma rede P2P, para implementação dos elementos responsáveis pelo controle de sessão da arquitetura IMS (P-CSCF, I-CSCF, S-CSCF e HSS). A Figura 5 apresenta uma visão geral da arquitetura, e seus elementos lógicos (componentes de software) e físicos (infraestrutura), (RONDINI; BRESSAN, 2012).

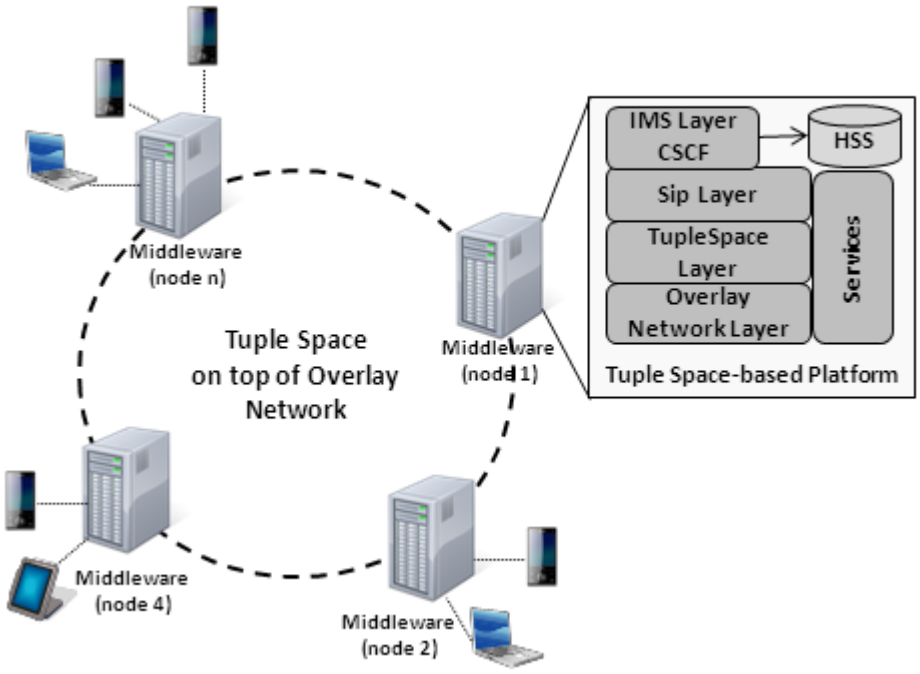

Figura 5: Visão geral da arquitetura baseada em espaço de tuplas.

Sob o aspecto físico, é possível observar a existência de servidores conectados a uma rede P2P, responsáveis por hospedar a plataforma baseada em espaço de tuplas (*Tuple Space-base Platform*), e dispositivos cliente (e.g celulares, tablets, computadores pessoais, entre outros) conectados aos servidores.

Sob o aspecto lógico, é possível observar uma arquitetura em camadas (representada no elemento *node1*), conforme descrito a seguir:

- *Camada da Rede P2P*. A arquitetura define como base uma camada de rede P2P estruturada seguindo o modelo baseado em DHT. Esta camada é responsável por construir a rede P2P e coordenar a comunicação entre os Servidores IMS da rede.

- *Camada do Espaço de Tuplas*. A camada de espaço de tuplas considera a construção de um espaço de tuplas local (LTS) executado em cada um dos servidores, e um espaço de tuplas distribuído (DTS) construído sobre a rede P2P. Ambos seguindo a semântica da linguagem Linda.

- *Camada SIP*. A camada SIP representa a implementação do protocolo SIP utilizando a abordagem baseada em espaço de tuplas. Apesar da mudança de paradigma, a arquitetura prevê a manutenção da semântica do protocolo, conforme será detalhado mais à frente.

- *Camada IMS*. A camada IMS é responsável pela implementação do componente CSCF e seus elementos. A principal característica da arquitetura é que cada servidor IMS pode atuar como P-CSCF, S-CSCF, ou I-CSCF, dependendo do tipo de tupla que recebe.

Adicionalmente, a arquitetura estabelece que todo Servidor IMS terá uma base de dados HSS local, ou seja, distribuir as consultas ao HSS é uma característica natural da arquitetura.

Completando a arquitetura, foi definido o componente *Services* que pode ser utilizado em qualquer umas das camadas. Esse componente possibilita a implementação de serviços adicionais tais como negociação de QoS, tarifação em tempo real, entre outros.

As principais diferenças em relação à arquitetura IMS padrão são:

1. A comunicação entre os elementos passa a ser feita através do espaço de tuplas, ou seja, toda operação SIP entre os elementos (dispositivos cliente e servidores) terá ao menos duas tuplas, uma de requisição e uma de resposta.

2. Cada servidor na rede pode atuar como P-CSCF, S-CSCF ou I-CSCF, dependendo do tipo de requisição que receber. Com isso, teremos um melhor aproveitamento dos recursos computacionais.

3. Distribuição do HSS, uma vez que cada servidor na rede possui um HSS local.

5.1.1 SIPSpace: SIP baseado em espaço de tuplas

O SIPSpace é o componente responsável pela implementação do protocolo através da abordagem baseada em espaço de tuplas. A Figura 6 ilustra sua arquitetura.

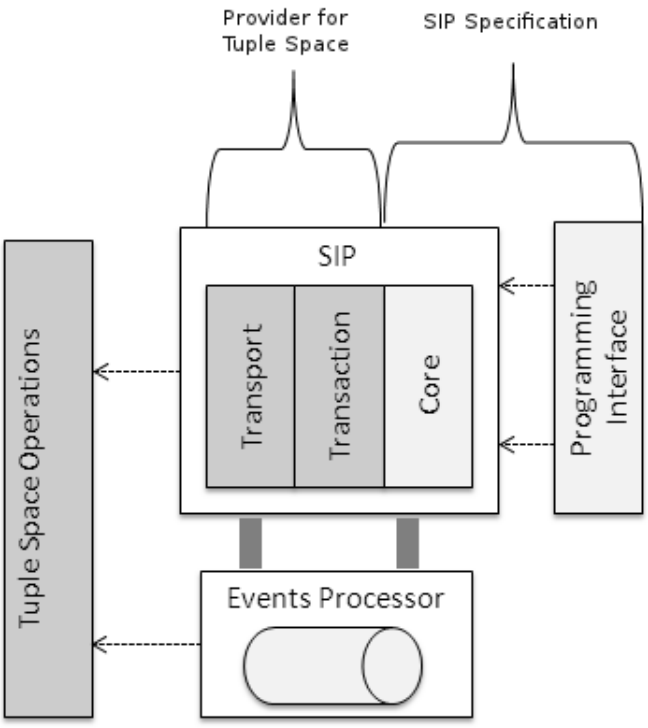

Figura 6: Visão geral da Arquitetura do SIPSpace

De acordo com a especificação, (ROSENBERG et al., 2002), a arquitetura do protocolo SIP é separada em três camadas: *core*, responsável pela definição da sintaxe e formatos de mensagens; *transaction*, responsável pelas máquinas de estado do protocolo (e.g *Client Transaction* e *Server Transaction*); e *transport*, responsável pelo

envio e recebimento de mensagens. Na arquitetura do SIPSpace, a camada *transport* é alterada para fazer o acesso ao espaço de tuplas, mantendo a semântica do protocolo.

Podemos considerar dois pontos importantes na arquitetura do SIPSpace, com relação ao suporte aos requisitos de larga escala:

1. *Registro e Serviço de localização*. A arquitetura baseada em espaço de tupla possibilita total distribuição do servidor de registro do protocolo SIP. Em primeira instância, as informações de registro são tuplas armazenadas no DTS (em memória ou persistente). Para a localização de um determinado dispositivo registrado, basta ao serviço de localização executar uma operação de leitura no espaço de tupla. Como a arquitetura considera a implementação do DTS sobre uma rede P2P baseada em DHT, uma busca pode ser feita em $O(logN)$ passos, onde N representa o número de servidores na rede, conforme demonstrado por (STOICA et al., 2003).

2. *Manutenção de sessão*. Da mesma forma que o registro de dispositivos, uma sessão será tratada como uma tupla. Dessa forma, qualquer servidor disponível na rede poderá ter acesso às informações da sessão, podendo alterá-la a qualquer momento a partir da ocorrência de algum evento (e.g acesso a novo serviço, transmissão de uma imagem, etc.).

5.1.2 Espaço de tuplas sobre redes P2P

Para o espaço de tuplas, a arquitetura apresentada neste trabalho indica um modelo baseado em redes P2P estruturadas, utilizando DHT como estratégia para organização dos Servidores IMS. A Figura 7 ilustra a topologia do espaço de tuplas sobre redes P2P.

Como pode-se observar, existe três tipos de elementos: (i) Servidores IMS (*IMS Servers*), que hospedam os elementos da arquitetura; (ii) Servidores de Aplicação (*Application Servers*), que hospedam serviços específicos disponíveis aos usuários; e (iii), os agentes (*SIP User Agents*), que são os dispositivos móveis propriamente ditos.

A arquitetura utiliza o conceito de *super-peers*, representados pelos servidores IMS, ou seja, a rede P2P é formada apenas pelos servidores IMS (*super-peers*). Os outros elementos, servidores de aplicação e dispositivos se comunicam diretamente com um determinado *super-peer*.

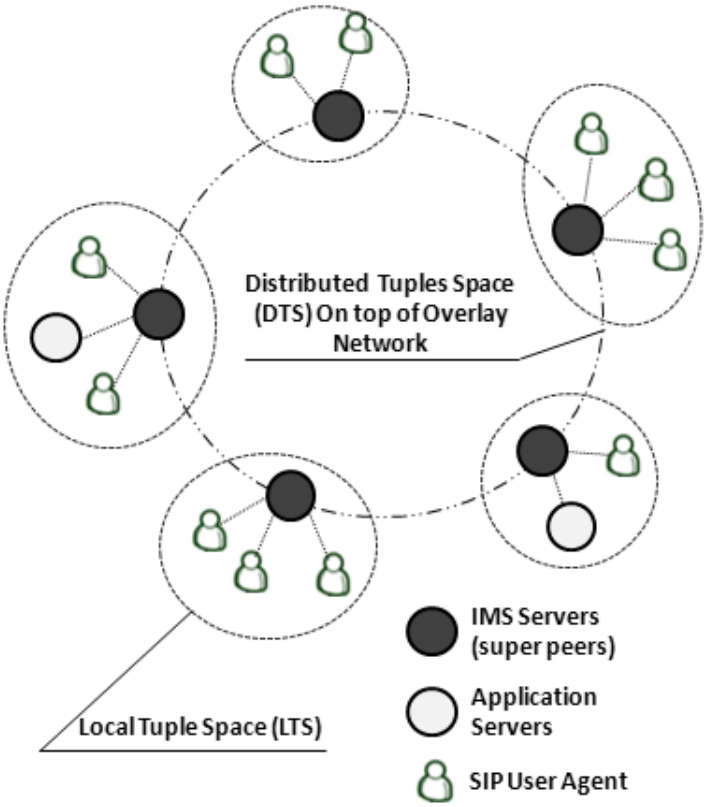

Figura 7: Topologia da arquitetura baseada em espaço de tuplas

As tuplas foram classificadas em três tipos: tupla de sessão, tupla de operação SIP e tupla de registro, cada uma com características e comportamentos específicos em relação à execução local (LTS) ou distribuída (DTS).

- *Tuplas de Operação*. Tuplas de operação representam as operações relacionadas ao estabelecimento de sessão do protocolo SIP, e.g INVITE, CANCEL, BYE, PRACK e ACK. Esse tipo de tupla é enviada pelo dispositivo cliente e publicada no LTS. Em caso de sobrecarga do servidor IMS de origem, o componente Event Processor irá publicá-la no DTS, para que qualquer outro servidor IMS com carga menor possa realizar o processamento.

- *Tuplas de Registro*. Tuplas de registro correspondem especificamente à operação REGISTER do protocolo SIP. Assim como as tuplas de operação, as tuplas de registro serão publicadas no LTS pelos dispositivos cliente. Ao ser concluído o processo de registro de dispositivo na rede, a tupla de registro é publicada no DTS para possibilitar eficiência na localização de dispositivos.

- *Tuplas de Sessão*. As Tuplas de sessão são criadas no DTS após a confirmação do estabelecimento de uma sessão por parte dos dispositivos cliente envolvidos, ou seja, uma tupla de sessão representa a própria sessão multimídia em curso. A tupla de sessão deve ficar no DTS para possibilitar eficiência na localização e manutenção, pois poderá ser alterada com frequência.

Considerando que o espaço de tuplas distribuído é formado pelo compartilhamento de memória de todos os servidores IMS conectados à rede P2P, é necessário um mecanismo de replicação que garanta a disponibilidade da tupla enquanto estiver ativa e que seja facilmente acessada a partir de qualquer um dos Servidores IMS.

Nesse sentido, foi definido o algoritmo a seguir para replicar tuplas no DTS visando manter o mínimo de réplicas necessárias.

Algoritmo 1 $ReplicarTupla(S)$

1: **while** $rc < n$ **do**
2: **if** $PF > p$ **and** $T.TTL > t$ **then**
3: replica tupla S no DTS seguindo o modelo da rede P2P em função do algoritmo DHT utilizado. {Importante observar que a réplica da tupla nasce com o $T.TTL$ da tupla principal, ou seja, o $T.TTL$ é o mesmo para toda réplica.}
4: incrementa rc
5: incrementa t
6: **end if**
7: **end while**

{onde: PF é a probabilidade de falha do servidor, p é um parâmetro do sistema que indica a probabilidade máxima aceita, $T.TTL$ é o tempo de vida da tupla, t é o tempo máximo para que uma tupla seja replicada, rc é o número de réplicas da tupla e n é o número máximo de réplicas permitidas no sistema.}

A replicação de tuplas irá ocorrer em função da probabilidade de falha do servidor e do tempo de vida da tupla, ou seja, decorridos um tempo de vida $T.TTL$, a tupla será replicada em um determinado servidor. A cada intervalo de tempo de vida $T.TTL$, a tupla será replicada novamente, até atingir um número máximo de réplicas.

Cada Servidor IMS tem sua probabilidade de falha, que pode, por exemplo, ser definida em função da carga[1]. O tempo t varia para cada réplica, ou seja, o t aumenta após cada replicação, fazendo com que o intervalo de replicação de uma única tupla aumente ao longo do tempo, evitando replicações excessivas.

[1]O estudo de probabilidades de falha de servidores, bem como a definição de uma função de probabilidade de falha para o algoritmo de replicação, ficou fora do escopo deste trabalho.

5.1.3 Aspectos Operacionais

Considerando que a arquitetura é baseada em espaço de tuplas, o mecanismo básico de operação da rede é a publicação e leitura de tuplas (operações `out(..)` e `in(..)`). O componente *Events Processor*, parte do SIPSpace conforme mostra a Figura 6, é o responsável pela leitura de tuplas tanto do LTS quanto do DTS. A leitura é feita em paralelo, desacoplada dos elementos funcionais da arquitetura IMS. A Figura 8 ilustra de forma simplificada o modo de operação do servidor IMS.

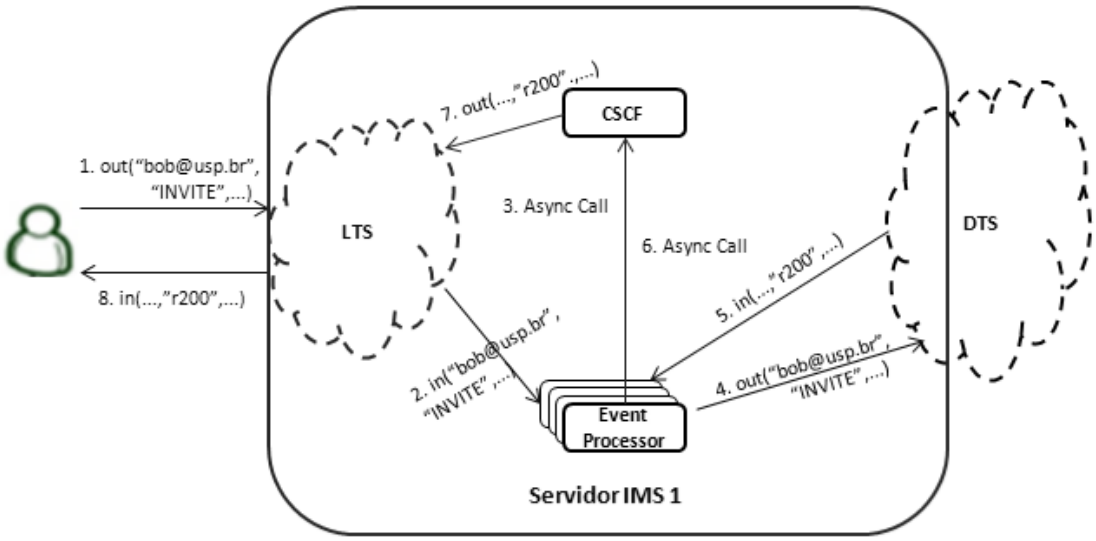

Figura 8: Modo de operação do servidor IMS (simplificado)

1. Para iniciar uma sessão ou se registrar na rede, o dispositivo publica uma tupla no LTS do servidor IMS ao qual estiver conectado.

2. O *Event Processor* lê a tupla do LTS.

3. O *Event Processor* faz uma chamada assíncrona ao CSCF para realizar o processamento da requisição.

4. Caso o CSCF esteja sobrecarregado, o *Event Processor* irá publicar a tupla no DTS para que seja lida e processada por qualquer outro servidor IMS da rede, ao invés de fazer a chamada assíncrona ao CSCF.

5. Se o *Event Processor* publicou a requisição no DTS, a resposta também será publicada no DTS pelo servidor IMS que realizou o processamento. Neste caso, o *Event Processor* faz a leitura da resposta publicada no DTS.

6. Se o *Event Processor* recebeu a resposta pelo DTS, este também fará uma chamada assíncrona para o CSCF finalizar o processamento.

7. Ao final do processamento, o CSCF local publica uma tupla de resposta no LTS.

8. Por fim, o dispositivo faz a leitura da tupla de resposta do LTS. A sequência de mensagens continua da mesma forma de acordo com o protocolo SIP.

Cada servidor IMS possui as mesmas características em termos de componentes de software e conectividade na rede. Essa característica permite que qualquer Servidor IMS da rede seja capaz de processar qualquer tipo de requisição.

A característica do servidor IMS aliada ao modelo de rede P2P adotado, possibilita a inclusão de novos servidores IMS na rede sem grandes esforços.

Um novo Servidor IMS na rede, ao ser iniciado, irá se conectar à rede P2P e o DHT se encarrega de reorganizar a estrutura. Após ser adicionado à rede, o Servidor IMS está apto a processar tuplas que estejam publicadas no DTS e receber requisições de dispositivos cliente.

5.1.4 Aspectos de Implementação

Para ilustrar uma abordagem de implementação da arquitetura, tomamos como base a especificação JAIN-SIP, (O'DOHERTY; RANGANATHAN, 2003), definida pelo *Java™Community Process* através da JSR-32, e o *framework* LighTS, (PICCO; BALZAROTTI; COSTA, 2005), para construção de espaços de tupla.

O JAIN-SIP é uma especificação aberta que serve como base para implementação do protocolo SIP utilizando tecnologia Java™. A Figura 9 mostra a arquitetura da API JAIN-SIP.

Implementações de JAIN-SIP são denominadas *providers*. Considerando a arquitetura apresentada neste trabalho, podemos dizer que o SIPSpace é um *provider* para o JAIN-SIP que utiliza um modelo de comunicação baseado em espaço de tuplas.

Os fragmentos de código a seguir ilustram o SIPSpace implementando as interfaces *stack* e *provider* do JAIN-SIP.

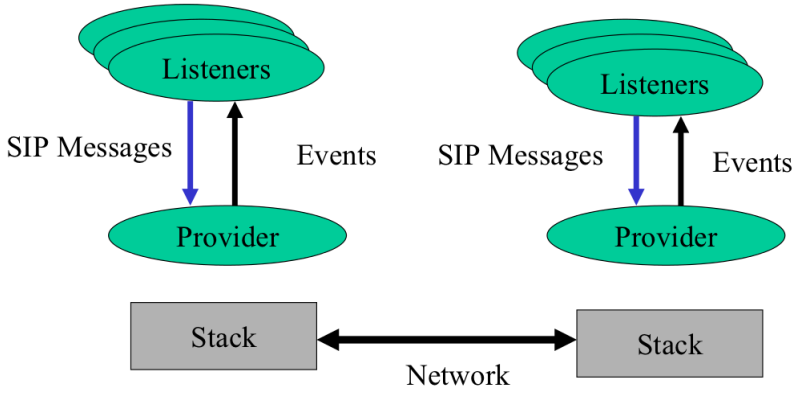

Figura 9: Arquitetura JAIN-SIP (O'DOHERTY; RANGANATHAN, 2003)

```
package br.usp.pcs.larc.sipspace.jainsip.javax.sip;
public class SipStackImpl implements javax.sip.SipStack{
        private Properties stackConfig = null;
        public SipStackImpl(Properties p) {
                this.stackConfig = p;
        }
        ...
        public SipProvider createSipProvider(ListeningPoint lp)
                        throws ObjectInUseException {
                return new SipProviderImpl();
        }
}

package br.usp.pcs.larc.sipspace.jainsip.javax.sip;
public class SipProviderImpl implements javax.sip.SipProvider{

}
```

Conforme mencionado anteriormente, o LighTS é um *framework* para implementação de espaço de tuplas que segue a semântica da linguagem Linda. Assim como o JAIN-SIP oferece interfaces para implementação do protocolo SIP, o LighTS oferece interfaces para implementação de espaço de tuplas. A interface `ITupleSpace` contém as operações básicas definidas pela linguagem Linda, tais como, publicação de tuplas (`out`) e as operações bloqueantes de consulta (`rd` e `in`). As interfaces `IField` e `ITuple` são utilizadas para descrever uma tupla.

Adicionalmente, para reduzir o tráfego na rede é possível utilizar o formato JSON para publicação e leitura de tuplas. JSON é um formato leve para troca de mensagens. Em testes simples, a mesma informação representada como objeto Java™quando serializada utilizando `java.io.ObjectOutputStream` tem um tamanho duas vezes maior que serializando no formato JSON. O mesmo acontece quando comparada a serialização JSON com o formato XML, sendo o XML duas vezes maior que o formato JSON.

O LighTS pode ser estendido para comportar a utilização de JSON para representar o conteúdo de um tupla. Isso é possível pela criação de interfaces que estendem as interfaces padrão do LighTS, tais como `FieldMessage` e `ITupleJSON`, conforme fragmentos de código apresentados a seguir.

```
public interface FieldMessage {
  public void setMessage(byte[] m);
  public byte[] getMessage();
  public Object getParsedMessage();
}

public interface ITupleJSON extends ITuple{
  public void addFieldMessage (FieldMessage fm);
}
```

E então implementar o espaço de tuplas (interfaces do LighTS estendidas) sobre a rede P2P, conforme mostrado na Figura 10 e nos fragmentos de código a seguir.

```
package br.usp.pcs.larc.p2pspace;
public class MessageImpl implements FieldMessage{
}

package br.usp.pcs.larc.p2pspace;
public class FieldImpl implements IField{
}

package br.usp.pcs.larc.p2pspace;
public class TupleImpl implements ITupleJSON{
}
```

```
package br.usp.pcs.larc.p2pspace;
public class P2PTupleSpace implements ITupleSpace {
}
```

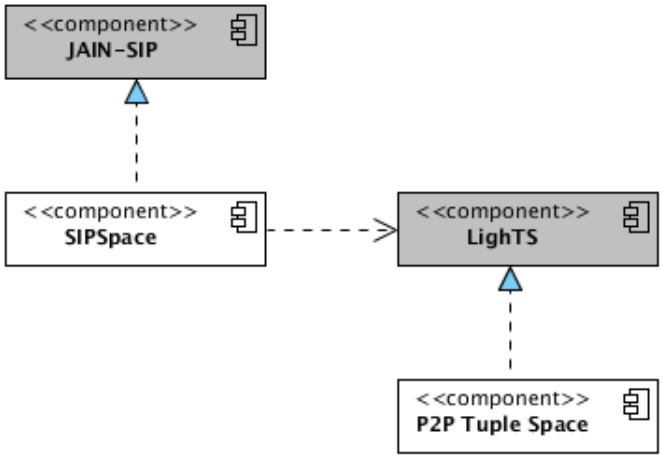

Figura 10: Arquitetura de Implementação

5.2 Modelagem Formal

Conforme já mencionado nos objetivos deste trabalho (seção 1.3), a validação da arquitetura baseada em espaço de tuplas será feita através de modelagem formal com redes de petri coloridas. Para auxiliar o leitor em uma melhor compreensão dos modelos, uma introdução sobre a notação CPN e o funcionamento do simulador foi incluída no Apêndice A.

5.2.1 Modelo CPN para o SIPSpace

Para validar o SIPSpace do ponto de vista funcional, foi utilizado um modelo CPN não-hierárquico temporizado (*Non-Hierarchical Timed CPN*). Conforme estudado em (BAI; YE; MA, 2011), (BARAKOVIC; JEVTIC; HUSIC, 2012), e (KIZMAZ; KIRCI, 2011), o modelo para análise funcional do SIPSpace considera a transação SIP INVITE, seguindo as especificações (ROSENBERG et al., 2002) e (SPARKS; ZOURZOUVILLYS, 2010).

A transação INVITE considera duas máquinas de estado, transação cliente (*Client Transaction*) e transação servidor (*Server Transaction*):

Transação Cliente: A transação cliente é criada pelo dispositivo quando uma requisição INVITE é criada. Nesse momento, a transação cliente pode alcançar cinco estados, *calling*, *proceeding*, *accepted*, *completed*, e *terminated*. Adicionalmente, quatro controladores de tempo podem ser disparados: *timer A*, utilizado para retransmissão de mensagem quando utilizado protocolo de transporte não-confiável; *timer B*, para controlar o *time-out* da transação; *timer D*, responsável pelo *time-out* de espera de uma resposta 3xx; e *timer M*, responsável pelo *time-out* de espera de uma resposta 2xx.

Quando criada, a transação cliente assume o estado inicial *calling*; a requisição é enviada para a camada de transporte do protocolo SIP para que seja enviada ao servidor. Nesse momento o *Timer B* é iniciado, o qual é responsável pelo controle de *time-out* da transação. O *Timer A* será iniciado apenas quando utilizar um protocolo de transporte não-confiável. Mais à frente veremos que o *timer A* é desnecessário para o SIPSpace.

Estando no estado inicial *calling*, a transação cliente pode causar a ocorrência de alguns eventos:

1. O *Timer A* pode disparar, forçando o reinício do tempo e a retransmissão da requisição;
2. O *Timer B* pode disparar, mudando a transação para o estado *terminated*, ou seja, ocorreu um *time-out*;
3. O transporte da mensagem falhar, mudando a transação para o estado *terminated*.
4. Receber uma resposta (1xx) indicando que um recurso adicional deve ser provisionado, mudando o estado da transação para *proceeding*.
5. Receber uma resposta (2xx) indicando que a requisição foi aceita pelo servidor, mudando o estado da transação para *accepted*.
6. Receber uma resposta (3xx) indicando que a requisição foi recebida pelo servidor mas não foi aceita (precisa de uma confirmação do cliente). A transação cliente cria uma nova requisição ACK e passa para o estado *completed*.

Quando a transação cliente entra no estado *accepted*, o *Timer M* deve ser iniciado. Esse *timer* reflete o tempo máximo de espera para receber uma resposta

2xx. Quando o *Timer M* dispara, a transação cliente passa para o estado *Terminated*.

Quando a transação cliente entra no estado *completed*, o *Timer D* deve ser iniciado. Esse *timer* reflete o tempo de espera por uma resposta 3xx. Antes do *Timer D* disparar, se uma resposta 3xx for recebida, a transação cliente cria e envia um ACK para o servidor, e permanece no estado *completed*. Adicionalmente, caso ocorra uma falha no envio do ACK, a transação cliente passa para o estado *terminated*.

Finalmente, após entrar no estado *terminated*, a transação cliente será concluída.

A especificação da transação cliente define os seguintes valores para os *timers*:

$TimerA = T1$ onde $T1 = 500ms$

$TimerB = 64 * T1$

$TimerD = 0$ transporte confiável

$TimerM = 0$ para transporte confiável

Transação Servidor: A transação servidor é criada quando o servidor recebe uma requisição INVITE. O servidor gera uma resposta 1xx (*trying*) e devolve para o cliente. Essa resposta provisória é necessária para evitar que o cliente retransmita a requisição rapidamente.

A transação servidor pode atingir cinco diferentes estados, *proceeding*, *accepted*, *completed*, *confirmed*, e *terminated*, e quatro indicadores de tempo: *timer G*, utilizado em protocolos de transportes não-confiáveis para controlar retransmissões de resposta; *timer H*, para indicar quando a transação servidor aborta o envio de resposta; *timer I*, para indicar a finalização da transação; e *timer L* utilizado para controlar o *time-out* de retransmissão de resposta r2xx..

A transação servidor inicia no estado *proceeding*. A partir desse momento podem ocorrer alguns eventos:

1. Disparar respostas provisórias (1xx) para os clientes, permanecendo no estado *proceeding*.
2. A camada de transporte falhar durante o envio de uma resposta, mantendo a transação servidor no estado *proceeding*.
3. Enviar uma resposta de sucesso (2xx), mudando o estado para *accepted*.

4. Enviar uma resposta final de insucesso (3xx), mudando e estado para *completed*.

O *Timer L* é iniciado quando a transação servidor entra no estado *accepted*. Esse *timer* reflete o tempo que a transação servidor espera para retransmitir uma resposta 2xx. Quando o *timer L* dispara, a transação entra no estado *terminated*.

Quando a transação entra no estado *completed*, o *Timer H* é iniciado. Esse *timer* determina quando o servidor abandona a retransmissão de uma resposta. Ao disparar, a transação entra no estado *terminated*. Se estiver utilizando um transporte não-confiável, o *Timer G* será iniciado para controlar o tempo entre cada retransmissão da resposta 3xx.

Enquanto estiver no estado *proceeding*, se um ACK for recebido a transação entra no estado *confirmed* e o *Timer I* é iniciado. Quando o *timer I* disparar, a transação entrara no estado *terminated* e será finalizada.

A especificação da transação servidor define os seguintes valores para os *timers*:

$TimerG = T1$ onde $T1 = 500ms$

$TimerH = 64 * T1$

$TimerI = 0$ para transporte confiável

$TimerL = 64 * T1$

A Tabela 4 apresenta as declarações utilizadas no modelo, e a Figura 11 apresenta o modelo CPN para a transação INVITE no contexto do SIPSpace, seguindo as especificações apresentadas anteriormente.

Conforme observado na Tabela 4, o modelo define o seguinte conjunto de *Cores*.

- CLIENT_STATES representa os estados da transação cliente, *calling*, *proceeding*, *accepted*, e *terminated*.
- SERVER_STATES representa os estados da transação servidor, *proceeding*, *completed*, *accepted*, *confirmed* and *terminated*. Para evitar duplicidade com os estados definidos na *Cor* CLIENT_STATES, foi acrescentado o sufixo "*S*"nos estados da transação servidor. Para propósitos de modelagem, foi adicionado o estado inicial *idle*.

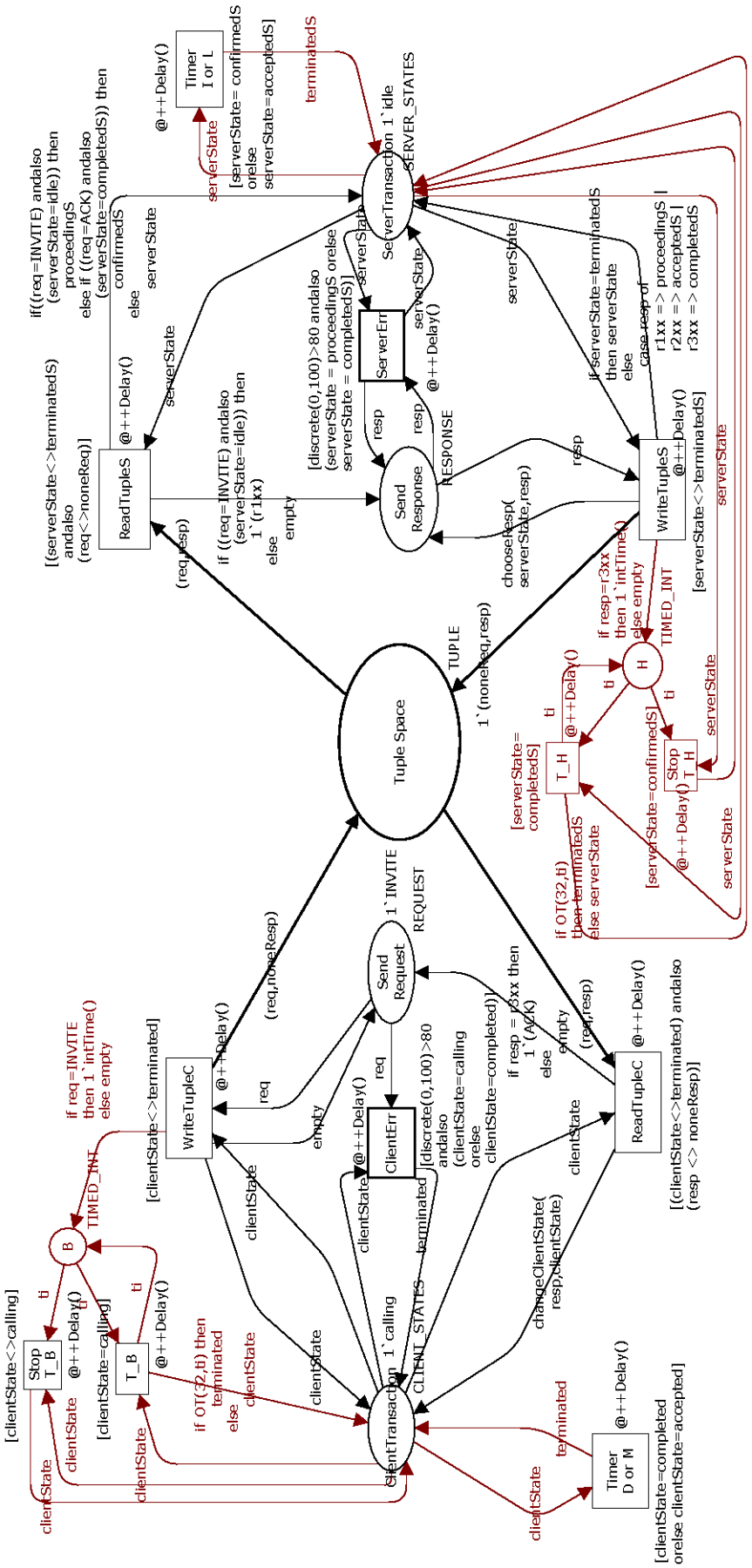

Figura 11: Modelo CPN para Transação INVITE - Validação Funcional

Tabela 4: Declarações para o modelo CPN da Figura 11

`globref sendRespControl = 0;`
`colset CLIENT_STATES = with calling | proceeding | completed | accepted | terminated;`
`colset SERVER_STATES = with idle | proceedingS | completedS | acceptedS | confirmedS | terminatedS;`
`colset REQUEST = with INVITE | ACK | noneReq timed;`
`colset RESPONSE = with r1xx | r2xx | r3xx| noneResp timed;`
`colset TUPLE = product REQUEST * RESPONSE timed;`
`var ti : TIMED_IND;`
`var clientState : CLIENT_STATES;`
`var serverState : SERVER_STATES;`
`var req : REQUEST;`
`var resp : RESPONSE;`

- REQUEST representa os métodos INVITE e ACK. Adicionalmente, para propósitos de modelagem foi definido o método *noneReq* para indicar ao modelo quando uma tupla é de requisição ou não.

- RESPONSE representa as classes de resposta do protocolo SIP, ou seja, r1xx, r2xx, r3xx. Para propósitos de modelagem, foi criado uma resposta *noneResp* para indicar ao modelo quando uma tupla é de resposta ou não.

- TUPLE representa os dados publicados no espaço de tuplas. No modelo, uma tupla recebe valores dos tipos REQUEST e RESPONSE. Quando o primeiro valor é um *noneReq*, significa que é uma tupla de resposta. Quando o segundo valor é *noneResp*, significa que é uma tupla de requisição.

Para adicionar expressividade ao modelo visando alcançar a correta transição de estado conforme especificação da transação SIP INVITE, foram definidas algumas funções CPN-ML. Os códigos CPN-ML a seguir ilustram o funcionamento de cada uma das funções.

Delay(): A função `Delay` é utilizada para simular o tempo gasto para transmitir mensagens pela rede. Para efeitos de modelo, utilizamos uma distribuição exponencial para gerar intervalos de tempo de 5ms.

```
1. fun Delay() =
2.   exponential(0.2);
```

OT(): A função OT (i.e. *O*ver *T*ime) é utilizada para verificar a ocorrência de time out na execução da transação INVITE. A função recebe um parâmetro *timeout* e um parâmetro *timecall* que representa o tempo atual da simulação, chamado pelo simulador CPN de *Model Time*, no momento em que a tupla é publicada no espaço (*Lugar Tuple Space*).

```
1. fun OT(timeout:INT,timecall:INT) =
2.  let
3.    val timepass = intTime() - timecall
4.  in
5.    if (timepass > timeout) then
6.      true
7.    else
8.      false
9.  end;
```

chooseResp(): A função chooseResp é utilizada para gerar as respostas SIP de acordo com o contexto da simulação. A função tem uma *flag* para controlar a primeira execução. Na primeira execução, a função utiliza uma distribuição discreta de probabilidade para selecionar a resposta entre r2xx e r3xx. Na próxima execução, a escolha será oposta à primeira, ou seja, se a primeira execução retornou r2xx, a segunda execução irá retornar r3xx.

```
1. fun chooseResp(ss:SERVER_STATES,lastResp:RESPONSE) =
2.  let
3.    val i = discrete(0,2)
4.    fun increment() = (inc sendRespControl)
5.  in
6.    increment();
7.    if !sendRespControl = 1 then
8.       if i=0 then 1‘r3xx
9.       else 1‘r2xx
10.   else
11.      if lastResp = r2xx then empty
12.      else 1‘r2xx
13. end;
```

changeClientState: A função `changeClientState` é responsável por implementar as regras de transição de estado do lado do cliente.

```
1. fun changeClientState(resp:RESPONSE,cs:CLIENT_STATES) =
2.   if resp=r1xx andalso cs=calling then
3.     proceeding
4.   else
5.     if resp=r3xx andalso cs<>accepted then
6.       completed
7.     else
8.       if resp=r2xx then
9.         accepted
9.       else
10.        cs;
```

No modelo CPN para o SIPSpace, Figura 11, pode-se observar a transação cliente do lado esquerdo e a transação servidor do lado direito.

A transação cliente é modelada com dois lugares, `ClientTransaction` e `SendRequest`, e duas transições, `WriteTupleC`, a qual representa a operação `out` e `ReadTupleC`, a qual representa a operação `in`, ambas do paradigma de programação com espaço de tuplas. A transição `ClientErr` representa a ocorrência de erros de transmissão e *timeout* (*timers* B, D, e M).

Assim como na transação cliente, a transação servidor é modelada com dois lugares, `ServerTransaction` e `SendResponse`, e duas transições, `WriteTupleS`, a qual representa a operação `out` e `ReadTupleS`, a qual representa a operação `in`. A transição `ServerErr` representa a ocorrência de erros de transmissão e *timeout* (*timers* H, I, e L).

O lugar `Tuple Space` é uma abstração do espaço de tuplas que recebe as marcações representando as tuplas, ou seja, as mensagens de requisição e resposta.

O modelo CPN para o SIPSpace desconsidera os *timers* A e G. Ambos são utilizados quando a camada de transporte é implementada sob canais não-confiáveis, porém, no contexto do SIPSpace assumiu-se que a implementação de espaço de tuplas sempre irá utilizar canais confiáveis para transporte de mensagens.

A simulação inicia com a marcação `1‘calling` no lugar `ClientTransaction`,

a marcação `1‘INVITE@0` no lugar `SendRequest` e a marcação `1‘idle` no lugar `ServerTransaction`. A simulação considera o fluxo completo de uma requisição entre um dispositivo cliente e o servidor SIP.

Conforme a especificação SIP INVITE, ao final da simulação os lugares `Client Transaction` e `Server Transaction` devem estar no estado *terminated* e *terminatedS* respectivamente.

Estando o lugar `ClientTransaction` com a marcação `1‘calling`, e o lugar `SendRequest` com a marcação `1‘INVITE`, a transição `WriteTupleC` é habilitada para início da simulação. A partir desse momento, a simulação segue a especificação definida para o cliente e o servidor.

5.2.2 Modelo CPN para a Rede IMS Baseada em Espaço de Tuplas

Para validar a arquitetura baseada em espaço de tuplas, considerando os aspectos lógicos e físicos, criou-se um conjunto de modelos CPN. Os modelos abordam os aspectos de infraestrutura baseada em P2P, o SIPSpace principalmente o comportamento do componente *Events Processor* e os espaços de tuplas local (LTS) e distribuídos (DTS).

A ferramenta CPN Tool permite desenvolver redes de petri coloridas de forma hierárquica, ou seja, é possível criar um modelo alto nível composto por módulos representando componentes do sistema. No contexto da arquitetura baseada em espaço de tuplas, foi criado um modelo alto nível representando a topologia da rede com os Domínios A e B, sendo que o Domínio A representa a rede P2P com 4 Servidores IMS conectados. As Figuras 12, 13, 14, 15, 16, 17 ilustram os detalhes do modelo.

A Tabela 5 apresenta a declaração do conjunto de "cores"definidos para o modelo CPN da arquitetura baseada em espaço de tuplas utilizado na análise de desempenho.

- P2PNODE representa os Servidores IMS que fazem parte da rede P2P. No modelo utilizado para simulação foram considerados quatro Servidores IMS.
- REQUEST representa os métodos utilizados na requisição com base na especificação SIP, e.g, REGISTER, INVITE, PRACK e ACK. Adicionalmente, para propósitos de modelagem foi definido o método *noReq* para indicar ao modelo quando uma tupla é de requisição ou não.

Tabela 5: Conjunto de cores para os modelos CPN da arquitetura baseada em espaço de tuplas

`colset P2PNODE = with node1 | node2 | node3 | node4 | domainB;`
`colset REQUEST = with REGISTER | INVITE | PRACK | ACK | noReq;`
`colset RESPONSE = with r200 | r401 | r183 | noResp;`
`colset NO = INT timed;`
`colset TOA = INT;`
`colset SOP = INT;`
`colset AUTH = BOOL;`
`colset FINISH = BOOL;`
`colset TUPLE = product NO * REQUEST * RESPONSE * TOA * SOP * P2PNODE * AUTH * FINISH timed;`

- RESPONSE representa os códigos de resposta utilizados durante a simulação, e.g, r200 para resposta de sucesso, r401 para usuário não autenticado e r183 para sessão em progresso. Para propósitos de modelagem, foi criado uma resposta *noResp* para indicar ao modelo quando uma tupla é de resposta ou não.
- NO representa números inteiros com marcação de tempo
- TOA representa a marcação de tempo na chegada de uma requisição
- SOP representa a marcação de tempo em um *Lugar* específico
- AUTH booleano que indica se o usuário foi autenticado ou não.
- FINISH boolean que indica que o processamento no CSCF foi finalizado.
- TUPLE representa os dados publicados no espaço de tuplas. No modelo, uma tupla é o produto das Cores NO, REQUEST, RESPONSE, TOA, SOP, P2PNODE, AUTH e FINISH.

A Tabela 6 apresenta as variáveis utilizadas no modelo. Variáveis CPN-ML tem a mesma característica de variáveis em qualquer linguagem de programação, porém os "tipos de dados"são as "cores".

A Tabelas 7 apresenta as declarações globais do modelo. Essas declarações são utilizadas para simplificar a configuração do modelo, pois, uma vez alteradas, mudam o resultado das simulações.

Importante destacar que as declarações globais apresentadas foram utilizadas tanto nas simulações do modelo CPN para a arquitetura baseada em espaço de tuplas quanto

Tabela 6: Declarações de variáveis para os modelos CPN da arquitetura baseada em espaço de tuplas

`var n: NO;`
`var node: P2PNODE;`
`var toa: TOA;`
`var sop: SOP;`
`var req: REQUEST;`
`var resp: RESPONSE;`
`var auth: AUTH;`
`var finish: FINISH;`
`var i: INT;`

Tabela 7: Declarações globais para os modelos CPN da arquitetura baseada em espaço de tuplas

`globref txRegister=0.6;`	Indica a taxa de requisições REGISTER utilizada na simulação. O valor 0.6 indica que 60% das requisições são do tipo REGISTER e o restante do tipo INVITE. Esse valor foi definido de acordo com estudo apresentado em (KANG et al., 2007).
`globref txInviteLocal=0.5;`	Indica o percentual de requisições do tipo INVITE que são feitas entre domínios ou no mesmo domínio. O valor 50% foi adotado por não ter um estudo a respeito que defina esse comportamento.
`globref txHSSQueryLocal=0.5;`	Indica o percentual de consultas ao HSS que serão realizadas localmente ou através da rede. Como na arquitetura padrão não existe essa característica, ou seja, todo acesso ao HSS é feito através da rede, foi utilizado o valor de 50%.
`globref leaseTime=1000;`	Indica o time-out da tupla publicada no espaço, representando o conceito de Lease do espaço de tuplas.
`globref iatNode1=0;`	Indica o intervalo de tempo entre disparos de requisição, utilizado pela função `NextArrival()`. Esse valor é alterado a cada execução da simulação.
`globref loadNode1=0;`	Indica o número de requisições que serão disparadas; equivale a dizer o número de dispositivos conectados em um Servidor IMS.Valor também alterado durante as simulações.
`globref node1Enabled=true;`	Indica se o Servidor IMS estará habilitado durante a simulação.
`globref threadsAvailable1=70;`	Indica o número de requisições simultâneas que o componente CSCF consegue processar. Para efeitos de simulação, foi adotado como sendo um servidor com capacidade de 100 requisições, 30 para o *Events Processor* e 70 para o CSCF.

para o modelo CPN da arquitetura padrão, mantendo assim o mesmo cenário para realizar as comparações.

Para adicionar expressividade ao modelo visando alcançar a correta simulação da arquitetura proposta, foram definidas algumas funções CPN ML.

RANDelay: A função `RANDelay()` é utilizada para representar o atraso na comunicação entre o dispositivo móvel e a rede IMS. O tempo considerado é de 100ms, obedecendo a uma distribuição exponencial, conforme código CPN-ML apresentado a seguir.

```
1. fun RANDelay () =
2.    exponential(0.01);
```

LTSAccessDelay: A função `LTSAccessDelay()` define o atraso durante o acesso ao LTS. A função utiliza uma distribuição exponencial de 10ms.

```
1. fun LTSAccesDelay() =
2.    exponential(0.1);
```

DTSAccessDelay: A função `DTSAccessDelay()` define o atraso no acesso ao DTS. A função gera números obedecendo uma distribuição exponencial. Para o modelo, estamos considerando uma intensidade de 0.07 para a função, o qual representa uma distribuição exponencial de intervalo de tempo de 15ms.

```
1. fun DTSAccessDelay() =
2.    exponential(0.07);
```

DomainBDelay: A função `DomainBDelay()` representa o tempo de processamento no domínio B, gerando intervalos de tempo de 160ms, através de distribuição exponencial.

```
1. fun DomainBDelay() =
2.  exponential(0.006);
```

InterDomainDelay: A função `InterDomainDelay()` é utilizada para representar o atraso na comunicação entre o Domínio A e o Domínio B. A função gera intervalos de tempo de 25ms obedecendo a uma distribuição exponencial conforme mostra o código CPN-ML apresentado a seguir.

```
1. fun InterDomainDelay()=
2.    exponential(0.04);
```

HSSQueryTime: A função `HSSQueryTime()` representa o tempo gasto para realização de consultas no HSS. Pela arquitetura baseada em espaço de tuplas, uma consulta ao HSS pode ser feita localmente ou em qualquer Servidor IMS da rede. A função utiliza uma distribuição uniforme de probabilidade para decidir se a consulta será feita local ou na rede. Para efeitos de simulação, consideramos uma taxa de 50% de consultas local e 50% de consultas remota, valor este definido pela variável global `txHSSQueryLocal`. No caso, foram adotados 2s para acesso local e 15s para acesso remoto, conforme utilizado por (LUO et al., 2009)

```
1. fun HSSQueryTime() =
2. let
3.   val hd = !txHSSQueryLocal;
4. in
5.   if (uniform(0.0,1.0) <= hd) then
6.    2
7.   else
8.    15
9. end;
```

CSCFProcessingTime: A função `CSCFProcessingTime()` representa o tempo de processamento no CSCF. A função gera intervalos de tempo de 3ms utilizando distribuição exponencial.

```
1. fun CSCFProcessingTime() =
2.  exponential(0.3);
```

InviteTargetUEdelay: A função `InviteTargetUEdelay` representa o atraso durante o envio de requisição INVITE para o UE destino. UEs destino podem estar na mesma rede ou em domínios diferentes. A variável global `txInviteLocal` simula o percentual de requisições locais e remotas. Para efeito de simulação, estamos considerando 50% de requisições locais e 50% de requisições remotas. Para requisições na rede local o atraso é de 50ms e para requisições remotas o atraso é de 160ms, ambos com distribuição exponencial.

```
fun InviteTargetUEdelay() =
let
   val invite = uniform(0.0,1.0)
in
   if (invite <= (!txInviteLocal)) then
     exponential(0.02)
   else
     exponential(0.006)
end;
```

initSim1: A função `initSim1()` é utilizada para inicializar a variável global `threadsAvailable1`, ou seja, garantir que ao iniciar a simulação o número de *threads* disponíveis para processamento de tuplas do Servidor IMS representado por *node1* é igual ao máximo definido. O mesmo vale para cada um dos Servidores IMS da rede, e.g `initSim2`, `initSim3`, etc.

```
1. fun initSim1() =
2.  let
3.    fun initThreadsAvailable () = threadsAvailable1:=70;
4.  in
5.    initThreadsAvailable();
6.    1‘1
7.  end;
```

NextArrival: A função `NextArrival()` utiliza a função de distribuição de Poisson para gerar intervalos de tempo com base no parâmetro `iat` (*inter-arrival time*). Essa função é utilizada para determinar a carga de requisições para a simulação. Como exemplo, ao passar o parâmetro `iat = 15`, estamos considerando para efeito de simulação um intervalo de de 15ms entre cada requisição, o equivalente a 240000 requisições p/hora obedecendo a uma distribuição de Poisson.

```
1. fun NextArrival iat =
2.     poisson(iat);
```

CreateTuple: A função `CreateTuple()` é responsável por gerar requisições utilizando a cor `TUPLE`. Como o modelo considera requisições de registro e de estabelecimento de sessão, essa função é responsável por gerar os dois tipos de

maneira uniforme. Para isso, utilizamos a variável global `txRegister` que é definida inicialmente com valor 0.6, conforme mostra a Tabela 7, indicando que 60% das tuplas geradas são requisições do tipo REGISTER.

```
1. fun CreateTuple(n:NO, node:P2PNODE) =
2.  let
3.    val t = time()
4.    val at = IntInf.toInt (RealToIntInf 0 t)
5.    val reqType = uniform(0.0,1.0)
6.  in
7.    if (reqType <= !txRegister) then
8.      1‘(n,REGISTER,noResp,IntTime(),0,node,false,false)
9.    else
10.     1‘(n,INVITE,noResp,IntTime(),0,node,false,false)
11. end;
```

LeaseTimeout: A função `LeaseTimeout()` é responsável por habilitar a transição `Lease` do modelo apresentado na Figura 16. O parâmetro `sop` recebido, representa o tempo no momento em que a tupla foi publicada no LTS. A função verifica se a tupla está em espera para processamento um tempo maior do que o definido pela variável global `leaseTime`.

```
1. fun LeaseTimeout sop =
2.  let
3.    val timeSpent = (IntTime()-sop)
4.  in
5.    if(timeSpent > (!leaseTime)) then
6.     true
7.    else
8.     false
9.  end;
```

HoldThread1: A função `HoldThread1()` é utilizada para decrementar o número de *threads* disponíveis no CSCF para processamento de requisições.

```
1. fun HoldThread1() =
```

```
2. let
3.   val ta = (!threadsAvailable1) - 1
4.   fun decThreadsAvailable ta = (threadsAvailable1:=ta)
5. in
6.   decThreadsAvailable(ta);
7.   if (!threadsAvailable1 = 0) then
8.     1`1
9.   else
10.    empty
end;
```

ReleaseThread1: A função `ReleaseThread1()` é utilizada para incrementar o número de *threads* disponíveis no CSCF para processamento de requisições. Em conjunto com a função `HoldThread1()`, faz o controle da capacidade de processamento paralelo do CSCF durante a simulação.

```
1. fun ReleaseThread() =
2. let
3.   val ta = (!threadsAvailable1) + 1
4.   fun incThreadsAvailable ta = (threadsAvailable1:=ta)
5. in
6.   incThreadsAvailable (ta);
7.   1
8. end;
```

A Figura 12 apresenta o modelo CPN hierárquico representando a topologia da rede. Como é possível observar, o DOMÍNIO A consiste de um espaço de tuplas distribuído, representado pelo lugar `DTS`, e quatro Servidores IMS conectados ao DTS, representados pelos módulos `Node1`, `Node2`, `Node3` e `Node4`.

Em cada um dos Servidores IMS existe uma transição representando a operação `out()` e outra transição representando a operação `in()` do paradigma de programação baseado em espaço de tuplas. Cada uma dessas transições possui a função `DTSAccessDelay()` associada para representar o atraso no acesso ao DTS.

A estratégia de modelagem adotada considerou que cada Servidor IMS da rede, representados pelos módulos `Node1`, `Node2`, `Node3` e `Node4` no modelo da Figura 12,

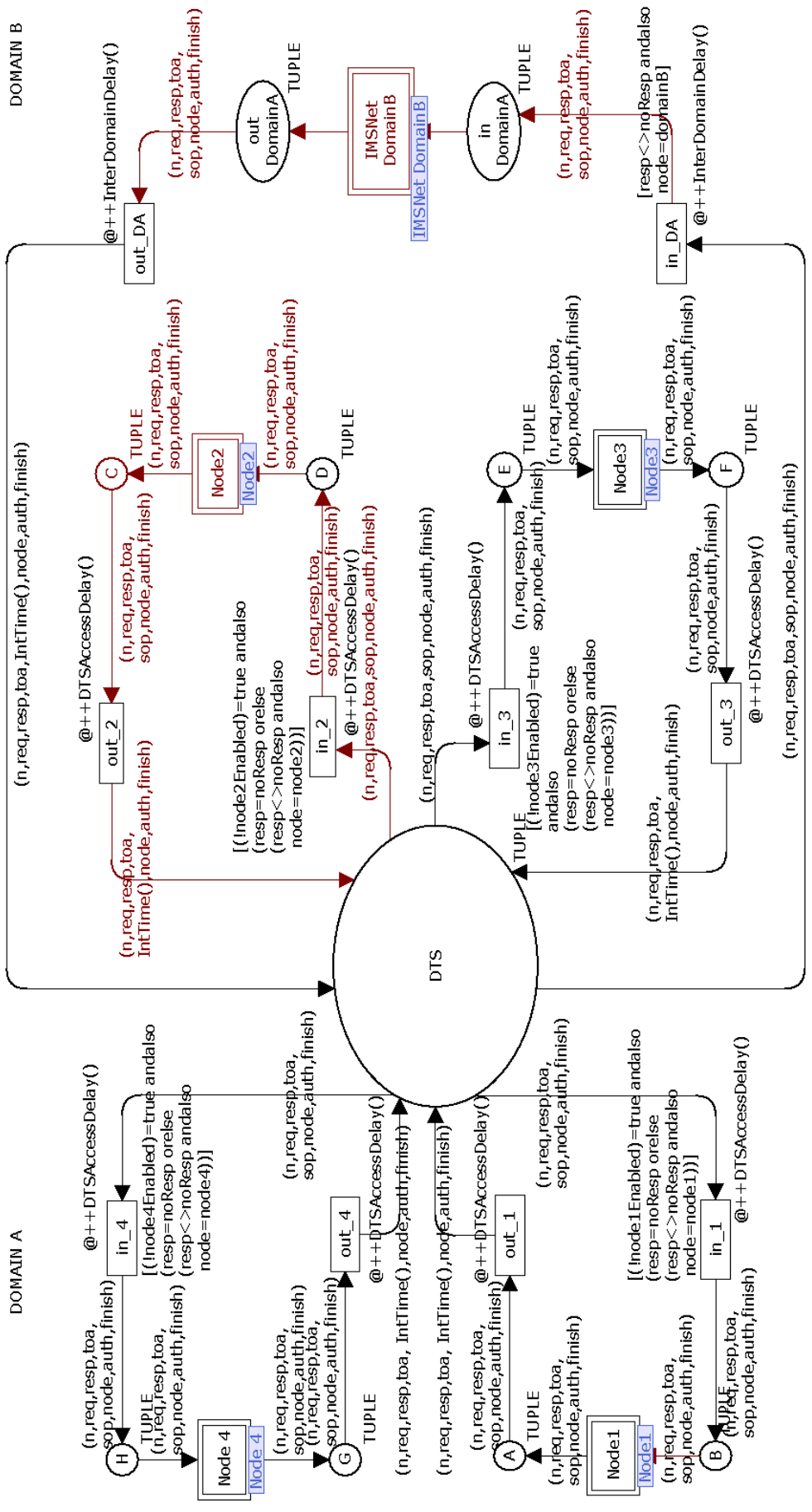

Figura 12: Modelo CPN Hierárquico representando a rede IMS baseada em espaço de tuplas

também serão representados através de um modelo CPN hierárquico, conforme mostra a Figura 13.

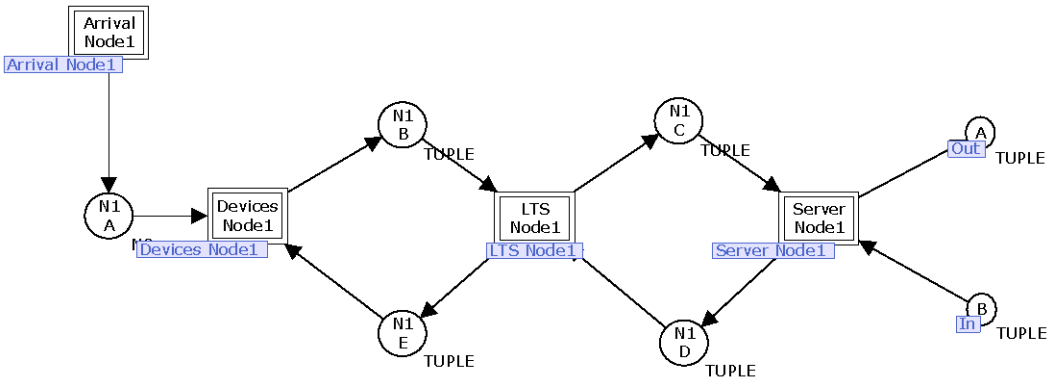

Figura 13: Modelo CPN Hierárquico representando um Servidor IMS da rede

O modelo consiste dos seguintes módulos:

- `Arrival Node1`, responsável pela modelagem da chegada de requisições na rede através de um servidor IMS específico.
- `Devices Node1`, um modelo CPN representando um dispositivo enviando e recebendo mensagens.
- `LTS Node1`, que modela o espaço de tuplas local.
- `Server Node1` que modela o funcionamento do Servidor IMS em relação aos componentes *Events Processor*, CSCF e HSS.

A Figura 14 apresenta o modelo CPN para o módulo `Arrival Node1`.

O módulo `Arrival Node1` representa a chegada de requisições na rede através de um Servidor IMS específico. O lugar `Start` inicia com uma marcação que habilita a transição `starting`. Após o disparo da transição, a função `initSim1()` é executada para iniciar as variáveis globais do modelo.

O lugar `Next Arrival` e a transição `Arrival Node1` representam a chegada de requisições propriamente ditas. A cada disparo da transição `Arrival Node1` a função `NextArrival(!iatNode1)` é executada para incrementar o tempo do modelo, fazendo com que uma nova requisição seja gerada após um intervalo de tempo.

Conforme discutido em (LUO et al., 2009), a chegada de requisições em uma rede IMS obedece à distribuição de Poisson. A função CPN-ML

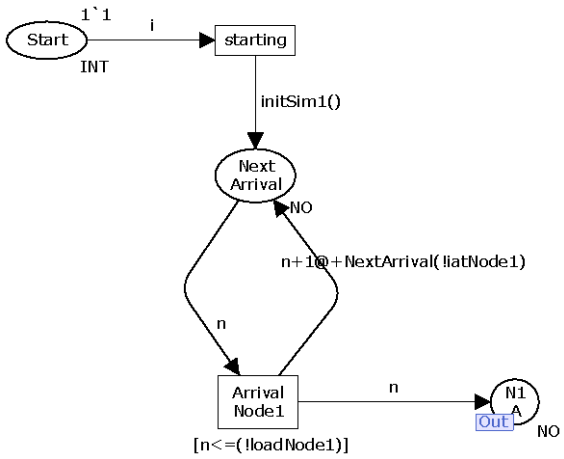

Figura 14: Modelo CPN representando a chegada de requisições (*Arrival Node*)

`NextArrival(!iatNode1)` irá utilizar como intensidade da função de Poisson o parâmetro `iat`, o qual irá determinar o número de requisições por segundo disparadas durante a simulação.

É importante observar que a transição `Arrival Node1` possui a guarda `[(!node1Enabled)=true andalso n<=(!loadNode1)]`. Isso garante que a transição só será disparada quando o Servidor IMS estiver habilitado e enquanto o número de requisições for menor que o total de requisições configurado, representado pela variável `loadNode1`. Nesse caso, a a variável `loadNode1` representa o número de dispositivos conectados em um Servidor IMS.

A Figura 15 apresenta o modelo CPN para o módulo `Device Node1`. Esse modelo representa o fluxo de mensagens de requisição e resposta dos dispositivos conectados à rede.

O lugar `N1A` representa a conexão do módulo `Arrival Node1` com o módulo `Device Node1`, ou seja, representa a ação do usuário ligando um dispositivo para se registrar na rede ou iniciando uma sessão.

O disparo da transição `Create Request` causa a execução da função CPN-ML `CreateTuple(n,node1)`, responsável pela geração de uma marcação com a cor `TUPLE` no lugar `Client Transaction`.

O disparo da transição `Send Request` representa a publicação de uma tupla no LTS. A cada disparo, o tempo do modelo é incrementado pela função `RANDelay()`, representando o atraso na comunicação do UE com a rede. Conforme explicado em (LUO et al., 2009), o atraso considerado na simulação é de 100ms.

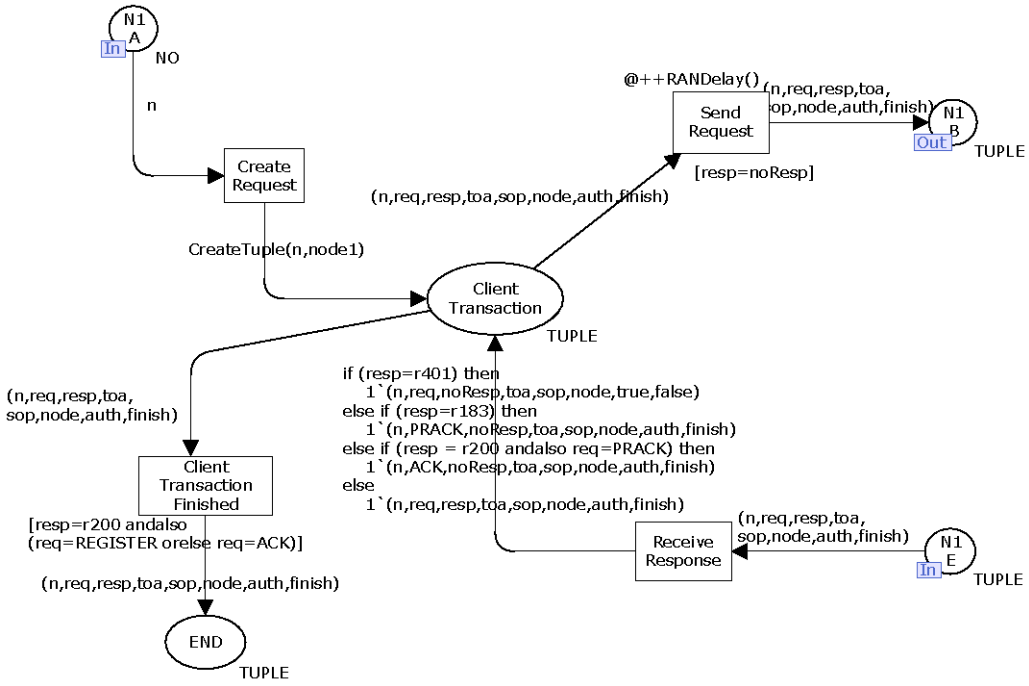

Figura 15: Modelo CPN representando os dispositivos conectados à rede (*Devices*)

A transição `Receive Response` representa o recebimento de uma mensagem pelo dispositivo (e.g., a leitura de uma tupla no LTS). O disparo desta transição causa a execução do código CPN-ML apresentado a seguir, que gera uma marcação no lugar `Client Transaction` de acordo com o tipo de resposta recebida.

Código CPN-ML *- Define o comportamento do dispositivo em função da resposta*[2]

```
1: if(resp=r401) then
2:     1‘(n,req,noResp,toa,sop,node,true,false)
3: else if(resp=r183) then
4:     1‘(n,PRACK,noResp,toa,sop,node,auth,finish)
5: else if(resp=r200 andalso req=PRACK) then
6:     1‘(n,ACK,noResp,toa,sop,node,auth,finish)
7: else
8:     1‘(n,req,resp,toa,sop,node,auth,finish)
```

A Figura 16 apresenta o modelo CPN para o módulo `LTS Node1`. O lugar `LTS Node1` representa o espaço de tuplas. As transições `outC` e `inC` representam as operações de leitura e escrita no LTS realizadas por um dispositivo. As transições `outS` e `inS` representam as operações de leitura e escrita no LTS realizadas pelo Servidor IMS (componentes *Events Processor* e CSCF).

[2]Detalhes da notação utilizada no código podem ser lidos no Apêndice A

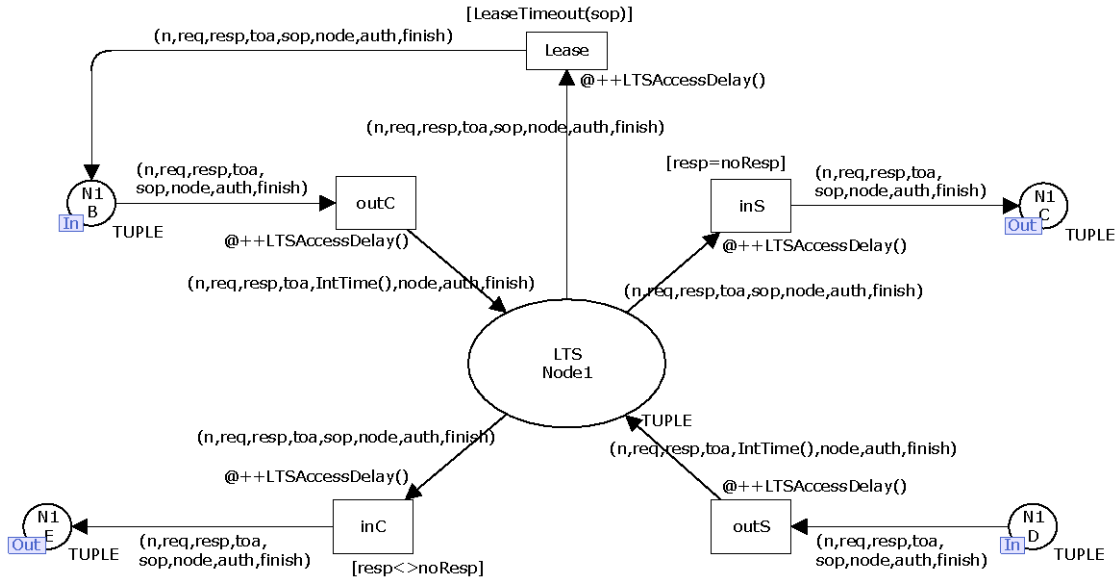

Figura 16: Modelo CPN para o espaço de tuplas local (LTS)

As transições de leitura e escrita possuem a função CPN-ML `LTSAccessDelay` associada, representando o tempo gasto em cada operação.

Adicionalmente, o modelo também representa o conceito de *Lease* implementado pelo espaço de tupla. *Lease* é uma marcação que determina o tempo de vida de uma tupla após ser inserida no espaço. No modelo CPN isto é representado pela transição `Lease` que está associada à função CPN-ML `LeaseTimeout()`. Quando a função retornar `true`, a transição será habilitada fazendo com que a tupla seja publicada no LTS novamente (*lease renew*).

A Figura 17 apresenta o modelo CPN para o módulo `Server Node1`. Esse modelo representa o funcionamento do Servidor IMS em relação ao gerenciamento de sessão e o comportamento do componente *Event Processor*, responsável pela iteração com o espaço de tuplas.

O *Event Processor* é representado no modelo pelos lugares `Event Processor Idle` e `Event Processor Working`, e pelas transições `in LTS` e `in DTS`. A marcação no lugar `Event Processor Idle` representa o número de *threads* disponíveis para realizar leitura de tuplas. Para o contexto desse trabalho, considerou-se uma capacidade de realizar a leitura de 30 tuplas simultaneamente.

Ao ler uma tupla do LTS, a transição `Process Tuple` poderá ser habilitada, determinando o início do processamento da requisição pelo CSCF, representado no mo-

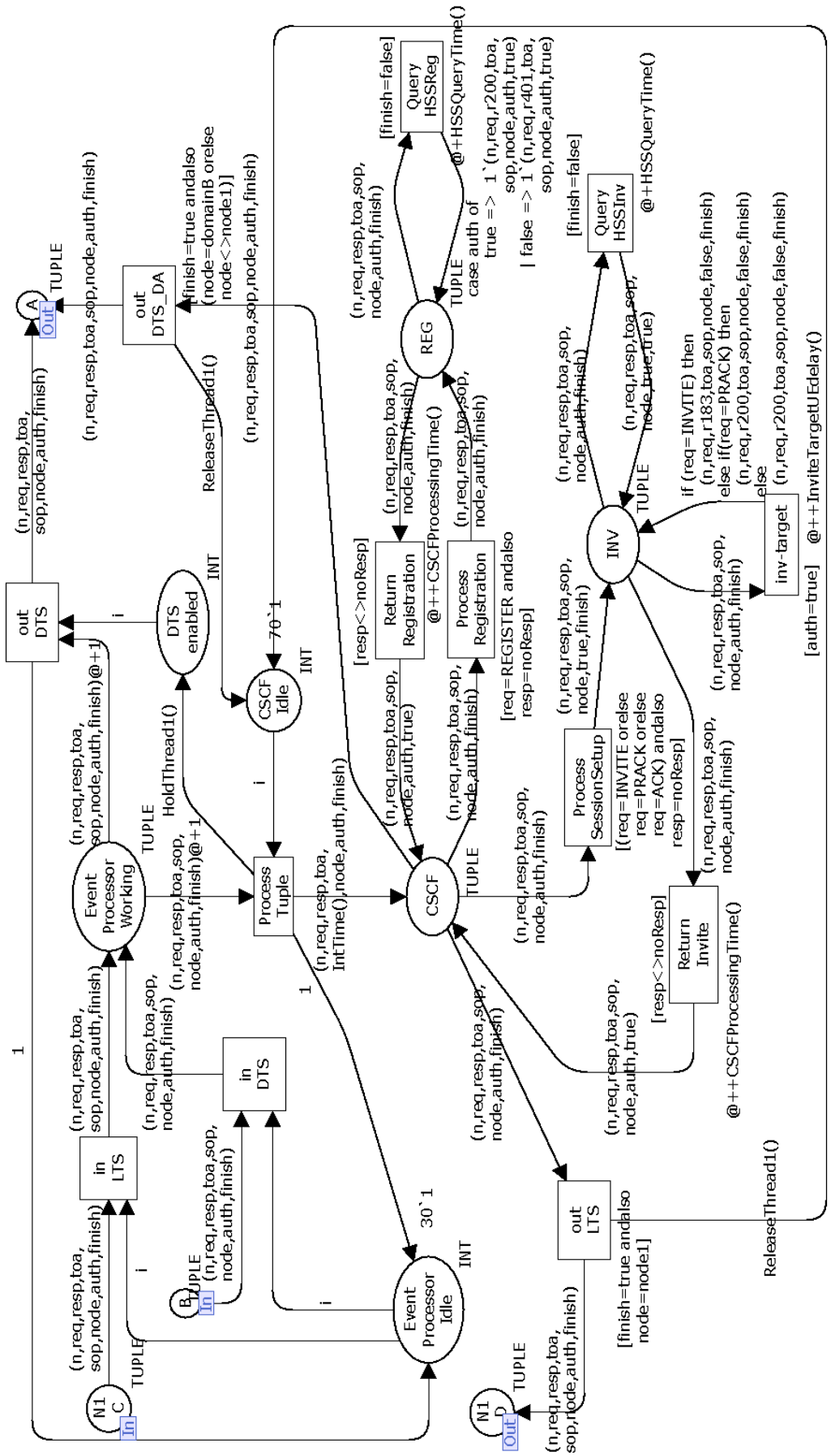

Figura 17: Modelo CPN para o CSCF e o Event Processor

delo pelo lugar CSCF. A capacidade do servidor em processar tuplas simultaneamente é representada pelo lugar CSCF Idle, tendo como início a marcação 70‘1, ou seja, consegue processar 70 requisições simultaneamente.

O balanceamento de carga realizado pelo *Events Processor* é representado no modelo pelos lugares CSCF Idle e DTS enabled. Quando a marcação do lugar CSCF Idle chega a um valor zero, ou seja, CSCF totalmente ocupado, o lugar DTS enable recebe a marcação 1'1, indicando que as tuplas lidas do LTS deverão ser publicadas no DTS para que sejam processadas por qualquer outro Servidor IMS da rede com menor carga. Para propósitos de modelagem, esse controle é feito através das funções CPN-ML HoldThread1() e ReleaseThread1().

É importante observar que na arquitetura IMS padrão, o CSCF é subdivido em três elementos, P-CSCF, S-CSCF e I-CSCF. Na arquitetura baseada em espaço de tuplas foi definida a existência de um único elemento CSCF que contempla todas as funcionalidades, conforme apresentado no capítulo 5.

Ao receber uma tupla, o CSCF poderá realizar o registro do dispositivo na rede, caso seja uma requisição do tipo REGISTER ou estabelecer uma sessão com outro dispositivo, caso seja uma requisição do tipo INVITE. Para efeitos de modelagem, cada um dos dois tipos de requisição foi modelado com seus respectivos lugares e transições.

A operação de registro é representada pelo lugar REG e pelas transições Process Registration, Query HSSreg e Return Registration. A transição Query HSSreg está associada à função CPN-ML HSSQueryTime() representando o tempo gasto para realizar uma consulta no HSS. Após a consulta ao HSS, será gerada uma resposta indicando usuário autenticado (r200) ou falha de autenticação (r401). Adicionalmente, a transição Return Registration está associada à função CPN-ML CSCFProcessingTime(), a qual representa o tempo de processamento do CSCF.

A operação de estabelecimento de sessão é representada pelo lugar INV, e pelas transições Process SessionSetup, Return Invite, Query HSSInv e inv-target. A transição Query HSSInv está associada à função CPN-ML HSSQueryTime(), representando o tempo gasto para realizar uma consulta no HSS. O mesmo ocorre com a operação de registro. A transição inv-target representa a comunicação com o dispositivo alvo ao qual se deseja estabelecer uma sessão. Esta transição está associada à função CPN-ML InviteTargetUEdelay(),

que representa o tempo de comunicação do Servidor IMS com o dispositivo alvo. Adicionalmente, a transição `Return Invite` está associada à função CPN-ML `CSCFProcessingTime()`, a qual representa o tempo de processamento do CSCF.

O modelo CPN da arquitetura baseada em espaço de tuplas apresentado anteriormente possibilita a simulação com diferentes cenários e parâmetros.

- permite habilitar/desabilitar Servidores IMS.
- permite alterar a taxa de chegada de requisições no sistema através dos parâmetros de intervalo entre requisições.
- possibilita a simulação do comportamento considerando diferentes tempos de atraso nos diversos elementos da rede.
- possibilita a simulação do comportamento do sistema alterando parâmetros de capacidade de processamento simultâneo dos componentes CSCF e *Event Processor*.

6 ANÁLISE FORMAL BASEADA EM REDES DE PETRI COLORIDAS

Este capítulo apresenta a validação formal da arquitetura utilizando Redes de Petri Coloridas. A validação é feita através de duas abordagens: (i) análise funcional do SIPSpace, utilizando-se a ferramenta de análise de espaço de estado; e (ii) análise de desempenho da rede através de simulação.

6.1 Análise Funcional do SIPSpace

A análise funcional do SIPSpace está baseada na especificação da transação SIP para a operação INVITE (*SIP INVITE Transaction*), conforme apresentado anteriormente na seção 5.2.1, utilizando o modelo CPN também apresentado na mesma seção.

6.1.1 Análise de Espaço de Estado

A ferramenta CPN gera um relatório de espaço de estado (*state space report*) para realizar análise de propriedades comportamentais de um modelo. Quando o espaço de estado é calculado, pode-se utilizar funções CPN ML de consulta para extrair resultados e efetuar análise com maior nível de detalhes. Adicionalmente, a ferramenta gera um grafo conectado como suporte para análise de propriedades comportamentais.

Para validar as propriedades funcionais do SIPSpace, utilizamos a análise de espaço de estado. As propriedades de interesse são *dead markings*, *dead transitions*, e *live-locks*.

Baseado nas propriedades de interesse, é importante considerar as seguintes questões:

1. A propriedade *dead marking*, ou seja, a marcação final do modelo após a simu-

lação, deve ser os lugares `ClientTransaction` e `ServerTransaction` com as marcações `terminated` e `terminatedS` respectivamente. Isso mostra que o protocolo funciona de acordo com a especificação.

2. Espera-se que o protocolo não apresente *dead code*, ou seja, nenhuma transição que nunca será habilitada.

3. Também não deve apresentar *live-lock*, ou seja, ciclos no espaço de estado. A ausência de *live-locks* pode ser constatada quando o relatório de espaço de estado apresentar o mesmo número de nós do grafo SCC.

Para analisar com mais detalhes as *dead markings* geradas no relatório de espaço de estado, utilizamos a função CPN-ML a seguir:

```
1: fun desired n =
2:     ((Mark.top'ClientTransaction 1 n) == 1`terminated) andalso
3:     ((Mark.top'ServerTransaction 1 n) == 1`terminatedS);
```

Após calcular o espaço de estado e o grafo conectado, as seguintes estatísticas foram geradas.

```
State Space            Scc Graph
   Nodes:  341            Nodes:  341
   Arcs:   441            Arcs:   441
   Secs:   0              Secs:   0
   Status: Full
```

Como pode-se observar, tanto o relatório de espaço de estado como o grafo conectado geraram 341 nós e 441 arcos. Com esse resultado, concluímos que o SIPSpace não possui ciclos (*live-lock*).

Outra propriedade interessante do relatório é o *Best Upper Multi-set Bounds*, conforme mostra o relatório a seguir. Esta propriedade mostra o conjunto de marcações alcançadas em cada lugar do modelo.

```
top'ClientTransaction 1   top'Send_Request 1
 1`calling++               1`INVITE++
 1`proceeding++            1`ACK
```

```
  1‘completed++
  1‘accepted++              top’Send_Response 1
  1‘terminated               1‘r1xx++
                             1‘r2xx++
top’ServerTransaction 1      1‘r3xx
  1‘idle++
  1‘proceedingS++           top’Tuple_Space 1
  1‘completedS++             1‘(INVITE,noneResp)++
  1‘acceptedS++              1‘(ACK,noneResp)++
  1‘terminatedS              1‘(noneReq,r1xx)++
                             1‘(noneReq,r2xx)++
                             1‘(noneReq,r3xx)
```

Analisando os lugares principais, (`ClientTransaction`, `ServerTransaction`, `SendRequest`, `SendResponse` e `Tuple Space`) mostrados no relatório, pode-se observar que:

- A transação cliente, representada no relatório por `top’ClientTransaction 1`, passou por todos os estados possíveis, na ordem correta, ou seja, recebeu todas as marcações representando os devidos estados.
- A transação servidor, representada no relatório por `top’ServerTransaction 1`, passou por todos os estados possíveis, na ordem correta, ou seja, recebeu todas as marcações representando os devidos estados.
- A transação cliente enviou todas requisições possíveis, ou seja INVITE e ACK. Pode-se observar isso no relatório através do lugar `top’Send_Request 1`.
- A transação servidor enviou todas as respostas esperadas, ou seja, r1xx, r2xx e r3xx. Pode-se observar isso no relatório através do lugar `top’Send_Response 1`.
- E por sua vez, o LTS, representado no relatório por `top’Tuple_Space 1` recebeu todas as tuplas possíveis, de requisição e resposta.

Com base no relatório *Best Upper Multi-set Bounds*, concluímos que durante a simulação do SIPSpace todas as transições de estado, requisições e respostas possíveis

são executadas corretamente pelo SIPSpace, seguindo fielmente a especificação do protocolo SIP.

O relatório gerou um *dead code*, i.e., *dead transition* no lugar `top'Stop_T_H 1`. Essa *dead transition* representa o *timeout* na transação servidor quando em espera por um ACK da transação cliente. Nesse modelo, é uma situação esperada.

Finalmente, pode-se observar que o espaço de estado gerou 59 *dead markings*. Analisando com mais detalhes através da função de consulta `derised n` apresentada anteriormente, observamos que 58 *dead markings* estão no estado desejado, ou seja, as transações cliente e servidor no estado *terminated* e *terminatedS* respectivamente. Em uma *dead marking*, a transação cliente está no estado *terminated* e a transação servidor no estado *Idle*. Isso acontece em função do disparo da transição `ClientErr`, a qual é um resultado esperado.

6.1.2 Considerações

A análise funcional do SIPSpace através da ferramenta de cálculo de espaço demonstrou que a mudança na forma de implementação do protocolo SIP não alterou sua semântica, ou seja, o resultado final de um fluxo de mensagens SIP se mantém consistente com a especificação.

6.2 Análise de Desempenho da Arquitetura baseada em Espaço de Tuplas

6.2.1 Métricas e Coleta de Dados

Esta sessão apresenta as métricas de desempenho consideradas para análise da arquitetura baseada em espaço de tuplas e seu relacionamento com o modelo CPN através das funções de monitoração e coleta de dados.

6.2.1.1 Taxa de utilização do CSCF

A taxa de utilização do CSCF, expressada por U_{CSCF_i}, indica, em termos percentuais, a média de utilização do componente CSCF em cada Servidor IMS da rede.

O principal objetivo dessa métrica é verificar a habilidade da arquitetura em re-

alizar balanceamento de carga de requisições, ou seja, dada uma carga acima da capacidade de processamento de um único servidor, verificar se a carga foi distribuída igualmente em todos os Servidores IMS habilitados na rede.

A taxa de utilização é dada pela média de requisições sendo atendidas pelo componente CSCF, em relação à sua capacidade máxima de atendimento simultâneo de requisições. Sendo assim, a taxa de utilização do CSCF será dada por:

$$U_{CSCF_i} = \frac{100 * mr_i}{sr_i} \tag{6.1}$$

onde:

U_{CSCF_i} é a taxa de utilização do componente CSCF no i-ésimo Servidor IMS.

mr_i é a média de requisições sendo atendidas pelo CSCF do i-ésimo Servidor IMS durante a simulação.

sr_i é o número máximo de requisições simultâneas cujo componente CSCF do i-ésimo Servidor IMS consegue processar, ou seja, o número de execuções em paralelo.

Considerando o modelo utilizado para as simulações, a média de requisições sendo atendidas pode ser calculada utilizando-se o recurso de *continuous-time statistics* oferecido pelo simulador CPN. O simulador CPN calcula a média (*timeaveraged*) com base nas informações de tempo do modelo, conforme descrito em (JENSEN; KRISTENSEN, 2009).

A média baseada em tempo considera o intervalo entre duas observações como peso para sua composição, e é dada por:

$$mr_i = \frac{(\sum_{j=1}^{n-1} x_j(mt_{j+1} - mt_j)) + x_n(mt - mt_n)}{mt - mt_1} \tag{6.2}$$

onde:

n é o número de observações realizadas.

x_j é a j-ésima observação.

mt_j é o tempo do simulador no momento da observação j.

x_n é a última observação realizada.

mt é o tempo final do simulador, ou seja, a última marcação de tempo do simu-

lador. Essa variável difere da variável mt_n pois o tempo do simulador pode ter sido incrementado sem que tenha ocorrido uma observação.

Considerado o modelo CPN da arquitetura, a taxa de utilização do CSCF será obtidas com base na média de marcações nos lugares `CSCF`, `INV` e `REG`. A média de marcações será gerada pelo monitor `LoadCSCF`, conforme código CPN-ML disponibilizado no Apêndice B.

A existência de pelo menos uma marcação em um dos lugares `CSCF`, `INV` ou `REG`, indica que o CSCF está sendo utilizado.

6.2.1.2 Tempo médio de residência nas operações de Registro e Início de Sessão

O tempo médio de residência indica o tempo de resposta nas operações de registro (REGISTER) e início de sessão (INVITE), e são expressados por R_r e R_s respectivamente.

A métrica é dada por:

$$R_r = \frac{\sum_{j=1}^{C_{REGISTER}} (mt_{c_{REGISTER_j}} - mt_{s_{REGISTER_j}})}{C_{REGISTER}} \tag{6.3}$$

$$R_s = \frac{\sum_{j=1}^{C_{INVITE}} (mt_{c_{INVITE_j}} - mt_{s_{INVITE_j}})}{C_{INVITE}} \tag{6.4}$$

onde,

$C_{REGISTER}$ é o número de entidades (tuplas) de registro processadas.

C_{INVITE} é o número de entidades (tuplas) de início de sessão processadas.

$mt_{c_{REGISTER_j}}$ é o tempo do simulador no momento da chegada da j-ésima requisição de registro no sistema, ou seja, momento da execução de uma operação `out()` representando a publicação de uma requisição REGISTER no LTS.

$mt_{s_{REGISTER_j}}$ é o tempo do simulador no momento de saída da j-ésima requisição de registro do sistema, ou seja, leitura de uma tupla realizada pelo dispositivo cliente através da operação `in()`, cujo código de resposta indique a finalização do processamento de registro.

$mt_{c_{INVITE_j}}$ é o tempo do simulador no momento da chegada da j-ésima requisição de

estabelecimento de sessão no sistema, ou seja, momento da execução de uma operação `out()` representando a publicação de uma requisição INVITE no LTS.

$mt_{s_{INVITE_j}}$ é o tempo do simulador no momento de saída da *j*-ésima requisição de estabelecimento de sessão do sistema, ou seja, leitura de uma tupla realizada pelo dispositivo cliente através da operação `in()`, cujo código de resposta indique a finalização do processamento de estabelecimento de sessão.

Com base no modelo CPN da arquitetura, as variáveis $mt_{c_{REGISTER_j}}$ e $mt_{c_{INVITE_j}}$ serão obtidas no momento do disparo da transição `Create Request`, e as variáveis $mt_{s_{REGISTER_j}}$ e $mt_{s_{INVITE_j}}$ serão obtidas no momento do disparo da transição `Receive Response`.

Para calcular o intervalo foram definidos dois monitores, `SessionSetupDelay` e `RegistrationDelay`, cujo código CPN-ML podem ser vistos no Apêndice B.

Ambos os monitores estão associados à transição `Receive Response`, ou seja, a cada disparo dessa transição o simulador CPN Tool irá executar os monitores para coletar o intervalo de tempo de uma requisição específica. O monitor `SessionSetupDelay` somente irá calcular o tempo de residência quando a requisição estiver com um valor `ACK` (requisição aceita) e código de resposta *r200* (com sucesso). O monitor `RegistrationDelay` somente irá calcular o tempo de residência quando a requisição for uma operação `REGISTER` e receber um código de resposta *r200*.

Ao final da simulação, o simulador CPN Tool calcula a média de tempo com base nas coletas de dados realizadas pelos monitores.

6.2.1.3 Tempo médio de espera no LTS

O tempo médio de espera no LTS indica o tempo em que as requisições permanecem em cada LTS desde a sua publicação até a leitura pelo componente *Events Processor*, e é dado por:

$$E[w]_{LTS_i} = \frac{\sum_{j=1}^{C}(mt_{in_{LTS_{i_j}}} - mt_{out_{LTS_{i_j}}})}{C} \tag{6.5}$$

onde:

$E[w]_{LTS_i}$ é o tempo médio de espera no LTS do *i*-ésimo Servidor IMS.

C é o número de tuplas publicadas em cada LTS.

$mt_{out_{LTS_{i_j}}}$ é o tempo do simulador no momento da publicação da j-ésima tupla no LTS do i-ésimo Servidor IMS.

$mt_{in_{LTS_{i_j}}}$ é o tempo do simulador no momento da leitura da j-ésima tupla do LTS do i-ésimo servidor IMS.

Com base no modelo CPN da arquitetura, o parâmetro $mt_{out_{LTS_{i_j}}}$ será obtido no momento do disparo da transição `OutC` e o parâmetro $mt_{in_{LTS_{i_j}}}$ será obtido no momento do disparo da transição `inLTS`.

O intervalo de tempo de espera de cada tupla será calculado pelo monitor `WaitingTimeLTS`, conforme código CPN-ML apresentado no Apêndice B. Ao final, o simulador CPN Tool calcula a média de tempo de espera com base nas coletas de dados realizadas. .

6.2.1.4 Tempo médio de espera no DTS

O tempo médio de espera no DTS indica o tempo em que as requisições permanecem no espaço de tuplas distribuído, desde a sua publicação até a leitura pelo componente *Events Processor* de qualquer um dos Servidores IMS da rede. A métrica é dado por:

$$E[w]_{DTS} = \frac{\sum_{j=1}^{C}(mt_{in_{DTS_j}} - mt_{out_{DTS_j}})}{C} \tag{6.6}$$

onde:

$E[w]_{DTS}$ é o tempo médio de espera no DTS.

C é o número de tuplas publicadas.

$mt_{out_{DTS_j}}$ é o tempo do simulador no momento da publicação da j-ésima tupla no DTS.

$mt_{in_{DTS_j}}$ é o tempo do simulador no momento da leitura da j-ésima tupla do DTS.

Com base no modelo CPN da arquitetura, o parâmetro $mt_{out_{DTS_j}}$ será obtido no momento do disparo de uma das transições `out1`, `out2`, `out3`, `out4` ou `outDA` e o parâmetro $mt_{in_{DTS_j}}$ será obtido no momento do disparo da transição `inDTS` de cada um dos Servidores IMS.

O intervalo de tempo de espera de cada tupla será calculado pelo monitor `WaitingTimeDTS`, conforme código CPN-ML apresentado no Apêndice B. Ao final, o simulador CPN Tool calcula a média de tempo de espera com base nas coletas de dados realizadas.

6.2.1.5 Média de tuplas em espera no LTS

A média de tuplas no LTS irá indicar o número médio de tuplas em espera no espaço de tuplas local de cada um dos Servidores IMS. Esse indicador é importante pois poderá ser utilizado como parâmetro para cálculo da capacidade necessária de memória para o Servidor IMS.

Considerando o modelo temporizado utilizado para as simulações, a média de tuplas no LTS será calculada utilizando-se o recurso de *continuous-time statistics* oferecido pelo simulador, conforme já utilizado para a métrica de Taxa de Carga do CSCF.

Nesse sentido, a média de tuplas em espera no LTS é dada por:

$$E[n_q]_{LTS_i} = \frac{(\sum_{j=1}^{n-1} x_j(mt_{j+1} - mt_j)) + x_n(mt - mt_n)}{mt - mt_1} \tag{6.7}$$

onde:

$E[n_q]_{LTS_i}$ é a média de tuplas em espera no LTS do i-ésimo Servidor IMS.

n é o número de observações.

x_j é a j-ésima observação realizada.

mt_j é o tempo do simulador no momento da j-ésima observação.

x_n é a última observação realizada.

mt é o tempo final do simulador, ou seja, a última marcação de tempo do simulador. Essa variável difere da variável mt_n pois o tempo do simulador pode ter sido incrementado sem que tenha ocorrido uma observação.

A coleta de dados para a média de tuplas em espera no DTS será feita pelo monitor `AvgTuplesLTS`, conforme código CPN-ML apresentado no Apêndice B. Em cada observação, o monitor conta o número de marcações existente nos lugares `LTSi`, `NiC` e *NiD*. É importante destacar que existirá um monitor `AvgTuplesLTS` para cada Servidor IMS. Ao final, cada um dos monitores gera a média para o seu respectivo LTS

com base no total acumulado em cada observação.

6.2.1.6 Média de tuplas em espera no DTS

A média de tuplas no DTS irá indicar o número médio de tuplas em espera no espaço de tuplas distribuído. Esse indicador é importante pois poderá ser utilizado como parâmetro para cálculo da capacidade necessária de memória compartilhada ou espaço em disco para o DTS.

Considerando o modelo temporizado utilizado para as simulações, a média de tuplas no DTS será calculada utilizando-se o recurso de *continuous-time statistics* oferecido pelo simulador, conforme utilizado na média de tuplas em espera no LTS e na taxa de utilização do CSCF.

Nesse sentido, a média de tuplas em espera no DTS é dada por:

$$E[n_q]_{DTS} = \frac{(\sum_{j=1}^{n-1} x_j(mt_{j+1} - mt_j)) + x_n(mt - mt_n)}{mt - mt_1} \tag{6.8}$$

onde:

$E[n_q]_{DTS}$ é a média de tuplas em espera no DTS.

n é o número de observações.

x_j é a j-ésima observação realizada.

mt_j é o tempo do simulador no momento da j-ésima observação.

x_n é a última observação realizada.

mt é o tempo final do simulador, ou seja, a última marcação de tempo do simulador. Essa variável difere da variável mt_n pois o tempo do simulador pode ter sido incrementado sem que tenha ocorrido uma observação.

A coleta de dados para a média de tuplas em espera no DTS será feita pelo monitor `AvgTuplesDTS`, conforme código CPN-ML apresentado no Apêndice B. Em cada observação, o monitor conta o número de marcações existente nos lugares `DTS`, `B1`, `D1`, `E1` e `H1`. Ao final, ele gera a média com base no total acumulado em cada observação.

6.2.1.7 Vazão do sistema

A Vazão do sistema, identificada por V, indica o número de requisições por segundo processada pelo sistema em diferentes condições de carga.

A vazão é dada por

$$V = \frac{C}{mt} \tag{6.9}$$

onde:

C corresponde ao número de dispositivos conectados na rede. Para fins de simulação, C foi definido em 25.000, sendo que cada entidade envia uma requisição.

mt é o tempo global do simulador ao final da simulação.

6.2.1.8 Escalabilidade

Em geral, a escalabilidade é a habilidade do sistema operar de forma eficiente e com qualidade de serviço à medida em que se aumenta a carga de trabalho. Em (JOGALEKAR; WOODSIDE, 2000) foi apresentado um arcabouço de métricas e estratégias para avaliação de escalabilidade.

No contexto deste trabalho, foi considerado a análise através de três abordagens: abordagem baseada na vazão do sistema (V), abordagem baseada nos tempos de resposta (R_r e R_s) e abordagem baseada na produtividade (F).

Análise com base na vazão do sistema A escalabilidade com base na vazão do sistema será demonstrada a partir da simulação do sistema completo (todos os Servidores IMS habilitados) exposto a diferentes condições de carga. O fator de escalabilidade será obtida pela razão entre vazão (V) e carga (λ), e é dada por

$$\psi = \frac{V}{\lambda} \tag{6.10}$$

Considerando que o fator de escalabilidade é a razão entre vazão e carga, o sistema se mantém escalável enquanto o fator de escalabilidade for mantido com diferentes cargas.

Exemplo: supondo uma execução com $\lambda = 100req/seg$ e $V = 100req/seg$, tem-

se um fator de escalabilidade $\psi = 1$; na próxima execução com $\lambda = 200req/seg$ e $V = 120req/seg$, tem-se um fator de escalabilidade $\psi = 0.6$. Neste caso, pode-se afirmar que para $\lambda = 200req/seg$ o sistema não escalou.

Nesse sentido, considerando que as simulação se baseou em uma rede com quatro Servidores IMS, o objetivo da abordagem baseada na vazão é indicar o limite de carga em que a rede se mantém escalável, considerando apenas os quatro Servidores IMS.

Análise com base em tempos de resposta Para análise com base no tempo de resposta serão utilizadas as duas métricas definidas a seguir:

1. Speedup (S), mede a taxa de crescimento de trabalho realizado com p Servidores IMS, comparado ao trabalho realizado com apenas um Servidor IMS. O SpeedUp é dado por

$$S\,p = \frac{R_1}{R_p} \tag{6.11}$$

onde:

R_1 é o tempo de resposta para simulação com apenas um Servidor IMS habilitado.

R_p é o tempo de resposta para simulação com p Servidores IMS habilitados.

O SpeedUp ideal ou SpeedUp linear é obtido quando $S_p = p$.

Considerando que existe um percentual (pc_r) de requisições de registro e um percentual (pc_s) de requisições de início de sessão, utilizaremos um tempo médio geral de residência (R) que é a média ponderada dada por

$$R = \frac{(R_r * peso_r) + (R_s * peso_s)}{100} \tag{6.12}$$

onde:

R_r é o tempo médio de residência nas operações de registro

R_s é o tempo médio de residência nas operações de início de sessão

$peso_r$ é o peso do tempo médio de residência nas operações de registro com base no percentual (pc_r) de requisições de registro definidos na simulação.

$peso_s$ é o peso do tempo médio de residência nas operações de início de sessão com base no percentual (pc_s) de requisições de início de sessão definidos na simulação.

2. Eficiência (E) é a taxa de trabalho por número de Servidores IMS habilitados, definido por

$$Ep = \frac{S_p}{p} \tag{6.13}$$

onde:

S_p é o valor obtido em (6.11)

p é o número de Servidores IMS habilitados.

Valores são obtidos entre 0.0 e 1.0, sendo maior a eficiência à medida em que se aproxima de 1.0, neste caso, SpeedUp linear.

Escalabilidade baseada na Produtividade Finalizando a análise de escalabilidade, foi adotada a abordagem baseada em produtividade, apresentada em (JOGALEKAR; WOODSIDE, 2000). O sistema é tido como escalável à medida em que a produtividade é mantida com o aumento da carga.

Para a métrica baseada em produtividade, a equação geral de escalabilidade é dada por

$$\psi(p_1, p_2) = \frac{F(p_2)}{F(p_1)} \tag{6.14}$$

onde:

$\psi(p_1, p_2)$ é o fator de escalabilidade partindo do cenário 1 para o cenário 2.

$F(p_1)$ é a produtividade do sistema para o cenário 1.

$F(p_2)$ é a produtividade do sistema para o cenário 2.

A produtividade é dada por:

$$F(p) = \frac{V_p * f_{QoS_p}(\sigma_1, .., \sigma_n)}{CT_p} \tag{6.15}$$

onde,

V_p é a vazão do sistema para o cenário de simulação com p Servidores IMS.

$f_{QoS_p}(\sigma_1, .., \sigma_n)$ é uma função de QoS quando considerado p Servidores IMS. Nesse caso, a qualidade de serviço será definida em função de uma lista de parâmetros, representados por $\sigma_1, .., \sigma_n$, definidos estrategicamente de acordo com a característica do sistema.

CT_p é o custo de execução proporcional considerando p Servidores IMS, também podendo ser expressado como a relação custo/benefício.

Para efeito de aplicação do método da produtividade na análise da arquitetura baseada em espaço de tuplas, a função de QoS $f_{QoS_p}(\sigma_1, .., \sigma_n)$ será calculada com base no tempo médio de resposta do sistema, definido anteriormente por R.

Nesse caso, a função de QoS adotada para análise é dada por:

$$f_{QoS_p}(\sigma) = \frac{1}{(\frac{R_p}{\hat{R}})} \tag{6.16}$$

onde,

$f_{QoS_p}(\sigma)$ representa o índice de QoS em função do tempo de resposta ($\sigma = R$), quando considerado p Servidores IMS habilitados.

R_p é o tempo médio de resposta quando considerado p Servidores IMS habilitados.

$\hat{R}$ é um tempo médio de resposta alvo, ou seja, o tempo médio que se espera atingir como aceitável. Nesse caso, à medida em que $R \to \hat{R}$, $f_{QoS_p}(\sigma) \to 1$.

Também para efeito de aplicação do método, o custo é dado por:

$$CT = \begin{cases} 1, & \text{quando } p = 1 \text{ ou seja, um Servidor IMS} \\ k * CT_{(p-1)}, & \text{quando } p > 1 \end{cases} \tag{6.17}$$

onde, k é uma constante que multiplica o custo a cada Servidor IMS adicionado à rede.

O sistema é considerado escalável de p_1 para p_2 se $\psi \geq 1$. Adicionalmente, é possível definir um valor mínimo limite (*threshold* τ) para indicar escalabilidade. Nesse caso, o sistema é tido como escalável se $\psi \geq \tau$.

6.2.2 Cenário Base

A análise de desempenho da arquitetura assume um cenário típico de registro de dispositivos e estabelecimento de sessão entre dispositivos localizados em dois domínios de rede, Domínio A (*Home Network*) e Domínio B (*Visited Network*), conforme utilizado por (LUO et al., 2009).

A Figura 18 ilustra esse cenário considerando uma arquitetura IMS padrão.

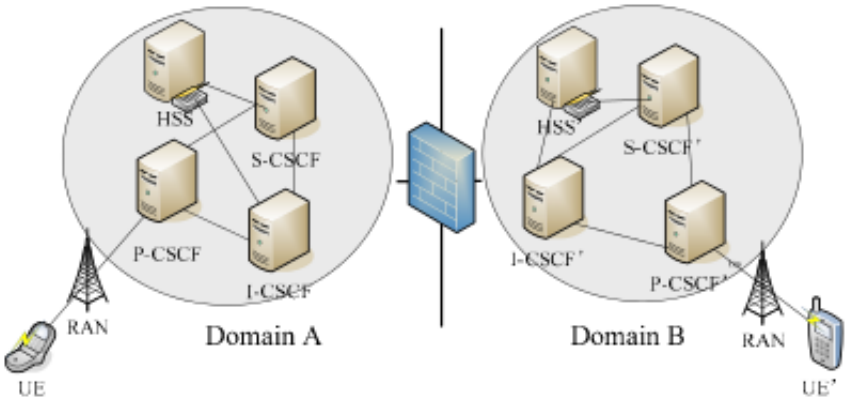

Figura 18: Cenário para análise de desempenho e construção dos modelos CPN

Com base nesse modelo, as operações na rede IMS podem ocorrer da seguinte forma:

1. Um dispositivo localizado no Domínio A, sendo esse domínio o seu *Home Network*, enviando uma requisição de registro. Nesse caso, a requisição é feita localmente, uma vez que o dispositivo está localizado dentro de sua rede, Domínio A.

2. Um dispositivo localizado no Domínio B, sendo esse domínio a sua *Visited Network*, enviando uma requisição de registro. Nesse caso, o dispositivo envia a requisição para a rede em que está localizado no momento, no caso o Domínio B, e então os Servidores IMS do Domínio B repassam a requisição para o Domínio A.

3. Dispositivo localizado no Domínio A (*Home Network*) tentando estabelecer uma sessão com dispositivos da própria rede.

4. Dispositivo localizado no Domínio A (*Home Network*) tentando estabelecer uma sessão com dispositivos localizados no Domínio B.

5. Dispositivo localizado no Domínio B (*Visited Network*) tentando estabelecer uma sessão com dispositivos da sua rede Domínio A, ou da rede em que está localizado, Domínio B

A simulação foi dividida em três etapas:

1. A primeira etapa (Simulação 01) considerou uma carga fixa de 1200 req/seg (acima de 4 milhões por hora), sendo disparadas por dispositivos localizados no Domínio B e dispositivos localizados no Domínio A conectados a um único Servidor IMS, ou seja, apenas um dos Servidores IMS recebe as requisições. Nessa etapa, a simulação inicia com um Servidor IMS habilitado (o que recebe as requisições), e a cada execução da simulação um novo Servidor IMS é habilitado (de 1 a 4 servidores IMS habilitados).

2. A segunda etapa (Simulação 02) considerou a rede completa, com os quatro Servidores IMS habilitados, variando a carga de requisições de 100 a 1150 req/seg, sendo disparadas por dispositivos localizados no Domínio B e no Domínio A com os quatro Servidores IMS recebendo requisições de maneira não uniforme (número de disparos diferentes para cada Servidor IMS).

3. Na terceira etapa (Simulação 03) foi realizada a simulação da rede IMS padrão, com o objetivo de comparar os resultados obtidos com os resultados obtidos na simulação da arquitetura baseada em espaço de tuplas. O modelo CPN para a arquitetura padrão pode ser analisado na Figura 36, no apêndice C.

Um parâmetro importante utilizado é o número de requisições de registro (REGISTER) e requisições de estabelecimento de sessão (INVITE); para este parâmetro foi utilizado uma distribuição uniforme de 60% para REGISTER e 40% para INVITE, seguindo os estudos sobre redes IMS apresentados em (KANG et al., 2007).

O intervalo de chegada de requisições segue uma distribuição de Poisson, e os demais intervalos de tempo (atrasos na rede, nas operações de escrita e leitura e o tempo de processamento dos elementos) seguem distribuições exponenciais, conforme funções CPN-ML apresentadas na sessão 5.2.2.

A Tabela 8 apresenta um resumo dos tempos utilizados nas simulações.

6.2.3 Resultados

6.2.3.1 Simulação 01 - Variação do número de Servidores IMS habilitados

O primeiro cenário de simulação teve como objetivo a análise de aspectos de escalabilidade da arquitetura baseada em espaço de tuplas, considerando uma taxa de

Tabela 8: Parâmetros utilizados durante as simulações

Parâmetro/Função	**Descrição**	**Valor**
RANDelay	Atraso no acesso à rede IMS	100ms
LTSAccessDelay	Atraso no acesso ao LTS	10ms
DTSAccessDelay	Atraso no acesso ao DTS	15ms
InterDomainDelay	Atraso na comunicação entre Domínios A e B	25ms
HSSQueryTime	Tempo gasto para consultar o HSS	2ms e 15ms
CSCFProcessingTime	Tempo de processamento do CSCF	3ms
InviteTargetUEDelay	Atraso na comunicação com o UE destino	50ms e 160ms

chegada (λ) de 1200 requisições por segundo, variando o número de Servidores IMS habilitados. As tabelas 9 e 10 mostram um resumo dos resultado obtidos.

Tabela 9: Resultados da Simulação 01 (a)

$No.Serv$	U_{CSCF_1}	U_{CSCF_2}	U_{CSCF_3}	U_{CSCF_4}	Rr	Rs
1	98,29%	—	—	—	13s	26s
2	95,91%	91,53%	—	—	3,5s	7,8s
3	81,60%	71,97%	69,96%	—	0,98s	2,3s
4	73,61%	49,73%	47,56%	50,66%	0,45s	1,1s

Tabela 10: Resultados da Simulação 01 (b)

$No.Serv$	$E[w]_{DTS}$	$E[n_q]_{DTS}$	$V(req/seg)$	$\psi(Sp)$	Ep	$\psi(F)k = 2$	$\psi(F)k = 1.5$
1	3779ms	15704	440	1	1	1	1
2	940ms	3752	829	1,88	0,94	3,4	4,5
3	189ms	760	909	2,06	0,69	6,5	11,3
4	33ms	130	919	2,09	0,52	9,3	16,2

Como pode-se observar nos gráficos das Figuras 19 e 20, à medida em que se aumenta o número de Servidores IMS na rede, aumenta a vazão do sistema e diminui o tempo de resposta das operações de registro e estabelecimento de sessão. Quando executado com apenas um Servidor IMS na rede, o tempo de resposta para operações de registro e estabelecimento de sessão é de aproximadamente 14s e 26s respectivamente, ao passo que na execução com 4 Servidores IMS os tempos de registro e estabelecimento de sessão caem para 0,45s e 1,10s respectivamente.

As gráficos das Figuras 21 e 22 apresentam os resultados de escalabilidades obtidos.

Para esse cenário de simulação foram utilizadas duas abordagens para avaliação da escalabilidade: escalabilidade com base no SpeedUp e Eficiência, conforme definidos

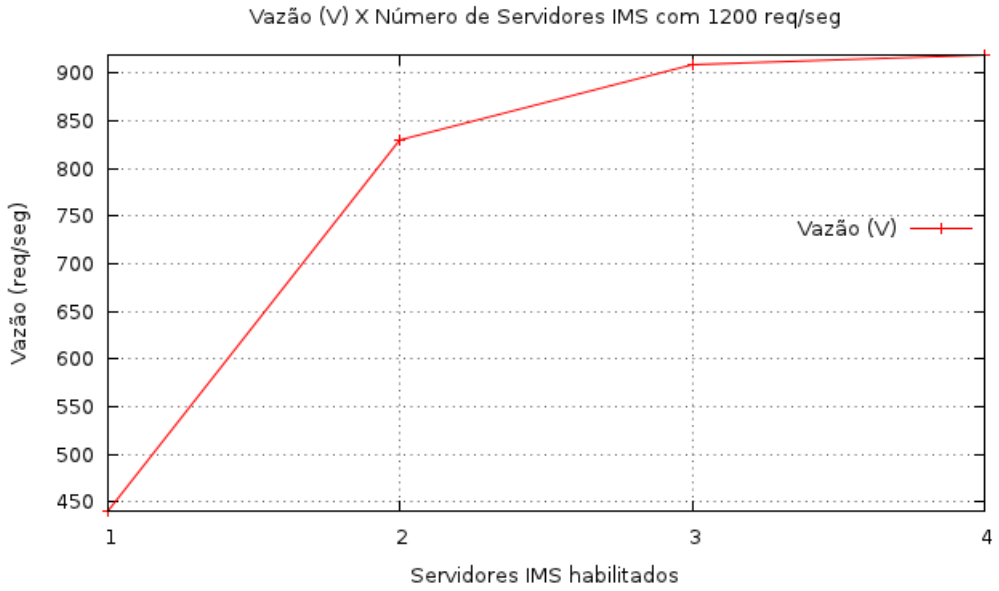

Figura 19: Simulação 01 - Vazão X Número de Servidores IMS

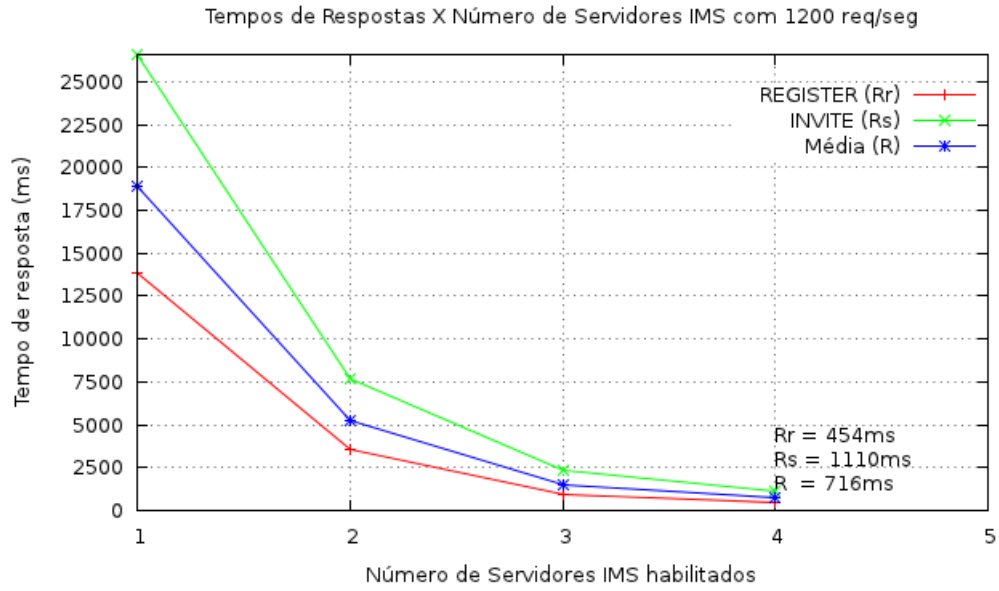

Figura 20: Simulação 01 - Tempos de Resposta X Número de Servidores IMS

pelas equações (6.11) e (6.13); escalabilidade com base na equação de produtividade (6.14).

Na Figura 21 é observado a escalabilidade com base no SpeedUp ($S\,p$) e Eficiência ($E\,p$). É importante notar que para o cenário com carga de 1200 req/seg o equilíbrio entre SpeedUp e Eficiência ocorre com a utilização de dois e três Servidores IMS. Esse resultado demonstra que para o cenário apresentado, até três Servidores IMS na rede seriam suficientes, uma vez que o SpeedUp aumenta e a Eficiência se mantém em

relação à execução com apenas um Servidor IMS.

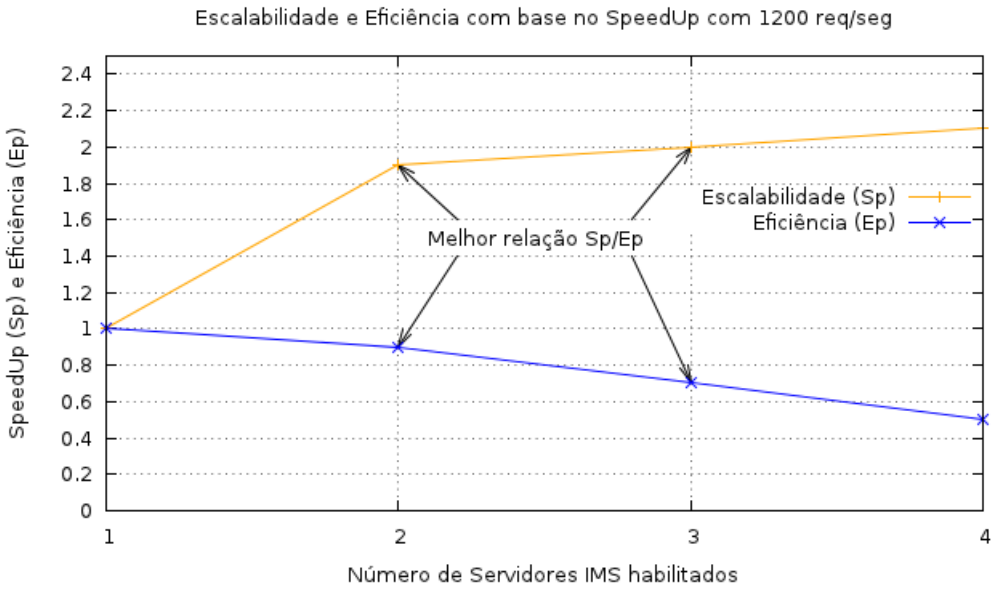

Figura 21: Simulação 01 - Escalabilidade e Eficiência com base no SpeedUp

O gráfico da Figura 22 apresenta a escalabilidade com base na equação de produtividade (6.14). Para efeito de demonstração da equação de produtividade, utilizamos os seguintes parâmetros:

- Para a função de QoS (6.16), foi adotado como base um tempo de resposta médio aceitável de 700ms, ou seja, o índice de QoS aumenta à medida em que o tempo de resposta obtido com a simulação tende a 700ms. O tempo de resposta limitante de 700ms foi adotado com base na execução da simulação com apenas uma requisição, ou seja, foi o melhor tempo alcançado com os modelos CPN adotados[1].

- Para o custo por unidade de processamento (6.17) foram considerados dois valores para a variável k: $k = 2$ representando um custo adicional de 100% por adição de Servidor e $k = 1.5$ representando um custo adicional de 50% por adição de Servidor IMS. Nesses casos, para cada Servidor IMS adicionado à rede o custo aumenta 100% ou 50% em relação ao custo anterior[2].

[1] O valor de 700ms foi adotado para efeito de demonstração da abordagem. Para uma análise mais precisa, é necessária a definição de uma função de QoS que utilize outras variáveis, além do tempo de resposta.

[2] Os valores de 50% e 100% foram adotados para efeito de demonstração da abordagem. Para uma análise mais precisa, é necessário um estudo mais preciso de custo por unidade de processamento considerando aspectos de hardware, custo de gerenciamento da infraestrutura e custo operacional.

Com os parâmetros definidos anteriormente, pode-se observar a comprovação da escalabilidade tanto para custo de 50% quanto para custo de 100% por Servidor IMS. Esse comportamento ocorre devido à melhora nos tempos de resposta que sobrepõem o custo adicional por Serivodor IMS.

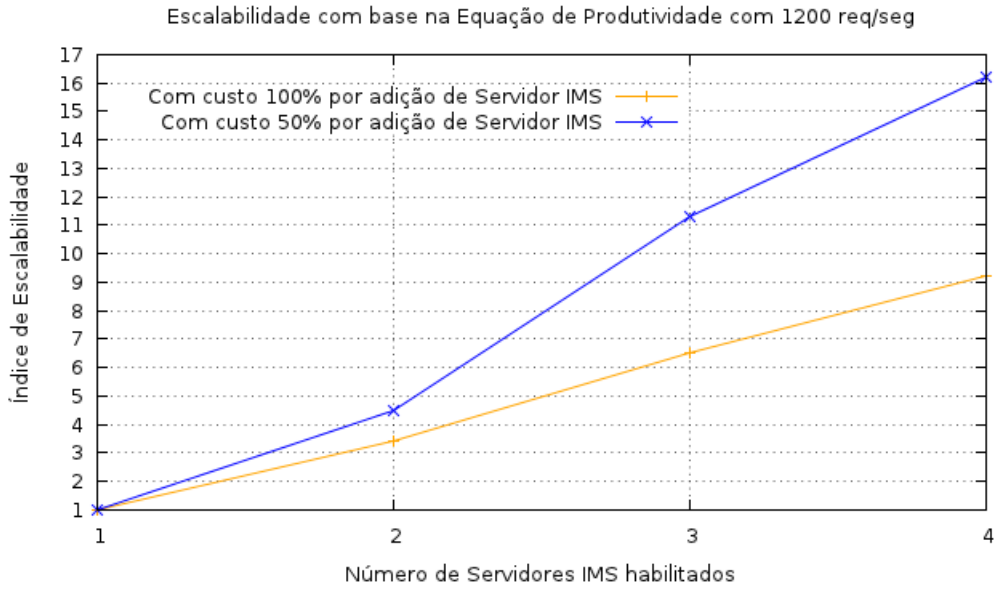

Figura 22: Simulação 01 - Escalabilidade com base na equação de produtividade

Os gráficos das Figuras 23 e 24 mostram a média de tuplas e o tempo de espera no DTS. Nota-se que à medida em que aumenta-se o número de Servidores IMS na rede, o tempo de espera reduz significativamente, reduzindo também a média de tuplas em espera durante a simulação.

Por fim, uma constatação importante é a capacidade de balanceamento de carga do sistema realizado pelo componente *Events Processor* (ver Figura 6). Essa observação pode ser feita no gráfico da Figura 25, que apresenta a utilização de cada Servidor IMS da rede.

Quando executado com apenas um Servidor IMS na rede, a taxa de carga média do CSCF é de 98%, ou seja, ocorre uma saturação do sistema. Quando executado com dois Servidores IMS na rede, a carga é igualmente distribuída, ainda que com uma taxa média de utilização de 95%. À medida em que são habilitados três e quatro Servidores IMS, observa-se a distribuição da carga de trabalho.

A arquitetura apresentada neste trabalho não depende de nenhum esquema de balanceamento de carga externo. Ao chegar uma requisição (e.g uma tupla publicada no LTS), o *Events Processor* verifica a carga do CSCF; caso este não tenha capacidade

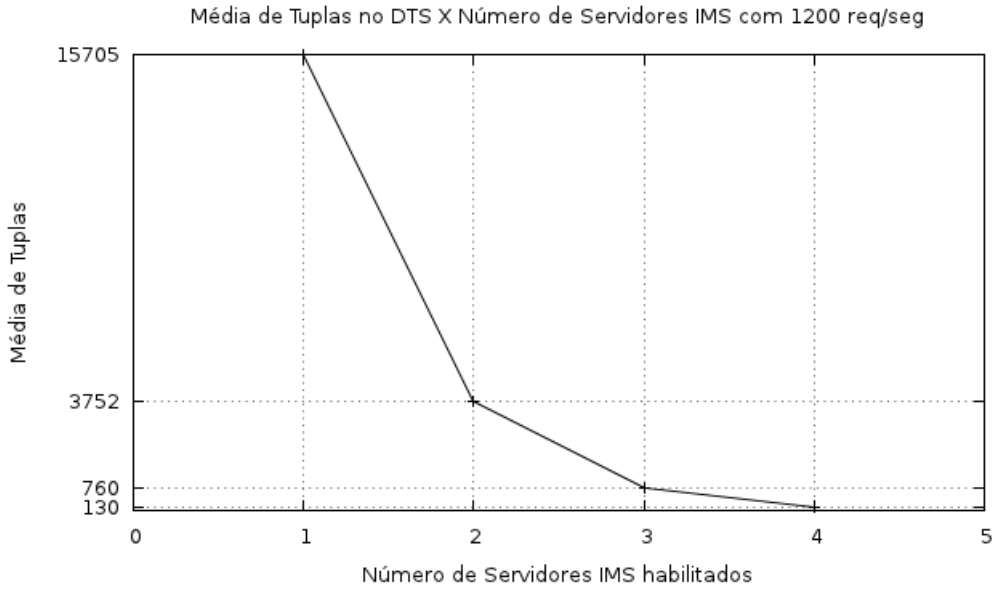

Figura 23: Simulação 01 - Média de Tuplas no DTS

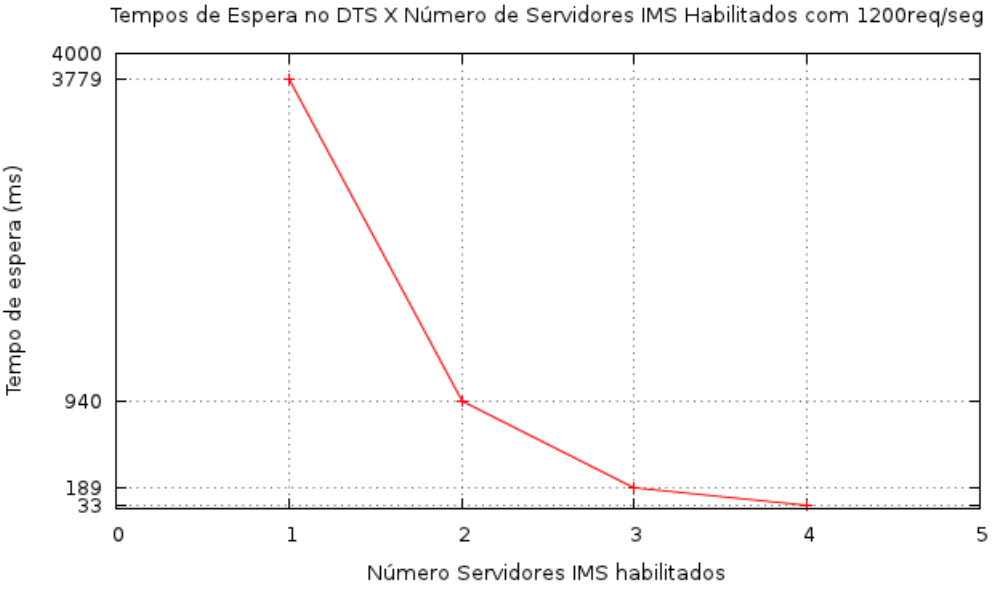

Figura 24: Simulação 01 - Tempo de Espera no DTS

para processar a requisição, o *Events Processor* publica a requisição no DTS para que qualquer Servidor IMS da rede com capacidade ociosa possa realizar o processamento.

6.2.3.2 Simulação 02 - Variação da Carga

O segundo cenário de simulação teve como objetivo a análise do comportamento do sistema em diferentes condições de carga. As tabelas 11, 12, 13 e 14 apresentam um resumo dos resultados obtidos nas simulações.

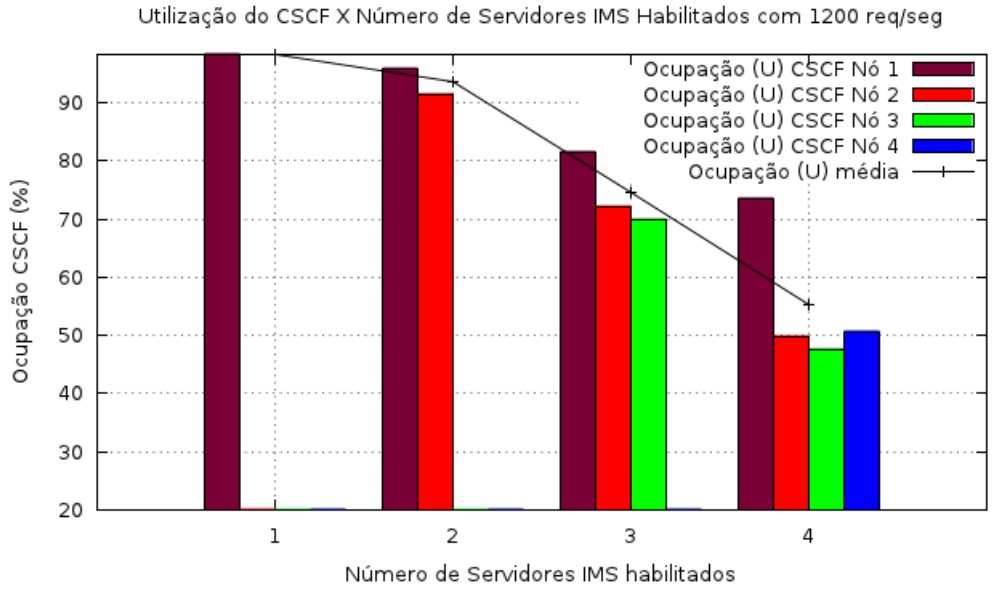

Figura 25: Simulação 01 - Taxa de utilização do CSCF por Servidor IMS Habilitado

Tabela 11: Resultados da Simulação 02 (a)

$\lambda(req/seg)$	U_{CSCF_1}	U_{CSCF_2}	U_{CSCF_3}	U_{CSCF_4}	Rr	Rs
100	11,43%	25,71%	20,00%	21,43%	0,34s	0,93s
200	20,00%	51,43%	37,14%	40,00%	0,34s	0,93s
400	58.73%	89,44%	60,41%	69,29%	0,72s	1,90s
520	78,57%	92,86%	82,86%	85,71%	1,82s	4,48s
676	81,43%	92,83%	87,14%	88,57%	2,86s	6,33s
879	81,43%	91,43%	85,71%	87,14%	3,77s	7,61s
1142	82,86%	93,57%	87,43%	88,14%	4,50s	8,55s

Tabela 12: Resultados da Simulação 02 (b)

$\lambda(req/seg)$	$E[w]_{DTS}$	$E[n_q]_{DTS}$	$V(req/seg)$	$\psi(V)$
100	36ms	10	308	3,1
200	36ms	19	601	3,0
400	130ms	688	1157	2,9
520	469ms	3411	1408	2,7
676	756ms	5665	1479	2,2
879	926ms	6766	1488	1,7
1142	983ms	7255	1453	1,3

Como pode-se observar na Figura 26, os tempos de resposta das requisições de registro e estabelecimento de sessão permanecem estáveis até uma taxa de chegada de 400 req/seg (1.440.000 req/hora), um valor extremamente alto comparado aos resultados obtidos em (LUO et al., 2009).

Em complemento, o gráfico da Figura 27 mostra o intervalo de confiança da me-

Tabela 13: Resultados da Simulação 02 (c)

$\lambda(req/seg)$	$E[w]_{LTS_1}$	$E[w]_{LTS_2}$	$E[w]_{LTS_3}$	$E[w]_{LTS_4}$
100	20ms	19ms	19ms	20ms
200	20ms	20ms	20ms	20ms
400	22ms	21ms	23ms	22ms
520	23ms	25ms	24ms	24ms
676	27ms	26ms	26ms	27ms
879	67ms	233ms	72ms	174ms
1142	148ms	566ms	204ms	428ms

Tabela 14: Resultados da Simulação 02 (d)

$\lambda(req/seg)$	$E[n_q]_{LTS_1}$	$E[n_q]_{LTS_2}$	$E[n_q]_{LTS_3}$	$E[n_q]_{LTS_4}$
100	3	11	7	9
200	6	22	14	17
400	11	42	24	32
520	14	55	30	40
676	16	59	33	45
879	30	312	70	191
1142	56	766	173	443

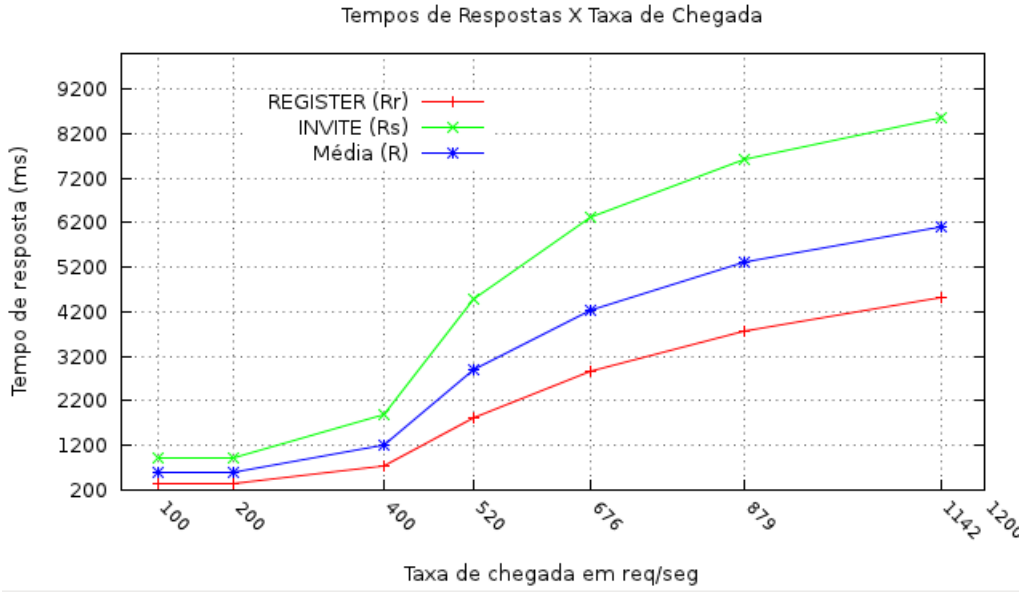

Figura 26: Simulação 02 -Tempos de resposta por taxa de chegada

dida obtida nos tempos de resposta, considerando 95% das requisições.

Os gráficos das Figuras 28 e 29 mostram a média de tuplas e o tempo de espera nos LTSs e no DTS durante a simulação. Assim como o sistema se mantém estável até uma taxa de chegada de 400 req/seg, o mesmo comportamento pode ser observado em relação à média de tuplas no espaço e ao tempo de espera no espaço de tuplas para uma

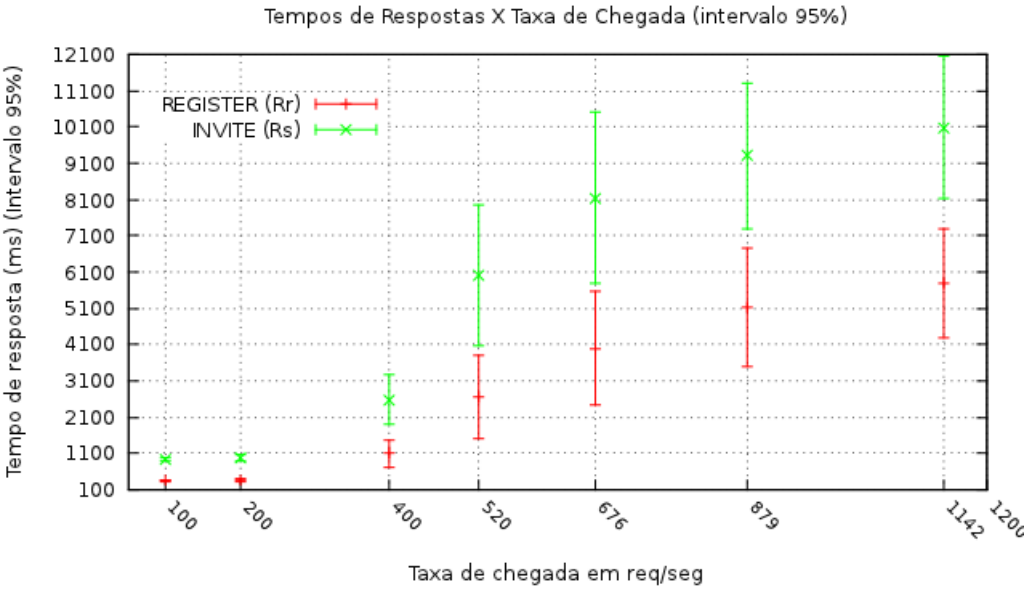

Figura 27: Simulação 02 - Tempos de resposta por taxa de chegada (95%)

requisição ser processada. Uma observação importante é que o aumento do tempo de espera e da média de tuplas é maior no DTS. Esse comportamento ocorre em função do balanceamento de carga realizado pelo *Events Processor*.

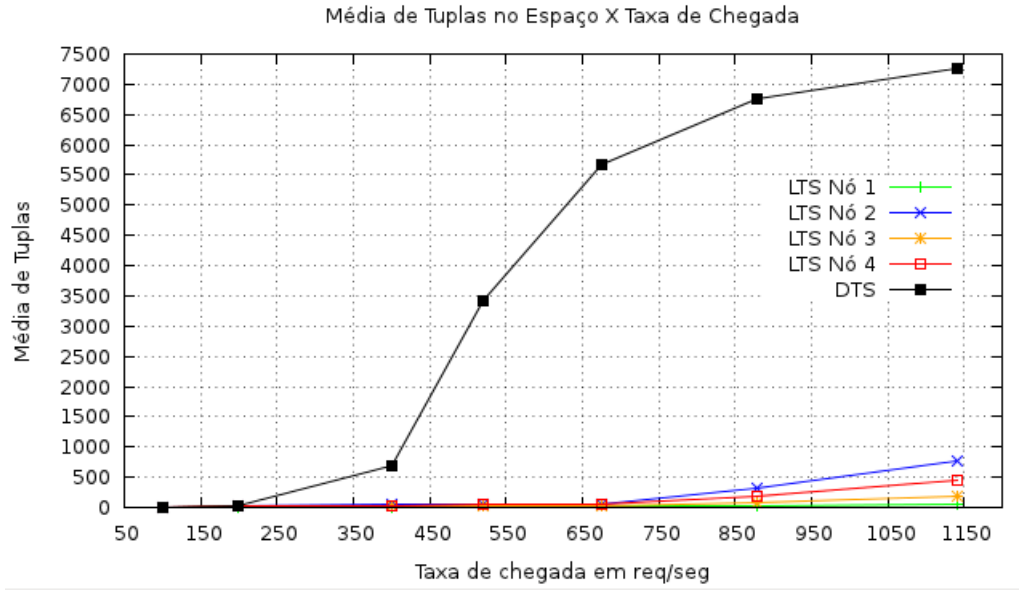

Figura 28: Simulação 02 - Média de tupla no espaço por taxa de chegada

Comparando esse resultados com os resultados da simulação anterior apresentados nos gráficos das Figuras 23 e 24, é possível afirmar que aumentando o número de Servidores IMS na rede a carga do DTS será distribuída, reduzindo o tempo de espera e, consequentemente, reduzindo os tempos de resposta.

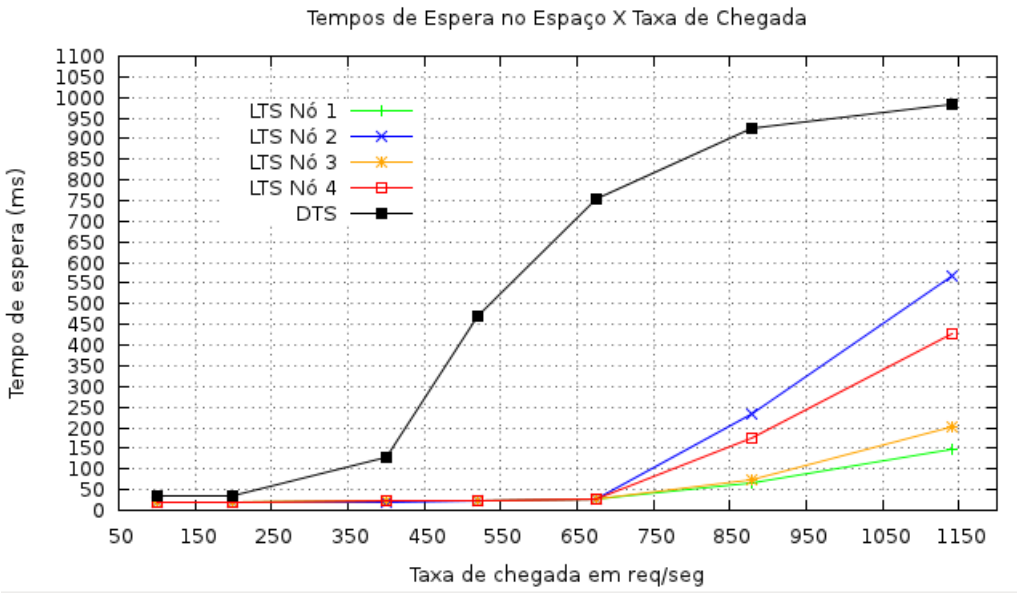

Figura 29: Simulação 02 - Tempo de espera no espaço por taxa de chegada

O gráfico da Figura 30 mostra a vazão do sistema à medida em que se aumenta a taxa de chegada. Pode-se observar que a vazão do sistema aumenta até uma carga de aproximadamente 500 req/seg.

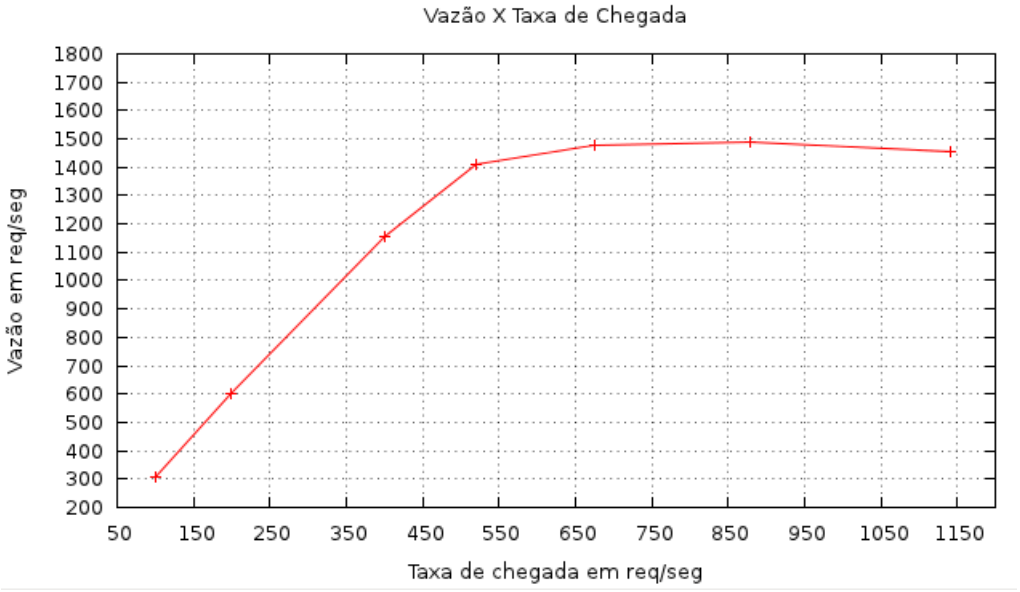

Figura 30: Simulação 02 - Vazão por taxa de chegada

O gráfico da Figura 31 mostra a escalabilidade com base na vazão do sistema, conforme definido na equação (6.10). Essa abordagem considera o sistema escalável enquanto o índice se mantiver estável em cada cenário de execução. Como pode-se observar, até uma carga de aproximadamente 600 req/seg, o índice de escalabilidade

se mantém em 3, ou seja, a vazão do sistema aumentou na mesma proporção que a taxa de chegada.

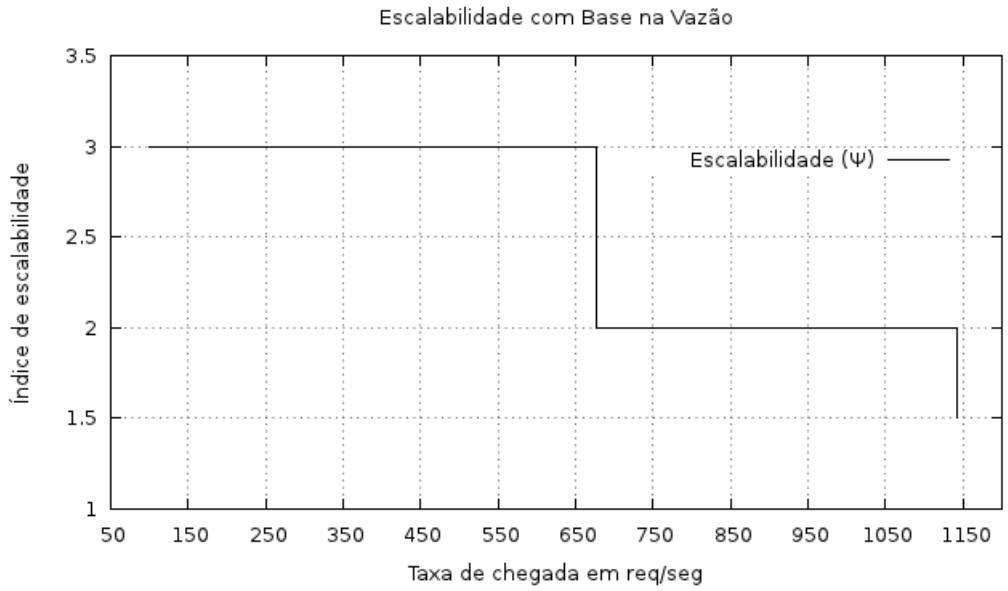

Figura 31: Simulação 02 - Índice de escalabilidade com base na vazão

Com as métricas de vazão, escalabilidade e comportamento do espaço de tuplas, podemos concluir que o sistema é escalável. Porém para cargas acima de 600 req/seg seria necessário adicionar Servidores IMS, ampliando a capacidade de processamento da rede.

Finalizando a Simulação 02 , pode-ser observar no gráfico da Figura 32 a distribuição da taxa de carga entre todos Servidores IMS da rede.

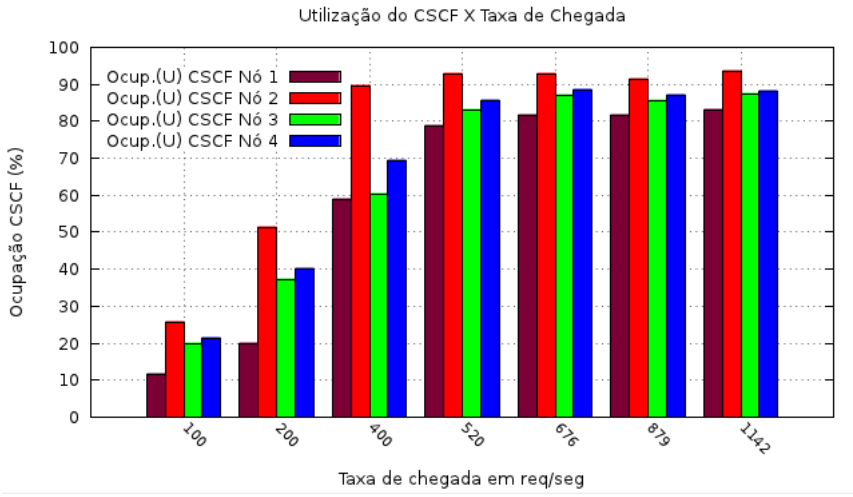

Figura 32: Simulação 02 - Taxa de carga do CSCF

6.2.3.3 Simulação 03 - Comparação com arquitetura padrão

O terceiro cenário de simulação teve como objetivo comparar os tempos de respostas e a vazão para as operações de registro e estabelecimento de sessão, e a taxa de carga nos servidores CSCF entre as duas arquiteturas. As tabelas 15 e 16 apresentam um resumo dos resultados obtidos nas simulações, e os gráficos das Figuras 33 e 34 apresentam as comparações de tempos de resposta e vazão, e carga dos servidores CSCF respectivamente.

Tabela 15: Resultados da Simulação 03 (Arquitetura Padrão)

$\lambda(req/seg)$	U_{I-CSCF}	U_{P-CSCF}	U_{S-CSCF}	Rr	Rs	V
200	45,60%	24,30%	67,34%	0,30s	0,52s	374
520	88,40%	87,85%	88,93%	5,49s	7,54s	486
676	89,50%	89,80%	90,53%	6,94s	9,32s	489
1142	90,70%	90,30%	91,00%	8,67s	11,34s	493

Tabela 16: Resultados da Simulação 03 (Espaço de Tuplas)

$\lambda(req/seg)$	U_{CSCF_1}	U_{CSCF_2}	U_{CSCF_3}	Rr	Rs	$V(req/seg)$
200	60,71%	17,43%	17,00%	0,40s	1,01s	390
520	92,42%	71,43%	68,57%	0,46s	1,16s	944
676	94,00%	79,86%	77,86%	2,25s	5,08s	1022
1142	95,29%	77,57%	77,71%	5,50s	10,45s	1024

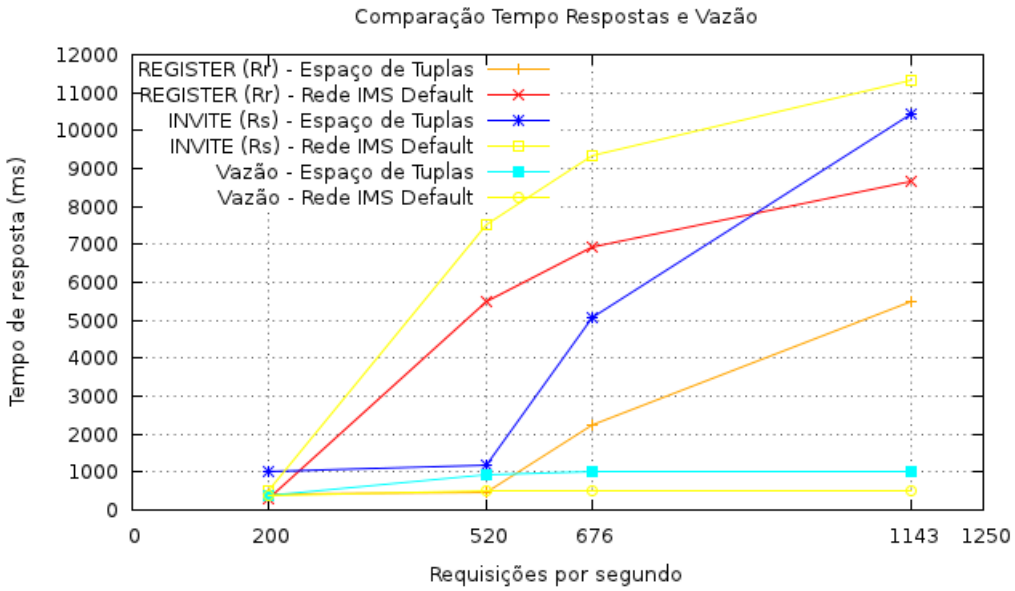

Figura 33: Simulação 03 - Comparação entre os tempos de resposta e vazão

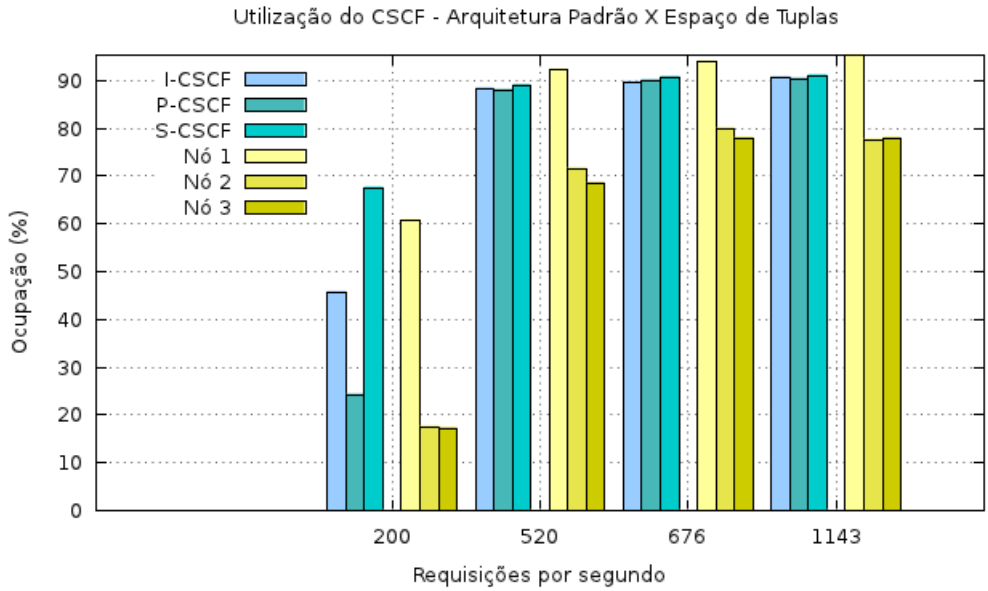

Figura 34: Simulação 03 - Comparação ente as taxas de carga nos servidores CSCF

Na Figura 33 observa-se uma diferença significativa entre a arquitetura IMS padrão e a arquitetura baseada em espaço de tuplas, sendo que a arquitetura baseada em espaço de tuplas suporta uma carga maior de requisições. Para ter uma comparação mais precisa, o cenário de simulação considerou apenas três Servidores IMS habilitados no modelo CPN da arquitetura baseada em espaço de tuplas, uma vez que na arquitetura padrão foram modelados três servidores, o P-CSCF, o I-CSCF e o S-CSCF. Importante lembrar que na arquitetura baseada em espaço de tuplas, não existe distinção entre P-CSCF, I-CSCF e S-CSCF, todos os Servidores IMS pode se comportar como qualquer um dos três.

O gráfico da Figura 34 apresenta algumas informações importantes. Primeiramente, observando o resultado para uma taxa de chegada de 200 req/seg verifica-se uma alta taxa de utilização no S-CSCF, enquanto o P-CSCF e o I-CSCF estão sendo subutilizado.

Esse comportamento também foi identificado em (LUO et al., 2009), conforme pode ser observado na tabela 17.

Tabela 17: Resultados obtidos em (LUO et al., 2009)

$\lambda(req/seg)$	U_{I-CSCF}	U_{P-CSCF}	U_{S-CSCF}
15	4,60%	31,20%	34,10%
40	13,30%	83,40%	91,90%

É possível notar que existe uma alta taxa de utilização no componente S-CSCF

e uma subutilização no I-CSCF. Nesse aspecto, tanto a simulação realizada por (LUO et al., 2009) com QPN, quanto a simulação da arquitetura IMS padrão realizada neste trabalho com CPN, demonstram que o ponto de gargalo da arquitetura IMS padrão é o S-CSCF.

Para a arquitetura baseada em espaço de tuplas observa-se no gráfico da Figura 34 uma alta taxa de utilização no Servidor IMS 1, enquanto os Servidores IMS 2 e 3 apresentam taxas de utilização baixas. Esse comportamento é esperado, uma vez que as requisições no cenário de simulação utilizado chegam somente pelo Servidor IMS 1, e, neste caso, esse terá prioridade no atendimento.

Para os demais cenários, aumentando a taxa de chegada de requisições, observa-se que a rede com arquitetura IMS padrão está totalmente sobrecarregada, enquanto a rede baseada em espaço de tuplas apresenta uma folga nos Servidores IMS 2 e 3, com o Servidor IMS 1 priorizando o atendimento das requisições e distribuindo-as igualmente entre todos os Servidores IMS da rede.

Com os resultados comparativos, nota-se que o fato de a arquitetura baseada em espaço de tuplas considerar que todos os Servidores IMS podem atuar como P-CSCF, I-CSCF ou S-CSCF traz um ganho significativo de desempenho, pois o processamento é distribuído entre todos, sem haver subutilização de recurso em determinados servidores enquanto ocorre uma sobrecarga em outros.

6.2.4 Considerações

A simulação da arquitetura baseada em espaço de tuplas demonstrou que existe ganhos significativos de desempenho e escalabilidade em relação à arquitetura padrão. Isso ocorre em função do desacoplamento entre processos oferecido pelo modelo de programação baseado em espaço de tuplas, e pelo fato de os Servidores IMS hospedarem todos os elementos responsáveis pelo controle de sessão, o que possibilita um melhor aproveitamento dos recursos computacionais.

Um dos principais comportamentos observados é a habilidade do componente *Events Processor* realizar o balanceamento de carga de requisições, sem a adição de software específico para essa finalidade. Esse comportamento demonstra a eficácia da proposta de manter um espaço de tuplas distribuído, possibilitando a inclusão de novos Servidores IMS na rede sem que o *Events Processor* tome conhecimento.

Com a comparação entre as duas arquiteturas, observa-se que até uma taxa de carga de 200 req/seg (720.000 por/hora) a arquitetura padrão oferece melhores resultados; o tempo médio de resposta é de 400ms para a arquitetura padrão, enquanto para a arquitetura baseada em espaço de tuplas o tempo médio é de 650ms. Essa diferença ocorre em função do *overhead* de comunicação com o espaço de tupla e da rede P2P. No entanto, a arquitetura baseada em espaço de tuplas escala melhor, e apresenta melhores resultados em cenários com alta taxa de chegada de requisições, alcançando os objetivos definidos inicialmente que considera utilização em cenários de larga escala.

7 CONCLUSÕES

7.1 Considerações Finais e Principais Contribuições

Este trabalho apresentou uma arquitetura baseada em espaço de tuplas para redes IMS, tendo como principais objetivos desempenho e escalabilidade nos componentes responsáveis pelo gerenciamento de sessão.

O trabalho apresentou uma arquitetura de rede e componentes de software, passíveis de implementação, e uma modelagem formal utilizando redes de petri coloridas. Através da modelagem formal foram realizadas simulações para demonstrar os resultados em relação a desempenho e escalabilidade.

Considerando que não identificamos nenhum trabalho específico que trate de implementação de protocolo de rede utilizando o paradigma de espaço de tuplas, este trabalho trouxe as seguintes contribuições:

1. Definição de um componente denominado SIPSpace que utiliza a abordagem baseada em espaço de tuplas para implementação do protocolo SIP, uma abordagem inédita para implementação de protocolos de rede.

2. Definição de uma infra estrutura de rede baseada em espaço de tuplas sobre redes P2P para implantação dos componentes de gerenciamento de sessão da arquitetura IMS. Espaço de tuplas sobre redes P2P tem sido pouco explorados o que faz deste trabalho uma contribuição original.

3. Modelagem formal de redes IMS utilizando redes de petri coloridas. Alguns trabalhos analisados utilizam modelos de fila ou QPN, porém nenhum explora os detalhes de operação da rede IMS como os modelos CPN apresentados neste trabalho. Nesse sentido, também consideramos a modelagem formal através de CPN como uma contribuição original deste trabalho.

Como é possível observar através dos resultados obtidos com as simulações, a arquitetura baseada em espaço de tuplas apresenta ganhos significativos em termos de desempenho e escalabilidade. Adicionalmente, a utilização de redes P2P baseada em DHT torna a adição de novos Servidores IMS na rede uma tarefa mais simples, uma vez que um Servidor IMS hospeda todos os componentes necessários para execução funcional da rede IMS.

7.2 Limitações

Uma limitação encontrada para a conclusão dos resultados é a complexidade da rede para representar um número significativo de Servidores IMS através de redes de petri coloridas. CPN é uma ferramenta excepcional para simulação de sistemas distribuídos complexos, mas a estratégia de modelagem adotada neste trabalho fez com que o modelo consuma recursos significativos da máquina, tanto para renderização quanto para simulação.

Apesar dessa limitação, foi possível mostrar o comportamento da rede com quatro Servidores IMS e realizar a comparação com a arquitetura IMS padrão.

Uma estratégia alternativa para modelagem de futuros trabalhos na área utilizando CPN é particionar ainda mais o modelo através dos módulos e executar simulações em separado, aplicando os resultados em um modelo de nível mais alto sem os detalhamentos.

7.3 Trabalhos Futuros

A proposta inicial considerava a implementação (codificação) da arquitetura baseada em espaço de tuplas. No entanto, devido à dificuldade de validar e demonstrar resultados com base em implementação de protótipos funcionais, uma vez que demandaria recursos de hardware para compor a rede, adotou-se a estratégia de validação formal com base em modelagem e simulação através de redes de petri coloridas.

Nesse sentido, como primeiro trabalho futuro destaca-se a importância da continuidade da implementação da arquitetura para validação em cenários reais e/ou ambientes de teste. A implementação teria como objetivos analisar a compatibilidade da arquitetura com as implementações existentes, e analisar o impacto de uma potencial

transição das implementações existentes para a implementação baseada em espaço de tuplas.

Adicionalmente, é possível destacar três tópicos importantes parar aprimoramento do trabalho:

Espaço de tuplas sobre redes P2P. A arquitetura apresentada neste trabalho adotou como base a definição do espaço de tuplas sobre redes P2P baseada em DHT, para simplificar a inclusão de novos Servidores IMS na rede, uma vez que redes baseadas em DHT são alto organizáveis. No entanto, não foi foco do trabalho estudar em profundidade aspectos de implementação, alguns dos quais já foram sugeridos na sessão 5.1.4, e gerenciamento de tuplas referente à sincronização (operações bloqueantes), replicação (análise e melhoria do algoritmo proposto) e tempo de vida da tupla no espaço (time-out).

Comportamento da rede considerando aspectos de mobilidade. Um aspecto importante a ser analisado é a mobilidade dentro da própria rede. A proposta prevê que dispositivos estejam conectados aos Servidores IMS, preferencialmente o mais próximo. Com isso, a resposta é publicada no mesmo LTS a qual foi publicada a requisição. No entanto, é possível que o mesmo dispositivo passe por um período de desconexão e retome a conexão em outro Servidor IMS da rede antes da ocorrência de um time-out da resposta. Neste caso, o dispositivo deve receber a resposta mesmo estando conectado em outro Servidor IMS.

Refinamento da análise de escalabilidade. Uma das abordagens para a análise de escalabilidade foi a utilização da equação de produtividade. Para efeitos de simulação e demonstração do resultado, as funções de QoS $f_{QoS_p}(\sigma_1, .., \sigma_n)$ e custo CT foram simplificadas. Nesse caso, é interessante refinar a função de QoS para adequação a cenários reais considerando não apenas o tempo de resposta, mas também aspectos de qualidade da experiência do usuário. Outro aspecto importante é definir mais precisamente o custo considerando não apenas aspectos de hardware, mas de complexidade de gerenciamento do ambiente.

REFERÊNCIAS

ABHAYAWARDHANA, V.; BABBAGE, R. A traffic model for the ip multimedia subsystem (ims). In: *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. [S.l.: s.n.], 2007. p. 783 –787.

AGRAWAL, P. et al. Ip multimedia subsystems in 3gpp and 3gpp2: Overview and scalability issues. *Communications Magazine, IEEE*, v. 46, n. 1, p. 138 –145, january 2008.

BAI, Y.; YE, X.; MA, Y. Formal modelling and analysis of sip using coloured petri nets. In: *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. [S.l.: s.n.], 2011. p. 1–5.

BAILLY, M.; DEBEAU, E.; JESTIN, J.-F. P2P Architecture for Conversational Services. In: *10th International Conference on Intelligence in Next Generation Networks, ICIN 2006.* [S.l.]: ICIN, 2006. p. 13–18 Session 9.

BALLETTE, M.; LIOTTA, A.; RAMZY, S. M. Execution time prediction in DSM-based mobile grids. In: *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2005. p. 881–888. ISBN 0-7803-9074-1.

BARAKOVIC, S.; JEVTIC, D.; HUSIC, J. B. Modeling of Session Initiation Protocol Invite Transaction using Coloured Petri Nets. *World Academy of Science, Engineering and Technology*, v. 61, p. 386–393, 2012.

BRYAN, D. et al. *Concepts and Terminology for Peer to Peer SIP draft-ietf-p2psip-concepts-04*. 2011. http://tools.ietf.org/html/draft-ietf-p2psip-concepts-04.

BRYAN, D. A.; LOWEKAMP, B. B. Decentralizing SIP. *Queue*, ACM, New York, NY, USA, v. 5, n. 2, p. 34–41, 2007. ISSN 1542-7730.

CAMARILLO, G.; GARCÍA-MARTÍN, M.-A. *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds, 2.ed*. [S.l.]: John Wiley & Sons, 2004.

CAO, Y. et al. A novel paralleling session setup mechanims in ims. In: *Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on*. [S.l.: s.n.], 2011. p. 250 –254.

CARRIERO, N.; GELERNTER, D. Linda in Context. *Communications of the ACM*, v. 32, n. 4, April 1989.

CARRIERO, N.; GELERNTER, D.; MATTSON, T. G. Linda in Heterogeneous Computing Environments. In: *Workshop on Heterogeneous Processing*. [S.l.]: Computer Society, 1992. p. 43–46.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems (4th ed.): concepts and design*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2005.

FREEMAN, E.; HUPFER, S.; ARNOLD, K. *JavaSpaces Principles, Patterns, and Practice*. [S.l.]: Sun Microsystems, Inc., 1999.

GELERNTER, D. Generative Communication in Linda. *ACM Trans. Program. Lang. Syst.*, ACM, New York, NY, USA, v. 7, n. 1, p. 80–112, 1985. ISSN 0164-0925.

HANDLEY, M.; JACOBSON, V.; PERKINS, C. *SDP: Session Description Protocol*. 2006. IETF RFC 4566 - obsoletes RFC 2327 and RFC 3266. http://www.ietf.org/rfc/rfc4566.txt.

JENNINGS, C. et al. *A SIP Usage for RELOAD - draft-ietf-p2psip-sip-06*. 2011. http://tools.ietf.org/html/draft-ietf-p2psip-sip-06.

JENSEN, K.; KRISTENSEN, L. M. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. [S.l.]: Springer-Verlag, 2009.

JOGALEKAR, P.; WOODSIDE, M. Evaluating the scalability of distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, v. 11, n. 6, p. 589 –603, jun 2000.

KANG, H. J. et al. SIP-based VoIP traffic behavior profiling and its applications. In: *MineNet '07: Proceedings of the 3rd annual ACM workshop on Mining network data*. New York, NY, USA: ACM, 2007. p. 39–44. ISBN 978-1-59593-792-6.

KHLIFI, H.; GRÉGOIRE, J.-C. IMS Application Servers: Roles, Requirements, and Implementation Technologies. *IEEE Internet Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, n. 3, p. 40–51, 2008. ISSN 1089-7801.

KIZMAZ, S.; KIRCI, M. Verification of Session Initiation Protocol Using Timed Coloured Petri Net. *Int'l J. of Communications, Network and System Sciences*, v. 4, n. 3, p. 170–179, 2011.

LI, Z.; PARASHAR, M. Comet: A Scalable Coordination Space for Decentralized Distributed Environments. *Hot Topics in Peer-to-Peer Systems, International Workshop on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 104–112, 2005.

LUO, A. et al. Performance modeling and evaluation using queuing petri nets in ims. In: *Communications and Networking in China, 2009. ChinaCOM 2009. Fourth International Conference on*. [S.l.: s.n.], 2009. p. 1 –5.

MKWAWA, I. M.; KOUVATSOS, D. D. Performance Modelling and Evaluation of IP Multimedia Subsystems. In: *HET-NETs'08 : 5th International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks, February 18-20, Karlskrona, Sweden*. [S.l.: s.n.], 2008.

MURPHY, A. L.; PICCO, G. P.; ROMAN, G.-C. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 15, p. 279–328, July 2006. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/1151695.1151698>.

O'DOHERTY, P.; RANGANATHAN, M. *JSR 32: JAIN SIP API Specification*. 2003. Java Community Process. http://jcp.org/en/jsr/detail?id=32.

OH, D. et al. Design of Next Generation Mobile Convergence Service Business Model. In: *11th International Conference on Intelligence in Service Delivery Networks (ICIN'07)*. [S.l.]: ICIN, 2007. Home Page http://www.icin.biz/files/programmes/Session6A-3.pdf.

PANDEY, S. et al. Performance study of IMS signaling plane. In: *IMSAA'07: Proceedings of International Conference on IP Multimedia Subsystem Architecture and Applications*. [S.l.]: IEEE Computer Society, 2007. p. 1–5.

PASCOTTO, R. et al. Are Partnerships Real Opportunities for Telecommunication Operators to Increase Revenues? In: *12th International Conference on Intelligence in Service Delivery Networks (ICIN'08)*. [S.l.]: ICIN, 2008.

PEREA, R. M. *Internet Multimedia Communications Using SIP*. [S.l.]: Morgan Kaufmann, 2008.

PICCO, G. P.; BALZAROTTI, D.; COSTA, P. Lights: a lightweight, customizable tuple space supporting context-aware applications. In: *Proceedings of the 2005 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2005. (SAC '05), p. 413–419. ISBN 1-58113-964-0. Disponível em: <http://doi.acm.org/10.1145/1066677.1066775>.

POIKSELKA, M. et al. *The IMS: IP Multimedia Concepts and Services*. [S.l.]: John Wiley & Sons, 2006.

QADEER, M. et al. IMS Network Architecture. In: *International Conference on Future Computer and Communication, 2009.(ICFCC 2009)*. [S.l.: s.n.], 2009. p. 329–333.

RONDINI, R. A.; BRESSAN, G. On the Design of a Tuple Space-based Platform for IMS Networks. In: *2012 16th International Conference on Intelligence in Next Generation Networks (ICIN 2012)*. Berlin, Germany: [s.n.], 2012. p. 100–107.

ROSENBERG, J. et al. *SIP: Session Initiation Protocol*. 2002. IETF RFC 3261. http://www.ietf.org/rfc/rfc3261.txt.

SPARKS, R.; ZOURZOUVILLYS, T. *Correct Transaction Handling for 2xx Responses to Session Initiation Protocol (SIP) INVITE Requests*. 2010. IETF RFC 6026. http://www.ietf.org/rfc/rfc6026.txt.

STOICA, I. et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 11, n. 1, p. 17–32, fev. 2003. ISSN 1063-6692. Disponível em: <http://dx.doi.org/10.1109/TNET.2002.808407>.

[TS 22.228]. *Service Requirements for the Internet Protocol (IP) multimedia Core Network Subsystem; stage 1*. 2008. 3GPP Technical Specification. V 8.6.0.

[TS 23.218]. *IP Multimedia (IM) session handling; IM call model; Stage 2*. 2008. 3GPP Technical Specification. V 8.4.0.

[TS 23.221]. *Architectural Requirements*. 2011. 3GPP Technical Specification. V 8.8.0.

[TS 23.228]. *IP Multimedia Subsystem (IMS); Stage 2*. 2008. 3GPP Technical Specification. V 8.7.0.

[TS 24.229]. *IP Multimedia Call Control Protocol Based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); stage 3*. 2012. 3GPP Technical Specification. V 8.18.1.

VINEEL, G. C. Analysis of system performance for differentiated handling of SIP requests. In: *IMSAA'07: Proceedings of International Conference on IP Multimedia Subsystem Architecture and Applications*. [S.l.]: IEEE Computer Society, 2007. p. 1–4.

WADE, S. P. *An Investigation into the use of the Tuple Space paradigm in Mobile Computing Environments*. Tese (Doctor of Philosophy) — Computing Department - Lancaster University - England, 1999.

WALDO, J. *The Jini Specification Second Edition*. [S.l.]: Sun Microsystems, Inc., 2000.

WELLS, G. C.; CHALMERS, A. G.; CLAYTON, P. G. Linda Implementations in Java for Concurrent Systems. *Concurrent Computation: Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, UK, v. 16, n. 10, p. 1005–1022, 2004. ISSN 1532-0626.

APÊNDICE A – INTRODUÇÃO À MODELAGEM E SIMULAÇÃO COM REDES DE PETRI COLORIDAS

Uma rede CPN é uma classe de redes de petri com recursos para definição e manipulação de tipos de dados denominados "cores", possibilitando adicionar semântica aos modelos. Em termos de modelagem isso significa que as marcações podem representar estruturas de dados tal como em ambientes de programação. As cores de um modelo CPN são declaradas através de uma linguagem funcional chamada CPN-ML.

As seções a seguir apresentam uma introdução ao formalismo de uma rede CPN, um exemplo didático de modelagem, o simulador para análise de desempenho e o relatório de espaço de estado, visando auxiliar o leitor no entendimento da estratégia de modelagem utilizada neste trabalho.

A.1 Definição Formal

Formalmente, uma rede de petri colorida é uma tupla $CPN = (P, T, A, \sum, V, C, G, E, I)$, onde:

1. P um conjunto finito de **lugares**

2. T é um conjunto finito de **transições** T tal que $P \cap T = \emptyset$.

3. $A \subset P \times T \cup T \times P$ é um conjunto de **arcos** direcionados.

4. $\sum$ é um conjunto finito não-vazio de **cores**.

5.V é um conjunto finito de **variáveis tipadas** tal que $Type[v] \in \sum$ para todas as variáveis $v \in V$.

6.$C : P \rightarrow \sum$ é uma **função de cores** que associa uma **cor** a cada um dos **lugares**.

7.$G : T \rightarrow EXPRv$ é uma **função de guarda** que associa a cada $t \in T$ uma expressão tal que $Type[G(t)] = Bool$.

8.$E : A \rightarrow EXPRv$ é uma **função de anotação de arco** que associa uma expressão a cada $a \in A$ tal que $Type[E(a)] = C(p)ms$, onde p é o **lugar** conectado ao **arco** a.

9.$I : P \rightarrow EXPR\emptyset$ é uma **função de inicialização** que associa a cada $p \in P$ uma expressão tal que $Type[I(p)] = C(p)ms$.

Uma rede de petri colorida com temporização adiciona os seguintes conceitos :

- Uma marcação é uma função M que mapeia cada lugar $p \in P$ em um conjunto $M(p)$ de marcações tal que $M(p) \in C(p)_{MS}$ se p não é temporizado;

  $M(p) \in C(p)_{TMS}$ se p é temporizado;

- Uma **marcação temporizada** é um par (M, t^*), onde M é uma marcação e $t^* \in \mathbb{T}$ é o valor do relógio global do modelo.

- A **marcação temporizada inicial** é o par $(M_0, 0)$, onde M_0 é definido por $M_0(p) = I(p)\langle\rangle$ para todo $p \in P$.

A.2 Exemplo de Modelo CPN

O modelo apresentado na Figura 35 foi adotado para explicar as características e a notação utilizada em modelagem baseada em redes de petri coloridas. O modelo ilustra um cenário simples de atendimento ao cidadão para retirada de documentos tipo RG, CPF e CNH. Para ser atendido, a pessoa deve aguardar em uma fila de triagem, onde será identificado o tipo de serviço e então, se encaminhar para uma sala de espera. Para os serviços de RG e CPF, as pessoas serão encaminhadas para um grupo de atendentes, e o serviço de CNH para outro grupo de atendentes. A triagem conta com duas atendentes, o serviço de RG/CPF conta com três atendentes e o serviço de CNH conta com duas atendentes.

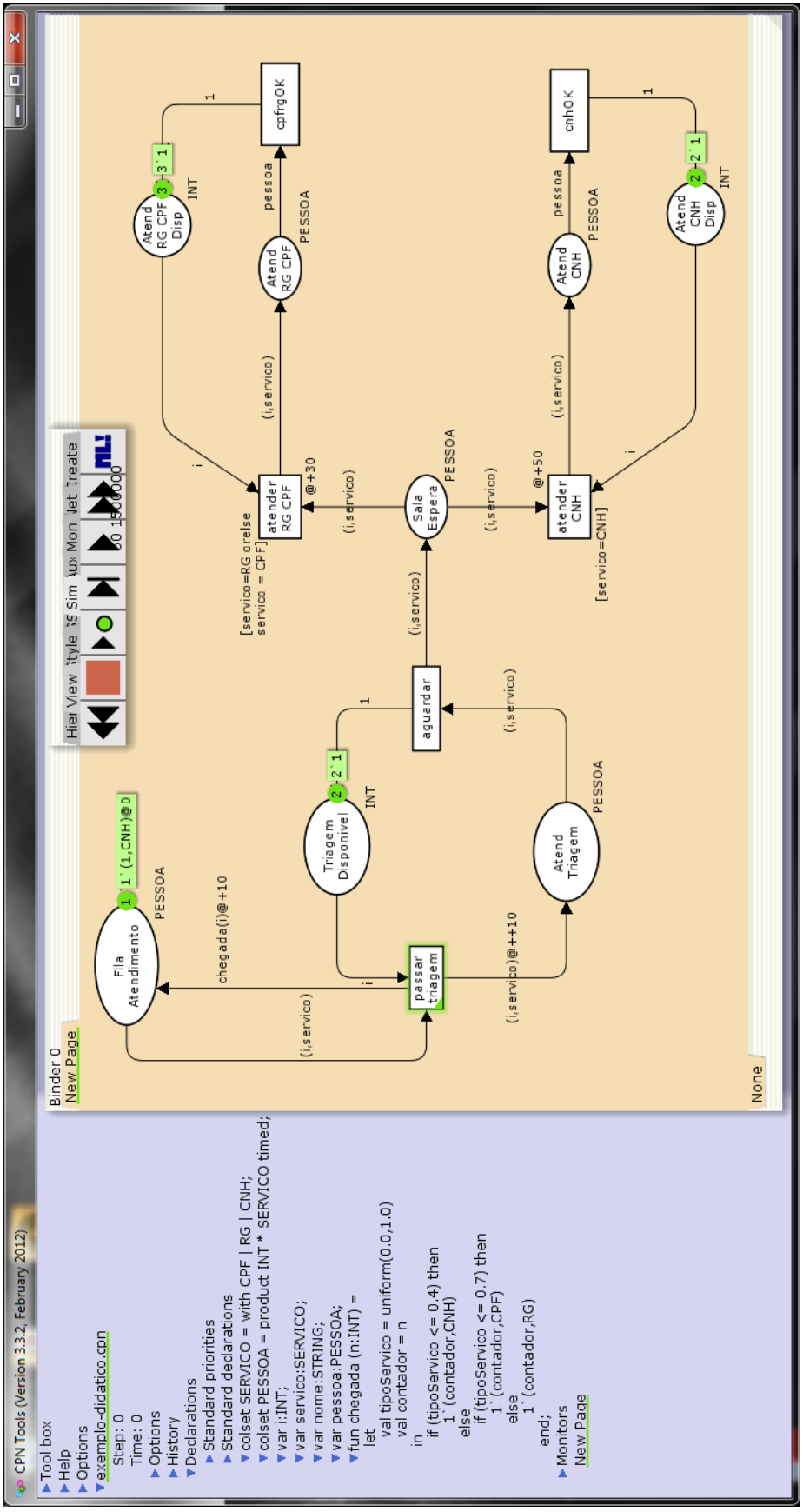

Figura 35: Modelo CPN Exemplo

A Figura 35 ilustra o ambiente CPN Tool com a área de modelagem, as declarações e as ferramentas de simulação.

Observando as declarações do modelo, é possível notar a existência das cores `SERVICO` e `PESSOA`. A cor `SERVICO` define os tipos de serviços prestados (CHN, RG e CPF) através da declaração `with`. Fazendo um paralelo a uma linguagem de programação, é o equivalente a utilizar enumerações, ou seja, os valores possíveis são previamente definidos. A cor `PESSOA` é o produto das cores INT e SERVICO. Novamente fazendo um paralelo a uma linguagem de programação, é o equivalente ás *structs* da linguagem C.

No modelo CPN é possível observar o lugar `Fila Atendimento`, com a indicação `PESSOA` no lado inferior direito. Isso significa que as marcações nesse lugar serão do tipo `PESSOA`. Adicionalmente, o lugar `Fila Atendimento` inicia com uma marcação onde o serviço a ser solicitado é o `CNH`. Observa-se isso pela inscrição `1‘(1,CNH)`.

Cada tipo de atendimento está representado no modelo por um lugar, são eles, `Atend Triagem`, `Atend RG CPF` e `Atend CNH`. Para modelar o número de atendentes, cada um dos lugares representando um tipo de atendente tem um outro lugar com o número de atendentes disponíveis, são eles, `Triagem Disponivel`, `Atend RG CPF Disp` e `Atend CNH Disp`. As marcações iniciais em cada um deles representam a quantidade de atendentes disponíveis.

A transição `atender RG CPF` possui a guarda `[servico=RG orelse servico=CPF]`, e a transição `atender CHN` possui a guarda `[servico=CNH]`. Isso indica que essas transições só serão habilitadas quando o valor da variável `servico` das marcações existentes no lugar `Sala Espera` obedecerem à condição declarada nas respectivas guardas. Adicionalmente, a transição `atender RG CPF` possui uma marcação de tempo `@+30`, e a transição `atender CHN` possui a marcação de tempo `@+50`. Esse tipo de inscrição na transição representa o tempo gasto para que uma pessoa seja atendida, 30 unidades de tempo para RG/CPF e 50 unidades de tempo para CNH. O mesmo pode ser observado na inscrição `(i,servico)@++10` do arco que conecta a transição `passar triagem` ao lugar `Atend Triagem`. Nesse caso, o atendimento na triagem demora 10 unidades de tempo.

Por fim, a inscrição `chegada(i)@+10` no arco que conecta a transição `passar triagem` ao lugar `Fila Atendimento` indica que a cada disparo dessa transição a função `chegada(i)` será executada. A função, por sua vez, em cada execução

gera uma marcação no lugar `Fila Atendimento` respeitando o tipo PESSOA, ou seja, uma marcação no formato `1‘(1,CNH)`, onde o valor CNH, RG ou CPF é gerado aleatoriamente obedecendo uma distribuição uniforme, observado no código CPN-ML da função pela linha `val tipoServico = uniform(0.0,1.0)`. A função `uniform(Real,Real)` é uma função pré-definida da linguagem CPN-ML, que no exemplo, irá gerar valores uniformemente distribuídos entre `0.0` e `1.0`.

Mesmo com um exemplo simples, é possível notar o potencial da ferramenta para simular sistemas complexos, pois combina as características do formalismo das redes de petri, com o poder de representatividade da linguagem funcional CPN-ML.

A.3 Simulação e Ferramentas de Análise

A.3.1 Espaço de Estado

Um espaço de estados é um grafo $G = (V, E)$, tal que V é o conjunto de vértices, representando o conjunto de marcações alcançáveis, e E é o conjunto de arestas rotuladas, representando o conjunto de transições.

A partir do grafo de espaço de estados, pode-se analisar diversas propriedades comportamentais, tais como a inexistência de *deadlock*, alcançabilidade de uma marcação específica, contorno limitado e marcação de origem. Adicionalmente, a ferramenta gera um grafo conectado - *Strongly-Connected-Component Graph* (SCC)

A.3.2 Análise de Desempenho

A análise de desempenho é conduzida através de sucessivas execuções de um modelo CPN temporizado. Para simular chegada de requisições (carga), normalmente utiliza-se um módulo específico onde uma função associada a um arco indica o intervalo entre cada requisição, conforme utilizado no modelo para a arquitetura baseada em espaço de tuplas, observado na Figura 14.

As demais marcações de tempo de um modelo podem indicar atrasos, tempos de processamento em recursos específicos, *timeout*, entre outros. A base de tempo utilizada pelo simulador é o chamado *Model Time*, ou seja, o tempo global do simular que inicia-se em `0` e é incrementado com base nas definições de tempo utilizadas nas inscrições dos arcos ou nas transições, conforme ilustrado no modelo exemplo.

As métricas de desempenho são obtidas através de coletas de dados durante as simulações. Em modelos CPN, a coleta de dados é realizada através de monitores denominados *Data Collection Monitors*, os quais podem ser definidos para cada medida de desempenho de interesse.

Um monitor inclui uma função de predicado (*predicate function*), que determina quando o dado deve ser coletado, e uma função de observação (*observation function*) que determina qual dado deve ser coletado.

As funções de monitoração são escritas em linguagem CPN-ML. No Apêndice B são apresentadas as funções de monitoração utilizadas para coleta de dados neste trabalho. Adicionalmente, a ferramenta de modelagem permite a utilização de monitores pré-definidos, sem a necessidade de escrever código CPN-ML.

APÊNDICE B – FUNÇÕES DE MONITORAÇÃO E COLETA DE DADOS

Conforme já mencionado anteriormente, o simulador CPN Tool permite a criação de monitores para coleta de dados de simulação utilizando-se "*Data Collector Monitoring Functions*", as quais apresentam um alto grau de flexibilidade, uma vez que as funções são escritas em código CPN-ML para atender necessidades específicas.

Este apêndice apresenta o código-fonte CPN-ML das funções de coleta de dados utilizadas para extração de resultados da simulação da arquitetura baseada em espaço de tuplas.

AvgTupleDTS, responsável pela coleta da média de tuplas no DTS.

```
fun pred (bindelem,
          P2P'DTS_1_mark : TUPLE tms,
          Server_Node1'B_1_mark : TUPLE tms,
          Server_Node2'D_1_mark : TUPLE tms,
          Server_Node3'E_1_mark : TUPLE tms,
          Server_Node4'H_1_mark : TUPLE tms) =
   let
      fun predBindElem (Server_Node1'in_DTS(1,
         {auth,finish,i,n,node,req,resp,sop,toa})) = true
       | predBindElem (Server_Node2'in_DTS (1,
         {auth,finish,i,n,node,req,resp,sop,toa})) = true
       | predBindElem (Server_Node3'in_DTS (1,
         {auth,finish,i,n,node,req,resp,sop,toa})) = true
```

```
            | predBindElem (Server_Node4'in_DTS (1,
              {auth,finish,i,n,node,req,resp,sop,toa})) = true
            | predBindElem _ = false
      in
         predBindElem bindelem
      end

   fun obs (bindelem,
            P2P'DTS_1_mark : TUPLE tms,
            Server_Node1'B_1_mark : TUPLE tms,
            Server_Node2'D_1_mark : TUPLE tms,
            Server_Node3'E_1_mark : TUPLE tms,
            Server_Node4'H_1_mark : TUPLE tms) =
      let
         val tDTS = (size P2P'DTS_1_mark)
         val tB = (size Server_Node1'B_1_mark)
         val tD = (size Server_Node2'D_1_mark)
         val tE = (size Server_Node3'E_1_mark)
         val tH = (size Server_Node4'H_1_mark)
         val total = tDTS + tB+tD+tE+tH+1
         fun obsBindElem (Server_Node1'in_DTS (1,
             {auth,finish,i,n,node,req,resp,sop,toa})) = total
           | obsBindElem (Server_Node2'in_DTS (1,
             {auth,finish,i,n,node,req,resp,sop,toa})) = total
           |obsBindElem (Server_Node3'in_DTS (1,
             {auth,finish,i,n,node,req,resp,sop,toa})) = total
           | obsBindElem (Server_Node4'in_DTS (1,
             {auth,finish,i,n,node,req,resp,sop,toa})) = total
           | obsBindElem _ = 0
      in
         obsBindElem bindelem
      end
```

WaitingTimeDTS, responsável pela coleta da média de tempo de permanência de tuplas no DTS.

```
fun pred (bindelem) =
   let
      fun predBindElem (Server_Node1'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) = true
        | predBindElem (Server_Node2'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) = true
        | predBindElem (Server_Node3'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) = true
        | predBindElem (Server_Node4'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) = true
        | predBindElem _ = false
   in
      predBindElem bindelem
   end

fun obs (bindelem) =
   let
      fun obsBindElem (Server_Node1'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) =
              Real.fromInt(IntTime() - sop)
        | obsBindElem (Server_Node2'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) =
              Real.fromInt(IntTime() - sop)
        | obsBindElem (Server_Node3'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) =
              Real.fromInt(IntTime() - sop)
        | obsBindElem (Server_Node4'in_DTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) =
              Real.fromInt(IntTime() - sop)
        | obsBindElem _ = 0.0
   in
         obsBindElem bindelem
   end
```

AvgTuplesLTS, responsável pela coleta da média tuplas no LTS.

```
fun pred (bindelem,
          LTS_Node1'LTS_Node1_1_mark : TUPLE tms,
          LTS_Node1'N1_C_1_mark : TUPLE tms,
          LTS_Node1'N1_E_1_mark : TUPLE tms) =
   let
      fun predBindElem (LTS_Node1'outC (1,
          {auth,finish,n,node,req,resp,sop,toa})) = true
        | predBindElem (LTS_Node1'outS (1,
          {auth,finish,n,node,req,resp,sop,toa})) = true
        | predBindElem _ = false
   in
      predBindElem bindelem
   end

fun obs (bindelem,
          LTS_Node1'LTS_Node1_1_mark : TUPLE tms,
          LTS_Node1'N1_C_1_mark : TUPLE tms,
          LTS_Node1'N1_E_1_mark : TUPLE tms) =
   let
      val tLTS = (size LTS_Node1'LTS_Node1_1_mark)
      val tN1C = (size LTS_Node1'N1_C_1_mark)
      val tN1E = (size LTS_Node1'N1_E_1_mark)
      val total = tLTS + tN1C + tN1E
      fun obsBindElem (LTS_Node1'outC (1,
          {auth,finish,n,node,req,resp,sop,toa})) = total
        | obsBindElem (LTS_Node1'outS (1,
          {auth,finish,n,node,req,resp,sop,toa})) = total
        | obsBindElem _ = 0
   in
      obsBindElem bindelem
   end
```

WaitingTimeLTS, responsável pela coleta do tempo médio de permanência de tuplas

no LTS.

```
fun pred (bindelem) =
   let
      fun predBindElem (Server_Node1'in_LTS (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) = true
        | predBindElem _ = false
   in
      predBindElem bindelem
   end

fun obs (bindelem) =
   let
      fun obsBindElem (Server_Node1'in_LTS (1,
       {auth,finish,i,n,node,req,resp,sop,toa})) =
            Real.fromInt(IntTime() - sop)
      | obsBindElem _ = 0.0
   in
      obsBindElem bindelem
   end
```

LoadCSCF, responsável pela coleta da ocupação média do componente CSCF.

```
fun pred (bindelem,
           Server_Node1'CSCF_1_mark : TUPLE tms,
           Server_Node1'INV_1_mark : TUPLE tms,
           Server_Node1'REG_1_mark : TUPLE tms) =
   let
      fun predBindElem (Server_Node1'Process_Tuple (1,
          {auth,finish,i,n,node,req,resp,sop,toa})) = true
        | predBindElem _ = false
   in
      predBindElem bindelem
   end
```

```
fun obs (bindelem,
          Server_Node1'CSCF_1_mark : TUPLE tms,
          Server_Node1'INV_1_mark : TUPLE tms,
          Server_Node1'REG_1_mark : TUPLE tms) =
   let
      val tcscf = (size Server_Node1'CSCF_1_mark)
      val tinv = (size Server_Node1'INV_1_mark)
      val treg = (size Server_Node1'REG_1_mark)
      fun obsBindElem (Server_Node1'Process_Tuple (1,
            {auth,finish,i,n,node,req,resp,sop,toa})) =
            tcscf + tinv + treg
        | obsBindElem _ = 0
   in
      obsBindElem bindelem
   end
```

SessionSetupDelay, responsável pela coleta do tempo médio gasto nas operações de início de sessão.

```
fun pred (Devices_Node1'Receive_Response (1,
   {auth,finish,n,node,req,resp,sop,toa})) =
        (resp=r200 andalso req=ACK)
   | pred _ = false

fun obs (Devices_Node1'Receive_Response (1,
   {auth,finish,n,node,req,resp,sop,toa})) =
     ModelTime.sub(time(),ModelTime.fromInt(toa))
   | obs _ = 0.0
```

RegistrationDelay, responsável pela coleta do tempo médio gasto nas operações de registro.

```
fun pred (Devices_Node1'Receive_Response (1,
   {auth,finish,n,node,req,resp,sop,toa})) =
```

```
      (resp=r200 andalso req=REGISTER)
   | pred _ = false

fun obs (Devices_Node1'Receive_Response (1,
   {auth,finish,n,node,req,resp,sop,toa})) =
     ModelTime.sub(time(),ModelTime.fromInt(toa))
   | obs _ = 0.0
```

APÊNDICE C – MODELO CPN PARA ARQUITETURA IMS PADRÃO

A Figura 36 apresenta o modelo CPN para a arquitetura IMS padrão, tradicionalmente utilizada para implantar redes IMS.

O modelo considera a existência de três servidores, cada um representando uma entidade funcional do CSCF, e a comunicação entre essas entidades. Importante observar que o modelo utilizou as mesmas funções de temporização empregadas no modelo CPN da arquitetura baseada em espaço de tuplas.

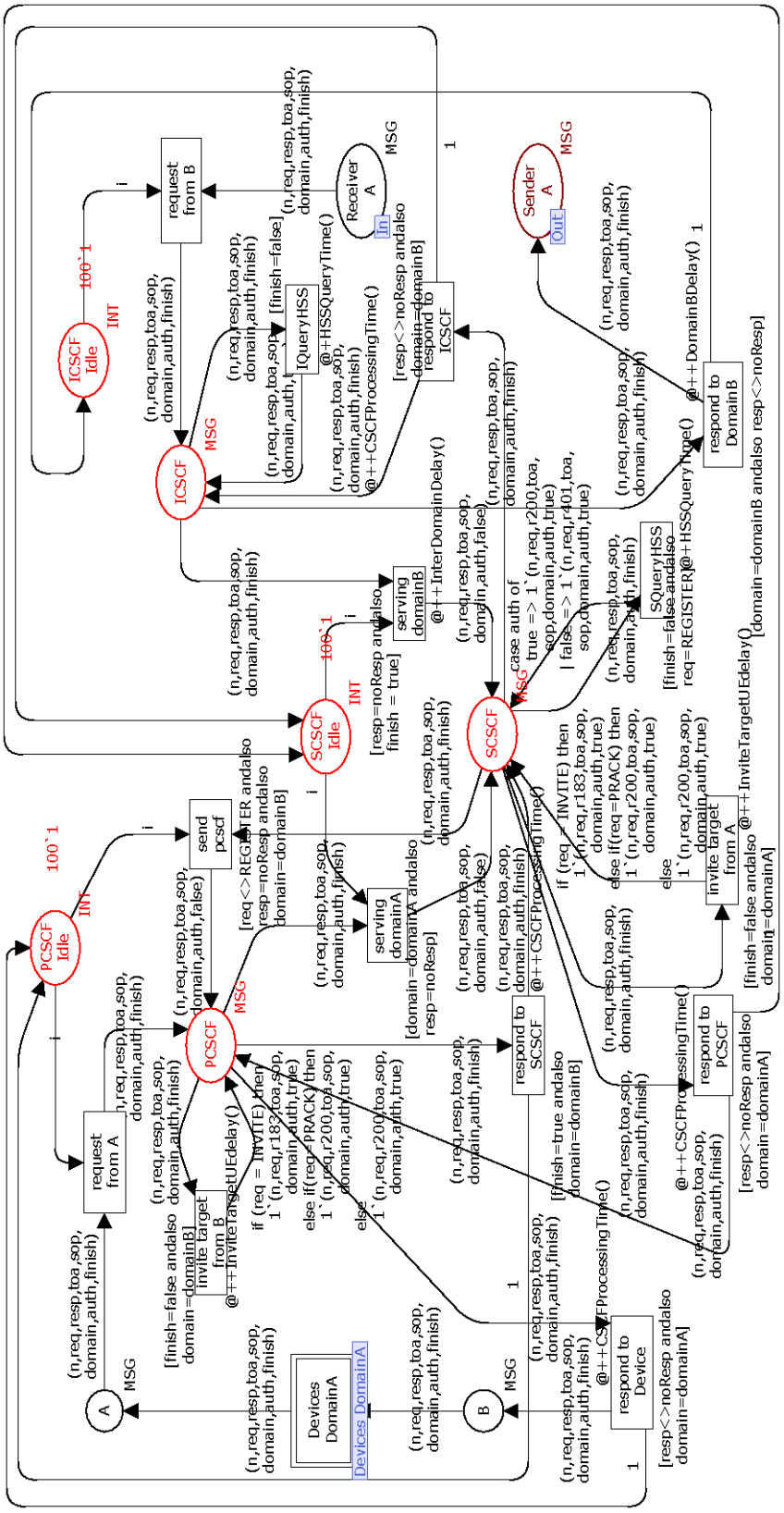

Figura 36: Modelo CPN para a Arquitetura IMS padrão