Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Migração de um simulador de protocolos para o ambiente Linux/x86 e um estudo de caso: especificação, verificação e simulação formais do protocolo RTSP

Autor:

Rodrigo Porto Avalle

Orientador:

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Examinador:

Prof. Jorge Lopes de Souza Leão, Dr. Ing.

Examinador:

Rodrigo de Carvalho Roman, M. Sc.

DEL

Abril de 2011

À minha família

Agradecimentos

Aos meus pais e minha família pelo carinho e eterna crença.

Ao professor Aloysio, meu orientador, pela generosidade, orientação e apoio constante que possibilitaram que este trabalho fosse concluído.

Ao professor Leão, pelas inesquecíveis aulas de Programação Concorrente e de Lógica para Computação e por abrilhantar a banca de defesa deste trabalho.

Ao estimado colega Roman, ex-companheiro de GTA, pelo apoio e motivação e, também, pela valiosa participação na banca examinadora.

Ao professor Casé, coordenador do DEL, pelas orientações e suporte nos trâmites burocráticos e gentileza em todos os contatos.

Aos professores Otto e Resende do GTA/UFRJ, e demais professores do curso de Engenharia Eletrônica com os quais obtive preciosos ensinamentos ao longo da graduação.

A todos os meus amigos que, de uma forma ou de outra, me fizeram chegar até aqui.

Por fim, dedico este trabalho ao povo brasileiro, que contribuiu de forma significativa à minha formação e estada nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

Resumo do Projeto Final apresentado ao DEL/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletrônico e de Computação

Migração de um simulador de protocolos para o ambiente Linux/x86 e um estudo de caso: especificação, verificação e simulação formais do protocolo RTSP

Rodrigo Porto Avalle

Abril de 2011

Orientador: Aloysio de Castro Pinto Pedroza

Departamento: Engenharia Eletrônica e de Computação

Neste trabalho, iremos estudar, em específico, um sistema de auxílio ao projeto de protocolos descritos em Estelle, o CAD-Estelle. Trata-se de um conjunto de ferramentas que contempla todas as etapas de desenvolvimento de um protocolo de comunicação, que foi desenvolvido no âmbito do Grupo de Teleinformática e Automação da UFRJ (GTA/UFRJ).

O primeiro objetivo deste trabalho enquadra-se num esforço de manter este conjunto de ferramentas atualizado em função da evolução das plataformas computacionais e, juntamente a isto, cooperar com projetos de melhor integração entre as diversas ferramentas. Portanto, a proposta é migrar o simulador de protocolos descritos em Estelle – a ferramenta SPDE – do ambiente SunOS/Sparc para o Linux/x86, garantindo seu perfeito funcionamento nesta nova plataforma.

Uma vez finalizada a migração do simulador de protocolos, dentro desse contexto de verificar e validar especificações e atestar a correção dos protocolos de comunicação, serão apresentados os resultados de um estudo de caso no qual examinamos todo o processo de desenvolvimento de protocolos suportado por ferramentas para métodos formais. Devido à sua relevância prática e à sua simplicidade, escolheu-se pela especificação em Redes de Petri e Estelle do protocolo RTSP como objeto desta avaliação.

Abstract of the Undergraduate Project presented to DEL/UFRJ as partial fulfillment of the requirements for the Bachelor's degree in Electronic Engineering

Porting a communication protocol simulator to Linux/x86 platform and a case study: formal specification, verification and simulation of RTSP protocol

Rodrigo Porto Avalle

April 2011

Supervisor: Aloysio de Castro Pinto Pedroza

Department: Computing and Electronic Engineering

In this work, we will study, more specifically, a computer-aided design system for protocols described in Estelle, the CAD-Estelle. It comprises a suite of tools which supports all design stages of a communication protocol and it has been developed in the GTA laboratory domain (GTA/UFRJ).

The first goal of this work is to join initiatives to keep this suite of tools updated due to the evolution of computing platforms and, therefore, to cooperate with projects which propose better integration between its various tools. The proposal is to migrate the Estelle protocol simulator – the SPDE tool – from SunOS/Sparc platform to Linux/x86, assuring its perfect operation in this target platform.

Once the protocol simulator porting is finished, aligned with this context of formal verification and validation of protocols specifications and assertion of communication protocols correction, the results of a case study will be presented where we explored the entire process of protocol design supported by suites of tools for formal methods. Due to its operational relevance and its simplicity, RTSP protocol was selected to be formally described with Petri Nets and Estelle specifications, subject of the evaluation of this work.

Palavras-Chave

Métodos Formais

Protocolos de Comunicação

Simulador

Estelle

RTSP

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O projeto de sistemas nos mais diferentes campos da engenharia conta com o suporte de metodologias já amplamente consolidadas e estabelecidas nos meios acadêmicos e da indústria. A validade e aplicabilidade das metodologias estão diretamente ligadas ao rigor e robustez das teorias que fundamentam o desenvolvimento destas.

O projeto de circuitos digitais, na área de engenharia eletrônica, por exemplo, faz uso de uma metodologia que se assenta na teoria de autômatos e na álgebra de Boole. O desenvolvimento de projetos em diversas outras áreas de aplicação de engenharia, tais como aeronáutica, civil, naval, entre outras, também fornecem exemplos análogos. O emprego de conceitos matemáticos é uma característica comum aos fundamentos teóricos clássicos que suportam muitas destas metodologias. A linguagem matemática, com o seu rigor e ausência de ambigüidades, permite conferir a uma teoria o formalismo necessário para que esta possa servir de suporte a todo um posterior desenvolvimento, desde metodologias formais até ferramentas de suporte a essas mesmas metodologias.

Considerando o cenário de constante e rápida evolução do mundo dos serviços de telecomunicações, em grande parte devido à contínua evolução dos recursos computacionais dos diversos componentes de um serviço de telecomunicações, a preocupação em se garantir a correção dos sistemas é crescente, refletindo-se em maiores esforços de investimentos em pesquisas de modo a aumentar a confiabilidade.

Um sistema de telecomunicações permite a comunicação entre entidades remotas. Ele é composto de dois subsistemas, chamados de rede de transporte e sistema de processamento. O sistema de processamento é o conjunto de recursos de computação e programas que controlam e gerenciam a rede de transporte; ou seja, implementa o software de comunicação. Este software é projetado de acordo com regras sintáticas e semânticas chamadas **protocolos.**

O desenvolvimento destes protocolos constitui a ***Engenharia de Protocolos***, que pode ser definida como uma especialização da engenharia de software para software de comunicação [7].

Os protocolos de comunicação estão, a cada dia, tornando-se mais complexos, proporcionalmente à funcionalidade a que se propõem e, conseqüentemente, o controle de qualidade do projeto também se torna mais complexo e difícil.

Essa preocupação reflete-se nos documentos oficiais que especificam os protocolos de comunicação. Por serem basicamente textuais, portanto ambíguos, não fornecem a necessária precisão lógica das funcionalidades do sistema, tornando difícil a análise do comportamento do sistema com a precisão necessária. Conseqüentemente, também há necessidade de emprego de linguagens ou métodos mais adequados para a especificação e documentação desses sistemas.

É precisamente essa a razão pela qual existem inúmeras ferramentas de software para apoio a projetos, algumas vezes reunidas em ambientes de desenvolvimento integrado, e que procuram implementar uma determinada metodologia, constituindo o que normalmente é referido como *computer aided design*.

A modelagem permite a verificação dos mecanismos básicos da comunicação, sendo as Redes de Petri um exemplo de notação amplamente usada na área de engenharia de protocolos, em função de seu alto poder de expressividade na especificação de paralelismo, sincronização e causalidade. Além do mais, ela permite a verificação de propriedades relacionadas ao controle e sincronização.

Visando viabilizar o desenvolvimento de especificações formais de sistemas distribuídos e protocolos de comunicação, várias Técnicas de Descrição Formais (TDFs) foram propostas, dentre as quais destacamos a *Extended State Transition Language* (Estelle) padronizada pela ISO [8], baseada na teoria de autômatos. O uso das TDFs permite a representação e teste de mecanismos orientados à implementação para os quais a modelagem não seria apropriada. Também são amplamente usadas no campo da engenharia de protocolos.

Para dar suporte à utilização dessas TDFs, foram construídas várias ferramentas que auxiliam o projetista tanto na edição, verificação e simulação das especificações, quanto na implementação e teste desses sistemas a partir das especificações.

Neste trabalho, iremos estudar, em específico, um sistema de auxílio ao projeto de protocolos descritos em Estelle, o CAD-Estelle [1]. Trata-se de um conjunto de ferramentas que contempla todas as etapas de desenvolvimento de um protocolo de comunicação, que foi desenvolvido no âmbito do Grupo de Teleinformática e Automação da UFRJ (GTA/UFRJ).

O primeiro objetivo deste trabalho enquadra-se num esforço de manter o conjunto de ferramentas atualizado em função da evolução das plataformas computacionais e, juntamente a isto, cooperar com projetos de melhor integração entre as diversas ferramentas [4]. Portanto, a proposta é migrar o simulador de protocolos descritos em Estelle [2] – a ferramenta SPDE – do ambiente SunOS/Sparc para o Linux/x86, garantindo seu perfeito funcionamento nesta nova plataforma.

Uma vez finalizada a migração do simulador de protocolos, dentro desse contexto de verificar e validar especificações e atestar a correção dos protocolos de comunicação, serão apresentados os resultados de um estudo de caso no qual examinamos todo o processo de desenvolvimento de protocolos suportado por ferramentas para métodos formais. Devido à sua relevância prática e à sua simplicidade, escolheu-se pela especificação em Redes de Petri e Estelle do protocolo RTSP [27] como objeto desta avaliação.

Com relação à estrutura do trabalho, no capítulo 2, apresentamos uma metodologia estruturada para o projeto de serviços em ambientes distribuídos, destacando os conceitos e terminologia propostos pelo modelo de referência para interconexão de sistemas abertos padronizado pela ISO (RM-OSI [11]). No capítulo 3, fazemos uma análise sobre o uso de métodos formais para a especificação e verificação de sistemas, descrevendo em mais detalhes os conceitos das Redes de Petri e da linguagem Estelle. No capítulo 4, por sua vez, fazemos um breve estudo sobre os tipos de ferramentas existentes para TDFs e descrevemos as características das duas ferramentas utilizadas na sequência deste trabalho: o ARP e o CAD-Estelle. O capítulo 5 descreve a abordagem, o desenvolvimento e os resultados do processo de migração da ferramenta do simulador de protocolos (SPDE) para a plataforma Linux/x86. O estudo de caso, que consistiu na especificação, verificação e simulação formais do protocolo RTSP, a partir de uma descrição informal e aplicando todos os conceitos abordados ao longo do trabalho, é detalhado no capítulo 6. Por fim, as conclusões a respeito dos temas estudados e dos resultados obtidos são apresentadas.

Capítulo 2

Projeto de serviços para uma arquitetura de rede

2.1 Introdução

Da experiência obtida no projeto de redes, vários princípios surgiram, possibilitando que novos projetos fossem desenvolvidos de uma forma mais estruturada que os anteriores [5]. Dentre esses princípios se destaca a idéia de estruturar a rede como um conjunto de camadas hierárquicas, cada uma sendo construída utilizando as funções e serviços oferecidos pelas camadas inferiores.

O conceito de hierarquia pressupõe camadas funcionais com atividades diferentes, de responsabilidades diferentes realizadas e organizadas em uma determinada ordem. Os serviços são procedimentos que uma camada oferece para uma outra camada.

Cada camada, ou nível, deve ser pensada como um programa ou processo, implementado por hardware ou software, que se comunica com o processo correspondente na outra máquina. A comunicação de dados, pois, envolve diferentes tipos de entidades. A solução para heterogeneidade é o uso de convenções – os protocolos. As regras que governam a conversação de um nível N qualquer são chamadas de *protocolo* de nível N. É importante ressaltar, contudo, que os dados transferidos em uma comunicação de um nível específico não são enviados diretamente (horizontalmente) ao processo do mesmo nível em outra estação. Na realidade, eles "descem", verticalmente, através de cada nível adjacente da máquina transmissora até o nível 1 (que é o nível físico, onde ocorre a única comunicação horizontal entre máquinas), para depois "subir", verticalmente, através de cada nível adjacente da máquina receptora até o nível de destino. Os limites entre cada nível adjacente são chamados interfaces. As interfaces definem a descrição dos serviços providos por uma camada.

A arquitetura da rede é formada, portanto, por níveis, interfaces e protocolos. O objetivo é o de reduzir a complexidade dos projetos de aplicações em rede, dividindo a tarefa de comunicação em módulos. Cada módulo é implementado por um nível que oferece um conjunto de serviços ao nível superior, usando funções realizadas no próprio nível e serviços disponíveis nos níveis inferiores. Como já foi mencionado, um protocolo de nível N é um conjunto de regras e formatos (semântica e sintaxe), através dos quais informações ou dados

do nível N são trocados entre entidades do nível N, localizadas em sistemas distintos, com o intuito de realizar funções que implementam os serviços do nível N. Um ou mais protocolos podem ser definidos em um nível.

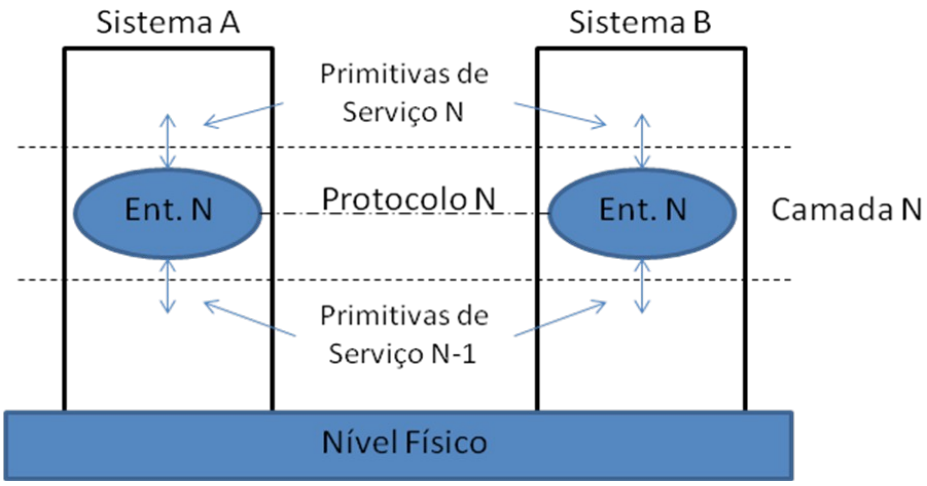


Figura 2.1 – Arquitetura de rede hierárquica

O projeto de protocolos em níveis é a maneira mais eficiente de se estruturar uma rede. Uma vez definida claramente a interface entre os diversos níveis, uma alteração na implementação de um nível pode ser realizada sem causar impacto na estrutura global [5].

2.2 O Modelo de Referência OSI da ISO – conceitos e terminologia

Para a descrição dos diversos serviços, protocolos e interfaces que compõem uma arquitetura de rede, vários métodos podem ser utilizados: de especificações informais através de textos e gráficos a técnicas formais de descrição (TDFs) - que iremos abordar com mais detalhes no capítulo seguinte. Inicialmente, vamos apresentar as convenções gráficas e de terminologia que são geralmente empregadas para a esquematização de um serviço, protocolo ou interface.

Dentro deste contexto de necessidade de padronização para o projeto de serviços em rede, foi criado pela ISO (*International Organisation for Standardisation*) o padrão internacional 7498, denominado ***Open Systems Interconnection***, que define um modelo de referência para a interconexão de sistemas abertos [11]. O modelo foi criado para suportar a troca de informações entre **processos de aplicação** ou **entidades de aplicação**.

Embora este Modelo (RM-OSI da ISO) não tenha se tornado preponderante e seus padrões sejam, às vezes, demasiado canônicos quando comparados a padrões utilizados pela indústria, sua arquitetura, serviços e protocolos serviram de referência para o desenvolvimento de diversos sistemas reais, atuando como autêntico catalisador de convergência. Sua importância tem sido, portanto, muito maior enquanto ferramenta do que como padrão de comunicação de redes.

A técnica básica de estruturação empregada no Modelo de Referência é a estratificação ou partição funcional em *camadas*. Cada sistema aberto é considerado, do ponto de vista lógico, um conjunto ordenado de S*ubsistemas*, representados por conveniência na seqüência vertical, segundo o conceito de camadas hierárquicas introduzido anteriormente.

A ISO definiu, então, uma terminologia para a descrição de serviços. Um ***serviço*** representa um conjunto de funcionalidades prestadas e oferecidas a um ***usuário*** por um ***fornecedor***.

Um ***usuário*** só pode acessar um ***serviço*** através de um ponto de ***acesso ao serviço*** (***SAP - service access point***). Os ***SAP***s da camada **N** são os locais onde os ***usuários*** da camada **N+1** podem acessar os ***serviços*** da camada **N**. Cada ***SAP*** tem um endereço único que o identifica.

As ***entidades*** de uma camada são elementos de *hardware* (por exemplo, uma placa de rede) ou *software* (processos) que atuam naquela camada. ***Entidades*** de uma mesma camada, mas que residem em máquinas diferentes são chamadas de ***entidades pares***.

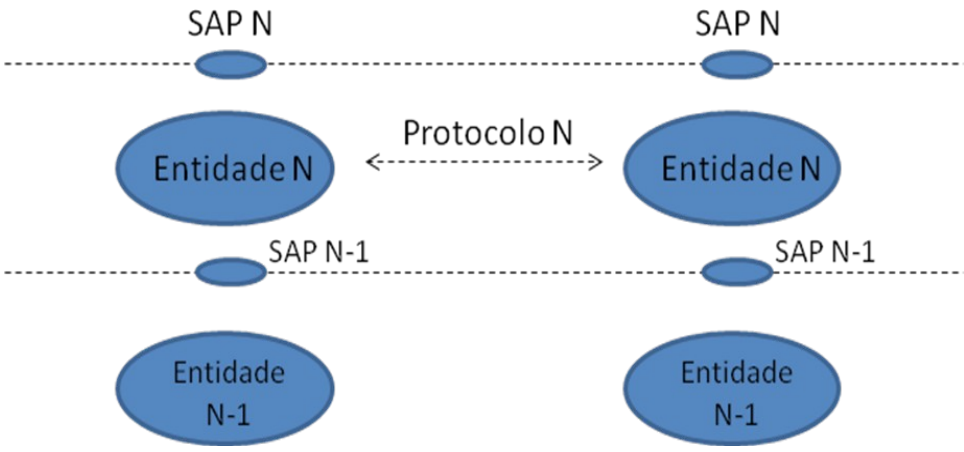


Figura 2.2 – Conceitos do RM-OSI

Um serviço é fornecido por uma camada (**N**) a outra camada (**N+1**) através das ***primitivas de serviço*** trocadas entre elas.

As primitivas são comandos e respectivas respostas correspondentes aos elementos de serviço solicitados entre camadas. As primitivas identificam a camada endereçada, e são designadas por um nome genérico e por um nome específico. O nome genérico especifica a ação a ser realizada pela camada endereçada e o nome específico define a direção do fluxo de primitivas.

Os nomes genéricos são estabelecidos em cada camada, já que as funções são características de cada uma delas, embora existam ações comuns a várias, como, por exemplo, conectar, transmitir dados, enviar reconhecimento, etc.

Os nomes específicos são padronizados independentemente das camadas, ou seja, são os mesmos para o Modelo OSI como um todo.

Basicamente, existem 4 tipos de primitivas:

- ***x***.REQUEST : enviada pelo usuário solicitante
- ***x***.INDICATION : entregue pelo fornecedor do serviço ao usuário que aceita o serviço
- ***x***.RESPONSE : invocada pelo usuário que aceita o serviço
- ***x***.CONFIRMATION : entregue ao solicitante pelo fornecedor do serviço

Onde ***x*** deve indicar o tipo de serviço que está sendo solicitado.

Nos serviços **confirmados**, é preciso haver um acordo entre o usuário **solicitante** e o usuário que aceita o serviço (**acolhedor**). Nesse caso, o usuário **acolhedor** pode aceitar ou rejeitar o serviço (**X**).

Nos serviços **não-confirmados** ou **sem confirmação**, não há necessidade de haver um acordo entre o usuário **solicitante** e o usuário **acolhedor**.

Existe, ainda, um tipo de serviço **iniciado pelo fornecedor**. Neste caso, o **fornecedor (ou provedor)** entrega a ambos usuários, a primitiva x.INDICATION em resposta a algum evento interno.

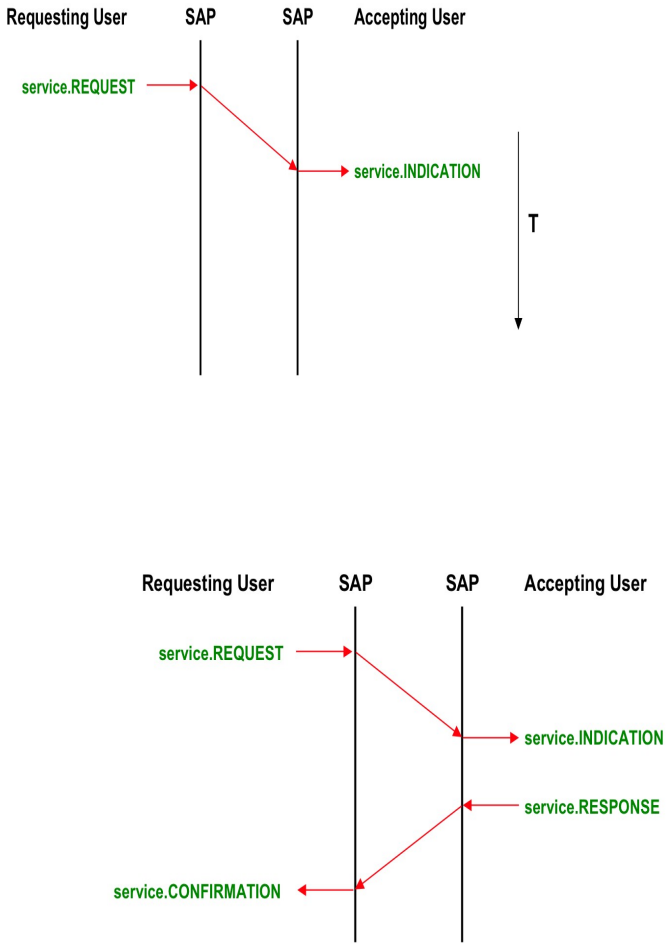


Figura 2.3 - Serviços sem confirmação e com confirmação

Existem normas para a composição de nomes de primitivas, conforme ilustrado na Figura 2.4:

Exemplo: T-CONNECT.request

Iniciais da camada	Nome do serviço	Tipo de primitiva
6 - P (Presentation) 5 - S (Session) 4 - T (Transport) 3 - N (Network) 2 - DL (Data link) 1 - PH (Physical)	CONNECT DISCONNECT RELEASE DATA EXPRESS_DATA TOKEN-GIVE . . . etc	request indication response confirmation

Figura 2.4 – Exemplo de composição de nome de primitiva

Para a transmissão de dados, o modelo OSI prevê que um serviço fornecido por uma camada pode ser orientado à conexão ou não-orientado à conexão. No modo de transmissão orientado à conexão o serviço pode ser dividido em três fases de operação: estabelecimento da conexão, quando os usuários e o fornecedor do serviço negociam parâmetros e opções que irão determinar como o serviço será utilizado; transferência de dados, quando os usuários do serviço trocam dados; e liberação da conexão, quando ocorre a desconexão, que pode ser feita de maneira ordenada (ambos os usuários concordam com o encerramento da conexão), abrupta (em resposta à solicitição de um dos usuários) ou abrupta pelo fornecedor do serviço.

No modo de transmissão não-orientado à conexão, uma única unidade de dados é transmitida do SAP de origem para um ou mais SAPs de destino sem que, para isso, seja estabelecida uma conexão entre eles. Toda a informação necessária para transmitir a unidade de dados (endereço, parâmetros de qualidade de serviço etc.) é passada para a camada que vai fornecer o serviço, junto com os dados a serem transmitidos.

A informação entre entidades de uma mesma camada ou entidades em camadas adjacentes é transferida em várias formas de *unidades de dados (Data Unit)*. A forma com que a unidade de dados é, em um sistema, passada de uma camada a outra é denominada *unidade de dados de serviço (Service Data Unit) - SDU*. A forma com que as unidades de dados são trocadas entre diferentes subsistemas é denominada *unidade de dados de protocolo (Protocol Data Unit)- PDU*. Desta forma, no Modelo OSI, cada prestação de serviço, correspondente ao tratamento das unidades de dados, em cada camada, acrescenta à unidade de dados um campo de controle de protocolo. No sentido inverso este campo é retirado em cada camada correspondente.

Resumindo:

- ao receber dados para efetuar um serviço, a camada N necessita incluir um cabeçalho, no qual são registradas informações relativas à camada. A esse cabeçalho, damos o nome de *Informação de Controle do Protocolo* - **PCI** (*Protocol Control Information*).
- aos dados recebidos pela camada N, damos o nome de *Unidade de dados do Serviço* - **SDU** (*Service Data Unit*).
- ao conjunto formado por PCI + SDU damos o nome de *Unidade de Dados do Protocolo* - **PDU** (*Protocol Data Unit). Portanto, **PDU = PCI + SDU***

2.3 Conclusões

Definida tal terminologia, temos uma padronização para a especificação de serviços. A partir de ilustrações, como o modelo de três camadas da arquitetura dos sistemas que interagem e diagramas de seqüência de tempo, e especificações textuais detalhando as regras, procedimentos e formatos para a troca de informações (protocolo de comunicação), é que se desenvolverá a implementação, seja em hardware ou software, das entidades do nível que oferece o serviço em questão.

Contudo, com a diversificação intensa das aplicações/serviços disponibilizados para ambientes distribuídos, tais sistemas de hardware e/ou software tendem a crescer em escala e funcionalidade. Em virtude deste acréscimo de complexidade, a probabilidade de erros sutis de projeto é muito maior. Além disso, alguns desses erros podem causar catastróficas perdas de tempo, dinheiro e, até mesmo, vidas humanas. Um grande objetivo, portanto, da engenharia de software é o de viabilizar o desenvolvimento de sistemas que operem de maneira confiável, não importando o quão complexos estes o sejam.

Um modo de se atingir este objetivo é através do uso de métodos formais, que são linguagens, técnicas e ferramentas baseadas em conceitos matemáticos, para especificar e verificar tais sistemas. O uso de métodos formais, a princípio, não garante a correção do sistema. Entretanto, eles podem aumentar consideravelmente nossa compreensão de um sistema revelando inconsistências, ambigüidades, e incompletudes que poderiam, em outra situação, não serem detectadas.

Capítulo 3

Métodos Formais

3.1 Introdução

Quando se iniciaram as pesquisas e estudos na área de métodos formais de descrição de hardware e software, a utilização destes, na prática, parecia bem distante e inviável. As notações propostas eram demasiado obscuras, as técnicas não escalavam, e o suporte de ferramentas era inadequado ou difícil demais de se operar. Havia poucos estudos de caso não--triviais e, juntos, eles não convenciam a comunidade de engenheiros da indústria de hardware e software. Além disto, poucas pessoas tinham sido treinadas para utilizá-los de maneira efetiva na indústria [13].

Somente recentemente, nas duas últimas décadas, que começou a se delinear um quadro mais promissor para o futuro dos métodos formais. Para a questão da especificação de software, a indústria tem se mostrado mais aberta para experimentar notações/linguagens para documentar as propriedades de um sistema mais rigorosamente. Ou ainda, para a verificação de sistemas implementados em hardware, a indústria tem adotado técnicas como a checagem de modelos ou a prova de teoremas para complementar a prática mais tradicional da simulação. Em ambas as áreas, pesquisadores e fabricantes têm realizado mais e mais estudos de caso de dimensões industriais e, assim, beneficiado-se com o emprego de métodos formais.

A palavra “método” pode ser definida como sendo um conjunto de técnicas aplicadas ao desenvolvimento de um processo. No nosso caso, queremos desenvolver sistemas (processos) que possam ser tratados em computadores. O desenvolvimento de sistemas computacionais pode ser dividido em várias fases: levantamento do problema, especificação do sistema, implementação (programação), testes, validação e manutenção.

Na fase de especificação do sistema, devemos nos preocupar em descrever o que (ou quais coisas) o produto final (software) deverá atender. Esta descrição pode ser totalmente informal, ou seja: usando uma base de descrição não muito bem definida (textos em linguagem comum e alguns tipos de diagramas) a qual, dificilmente, conduzirá a um desenvolvimento científico do sistema. Ou, pode ser estruturada (semi-formal) na qual, apesar de se utilizar de um conjunto de símbolos bem definidos, geralmente na forma de diagramas, a forma da descrição do processo tem um alto caráter subjetivo no significado das

propriedades do sistema (o entendimento pleno depende de quem fez e de como foi feito). Métodos consagrados na área de Engenharia de software, tais como Diagrama Entidade-Relacionamento, SQL e até mesmo a linguagem de modelagem UML, apresentam estas limitações: expressividade restrita (caso do DER e SQL) e suscetibilidade a ambigüidades e interpretações múltiplas (UML).

Por fim, a descrição do sistema pode ser feita formalmente. Os métodos formais definem matematicamente (estrutura e funcionalidade) os elementos da linguagem descritiva, os quais permitem o tratamento de uma classe substancial de problemas de forma concisa. Permitem, de forma mais natural, adotar uma abordagem modular no desenvolvimento da descrição do sistema (decomposição + composição + reutilização) e, devido ao tratamento matemático (portanto sem ambigüidades) que esta forma de especificar sistemas permite, é possível estabelecer mecanismos de interação com, por exemplo, linguagens de programação. Como a matemática é uma ciência exata, é possível submeter qualquer descrição formal (especificação) à prova, permitindo, assim, verificar a sua consistência de equivalência semântica em diferentes níveis de abstração (diferentes níveis de refinamentos do sistema).

Assim sendo, a especificação de um sistema através de um método formal deve apresentar as seguintes propriedades:

i. ser clara, concisa e sem ambigüidades;
ii. ser completa, sem omitir nenhum detalhe do sistema;
iii. ser consistente, ou seja, a especificação deve refletir com fidelidade todas as características do sistema. Especificações do mesmo sistema em diferentes níveis de abstração devem ser equivalentes;
iv. ser tratável, o que significa que a especificação pode ser submetida à análise para verificação e validação de suas propriedades;

O uso de métodos formais logo nas primeiras fases de desenvolvimento de um sistema resulta em uma boa estruturação do problema. Mesmo que o processo de desenvolvimento esteja já em curso, a utilização de formalismos pode identificar deficiências, que de outra forma não seriam descobertas, antes que estas causem problemas mais graves (i.e. numa fase posterior do processo de desenvolvimento). Embora tenham sido especificamente desenvolvidas para a área de telecomunicações (especificação de protocolos e processamento de dados), assiste-se atualmente a um interesse cada vez maior no uso de TDFs por parte de outras áreas como, por exemplo, robótica, segurança, finanças, medicina, sistemas produtivos, etc.

3.2 Especificação

Especificação é o processo de descrever um sistema e suas propriedades desejadas. A especificação formal se utiliza de linguagens com semântica e sintaxe definidas matematicamente.

Os tipos de propriedades descritas de um sistema podem incluir o comportamento funcional, o comportamento temporal, características de desempenho, ou a sua estrutura interna. Até então, a especificação de sistemas tem sido mais bem sucedida para propriedades comportamentais. Uma corrente atual é a de integrar diferentes linguagens de especificação, cada uma capaz de lidar com um aspecto diferente de um sistema. Uma outra corrente visa lidar com aspectos não-comportamentais de um sistema, como seu desempenho, restrições de tempo real, políticas de segurança e projeto de sua arquitetura.

Desta forma, um tipo de classificação usual para os métodos formais é o seguinte:

- orientado a modelos (denotacional) – descrevem o comportamento de um sistema a partir de estruturas matemáticas como tuplas, conjuntos, relações e funções;
- orientado a propriedades – definem o comportamento a partir de propriedades, normalmente na forma de axiomas, as quais o sistema deve satisfazer;
- orientado a comportamento – define um modelo a partir da possível sequência de estados, sendo usado na especificação de sistemas distribuídos, concorrentes e paralelos;
- híbridos.

Outra classificação análoga é apresentada em [14], na qual divide os métodos em:

- *history-based* – usa lógica temporal para se referir aos estados passados, presentes e futuros;
- *state-based* – caracterizam o estado de um sistema em um dado momento. Invariantes e pré e pós-condições restringem a aplicação de operações ao sistema;
- *transition-based* – representam as transições de um estado para o outro;
- *functional specification* – representam um sistema como uma coleção estruturada de funções;
- *operational specification* – coleção estruturada de processos.

Enfim, o processo de especificação consiste do ato de escrever coisas de maneira precisa, criteriosa. O principal benefício que advém de tal prática é o de se ganhar uma

compreensão mais aprofundada do sistema sendo especificado. É através do processo de especificação que desenvolvedores descobrem falhas de projeto, inconsistências, ambigüidades e incompletudes. Um produto resultante deste processo pode, por sua vez, ser formalmente analisado, seja através da verificação de que este é internamente consistente ou usando-o para derivar outras propriedades do sistema especificado. A especificação é uma útil ferramenta de comunicação entre o cliente e o projetista, entre o projetista e o implementador, e entre o implementador e a equipe de testes. Serve, ainda, como documentação para o código-fonte de um sistema, porém em um nível de descrição mais elevado.

3.3 Verificação

De acordo com a classificação dos métodos formais apresentada anteriormente, temos duas abordagens bem estabelecidas para a verificação de sistemas, que são a checagem de modelos, para métodos orientados a modelos e a comportamento; e a prova de teoremas, para métodos orientados a propriedades. Elas vão um passo além da especificação, já que estes métodos formais são usados para analisar um sistema quanto às suas propriedades desejadas.

3.3.1 Checagem de Modelos

A checagem de modelos é uma técnica que depende da definição de um modelo finito de um sistema e, então, checar se uma propriedade desejada é atendida neste modelo. Falando em termos gerais, o teste é realizado como uma busca exaustiva sobre o espaço de estados, cujo término é garantido já que o modelo é finito. O desafio, do ponto de vista técnico, na checagem de modelos é o de se desenvolver algoritmos e estruturas de dados que nos permitam trabalhar com grandes espaços de busca. A checagem de modelos tem sido usada, primariamente, para a verificação de hardware e protocolos. A corrente atual é a de se aplicar esta técnica, também, para analisar especificações de sistemas de software.

Duas abordagens gerais para a checagem de modelos são usadas na prática hoje. A primeira, a checagem de modelos temporal, é uma técnica que foi desenvolvida independentemente nos anos 80. Nesta abordagem, as especificações são expressas em uma lógica temporal e os sistemas são modelados como sistemas estado-transição finitos (métodos orientados a modelos, ou *history-based* e *transition-based*). Um procedimento eficiente de busca é usado para checar se um dado sistema estado-transição finito é, de fato, um modelo para a especificação.

Na segunda abordagem, a especificação é dada como um autômato (método orientado a comportamento, ou *operational specification*). O sistema, então, também modelado como um autômato é comparado com a especificação para determinar se o seu comportamento está em conformidade com aquele da especificação.

Em contraste com a prova de teoremas, a checagem de modelos é completamente automática e rápida, produzindo uma resposta, às vezes, em uma questão de minutos. A checagem de modelos pode ser usada para checar especificações parciais e, desta forma, fornecer informações úteis sobre a correção de um sistema mesmo se este não foi completamente especificado. Acima de tudo, um dos pontos mais fortes desta técnica é que ela produz contra-exemplos, que geralmente representam erros sutis no projeto, e, desta forma, pode ser empregada para auxiliar na depuração do sistema. A principal desvantagem da checagem de modelos, contudo, é o problema da explosão de estados, já tendo sido propostas diversas abordagens para aliviar tal problema.

3.3.2 Prova de Teoremas

A prova de teoremas é uma técnica onde, ambos o sistema e suas propriedades desejadas são expressas como fórmulas em alguma lógica matemática. Esta lógica é dada por um sistema formal, que define um conjunto de axiomas e um conjunto de regras de inferência; ou seja, aplica-se a métodos formais orientados a propriedades ou "*functional specifications*". A prova de teoremas é o processo de se achar uma prova de uma propriedade a partir dos axiomas do sistema. Os passos na prova se relacionam com os axiomas e regras, e possivelmente com definições derivadas e lemas intermediários. Enquanto que provas podem ser desenvolvidas manualmente, aqui, focalizamos somente em provas de teoremas assistidas por máquinas. Provadores de teoremas estão sendo utilizados cada vez mais hoje para a verificação mecânica de propriedades de projetos de hardware e software critícas quanto ao quesito segurança.

Provadores de teoremas podem ser classificados, a grosso modo, em um espectro que vai de programas de propósito geral altamente automatizados a sistemas interativos com capacidades de propósito específico. Os sistemas automatizados têm se mostrado úteis como procedimentos de busca geral e têm obtido notório sucesso em resolver vários problemas de combinatória. Os sistemas interativos têm sido mais adequados para o desenvolvimento formal sistemático de matemáticas e para a mecanização de métodos formais.

Em contraste com a checagem de modelos, a prova de teoremas pode lidar diretamente com espaços de estado infinitos. Provadores de teoremas interativos, por definição, requerem interação com um humano, de modo que o processo de prova de teoremas é lento e, freqüentemente, sujeito a erros. No processo de encontrar a prova, todavia, o usuário geralmente adquire uma percepção bastante valiosa a respeito do sistema ou da propriedade sendo provada.

A exemplo da checagem de modelos, um acréscimo no número e nos tipos de provadores de teoremas evidenciam um crescente interesse na técnica de prova de teoremas. Têm havido um crescimento correspondente, também, no número e nos tipos de exemplos aos quais têm se aplicado provadores de teoremas.

3.4 Métodos formais abordados no trabalho

O caminho até a proposição e a normatização das técnicas de descrição formal (TDF; ou FDT, em inglês) foi sendo construído a partir de décadas de pesquisas e trabalho em métodos rigorosos e linguagens de especificação. Provavelmente, ainda nos anos 70, a área de telecomunicações foi a primeira a manifestar a necessidade de recurso a estas técnicas formais. De fato, e de acordo com [18], "apenas abordagens formais para a especificação, verificação, análise, implementação, teste e operação de sistemas serão capazes de lidar com a crescente complexidade das normas para telecomunicações e para OSI". Além disso, é ainda aplicável, num contexto geral, o fato de as especificações/descrições convencionais de sistemas serem normalmente feitas em linguagem natural ou recorrendo a diagramas (ou a uma mistura de ambos), sendo por isso difíceis de analisar e propensas à ocorrência de ambigüidades [18].

A organização ISO criou um grupo de trabalho (*ISO FDT Group*) que apontou, em seus estudos iniciais, para o desenvolvimento de técnicas de descrição formal segundo duas linhas de arcabouços teóricos: autômatos de estados finitos e conceitos algébricos. A identificação destas duas linhas de desenvolvimento levou a ISO a criar e normalizar duas TDFs, uma para cada abordagem. Assim surgiram ESTELLE ("Extended Finite State Machine Language") [8], baseada na primeira abordagem, e LOTOS ("Language of Temporal Ordering Specification") [16], baseada na segunda abordagem.

Paralelamente ao trabalho desenvolvido pela ISO, o então CCITT (*International Consultative Committee on Telegraphy and Telephony*), atualmente ITU-T (*International*

Telecomunications Union – Telecommunications Standardisation Sector), criou e normalizou a terceira TDF. Assim surgiu o SDL (*Specification and Description Language*) [15], que tem a particularidade de incluir características de ESTELLE e LOTOS, especialmente o conceito de autômato de estados finitos utilizado na primeira, e os conceitos algébricos da segunda. Isto não significa que se possa concluir sumariamente que se trata de uma linguagem mais poderosa do que as outras.

As técnicas de descrição formal (TDFs) ESTELLE, LOTOS e SDL são, obviamente, apenas três das muitas linguagens a que as diversas arquiteturas e metodologias existentes podem recorrer para especificar ou descrever sistemas – no caso, as únicas que são normas internacionais. A aplicação destas linguagens para especificação de protocolos e serviços de comunicação à área de engenharia de serviços resultou em muitas extensões, por exemplo, para orientação a aplicações em tempo real e a objetos.

Para ilustrar os vários formalismos já propostos na literatura, uma compilação de outras linguagens e métodos formais extraídos de [17] é apresentada na tabela 3.1, onde estes são divididos de acordo com a categorização discutida na seção 3.2.

Baseado em autômatos (comportamento) e/ou lógica temporal (modelo denotacional)	Verificação baseada em métodos dedutivos (orietando a propriedades)	Modelo baseado em objetos
➢ CSL - "Control and Specification Language" ➢ Esterel ➢ Lustre ➢ Signal ➢ Statecharts ➢ TLT - "Temporal Language of Transitions"	➢ KIV - "Karlsruhe Interactive Verifier" ➢ Tatzelwurm ➢ HTTD - "Hierarchical Timed Transition Diagrams" ➢ RAISE - "Rigorous Approach to Indu"	➢ LCM («Language of Conceptual Modelling») ➢ Modula-3 ➢ TROLL-light

Tabela 3.1 – Exemplos de linguagens/métodos formais

3.4.1 Redes de Petri

Como parte da metodologia estudada neste trabalho para o desenvolvimento de protocolos e serviços de rede, iremos analisar a utilização de Redes de Petri como técnica de modelagem.

A grande vantagem das Redes de Petri (RdP) no tocante ao projeto de protocolos decorre da capacidade que estas possuem de descrever sistemas concorrentes. São uma ferramenta para **a modelagem e projeto de sistemas**, utilizando uma representação matemática do sistema, sendo uma extensão das **máquinas de estados finitos**.

Ao contrário das máquinas de estado, em que apenas um lugar pode estar com "uma ficha"- ou seja, com uma transição sensibilizada, pronta para ocorrer -, as Redes de Petri permitem que mais de um lugar esteja com ficha. Os lugares com ficha evoluem de forma independente e assíncrona. A sincronização pode ser realizada tomando-se o cuidado de modelar o comportamento do sistema de tal forma que se consiga o encadeamento temporal das transições que se deseja. Com esta característica garantida, é possível modelar duas máquinas de estado separadamente (iniciador e respondedor, cliente e servidor) e, posteriormente, sincronizá-las.

Como ferramentas matemáticas e gráficas, as RdP oferecem um ambiente uniforme para a modelagem, análise formal e simulação de sistemas a eventos discretos, permitindo uma visualização simultânea da sua estrutura e comportamento [19]. Mais especificamente, as RdP modelam dois aspectos desses sistemas, eventos e condições, bem como, as relações entre eles. Segundo esta caracterização, em cada estado do sistema verificam-se determinadas condições. Estas podem possibilitar a ocorrência de eventos que por sua vez podem ocasionar a mudança de estado do sistema. Como veremos, é possível relacionar, de uma forma intuitiva, condições e eventos com os dois tipos de nós da rede, respectivamente lugares e transições.

Os elementos dos dois conjuntos em que se podem dividir os nós constituintes de uma RdP denominam-se, respectivamente, *lugares* e *transições*. Os lugares são normalmente representados por circunferências ou elipses, e as transições por segmentos de reta, retângulos ou barras. Os lugares encontram-se ligados às transições, e estas aos lugares, através de arcos dirigidos.

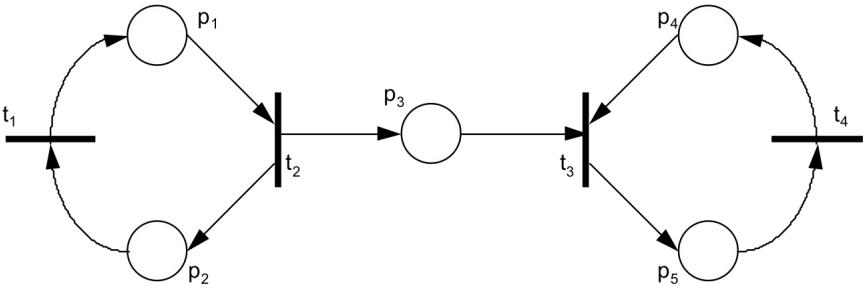


Figura 3.1 – Representação gráfica de uma Rede de Petri

Uma rede de Petri **C** é composta por quatro partes:

- um conjunto de **lugares P = {p1, p2, ..., pn}**;
- um conjunto de **transições T = {t1, t2, ..., tm}**;
- uma função de **entrada I : T → P∞** ;
- um conjunto de **saídas O : T → P∞**.

Então, **C = (P, T, I, O)**

A função de entrada **I** mapeia uma transição **tj** para uma coleção de lugares **I(tj)**, conhecida como **lugares de entrada de uma transição**.

A função de saída **O** mapeia uma transição **tj** para uma coleção de saídas **O(tj)**, conhecida como **lugares de saída de uma transição**.

Uma **marcação** μ é uma atribuição de **fichas** aos lugares de uma rede de Petri. Uma **ficha** é um conceito primitivo para redes de Petri, da mesma forma que lugares e transições. O número e posição das fichas pode mudar durante a execução de uma rede de Petri. Dessa forma, as fichas são usadas para definir a execução de uma rede de Petri.

A marcação μ pode ser definida como sendo um vetor:

$$\mu = (\mu 1, \mu 2, ..., \mu\ n) \mid n = |P| \text{ e } \mu i \in N,\ i=1, ..., n$$

O número de fichas no lugar **pi** é μ**i**, i = 1, ..., n. Assim sendo: μ**(pi) =** μi

Uma rede de Petri marcada pode ser definida como: **M = (P, T, I, O,** μ**)**

Num grafo de rede de Petri, fichas são representadas por pontos dentro dos círculos. No exemplo abaixo, temos:

$$\mu = (1, 2, 0, 0, 1)$$

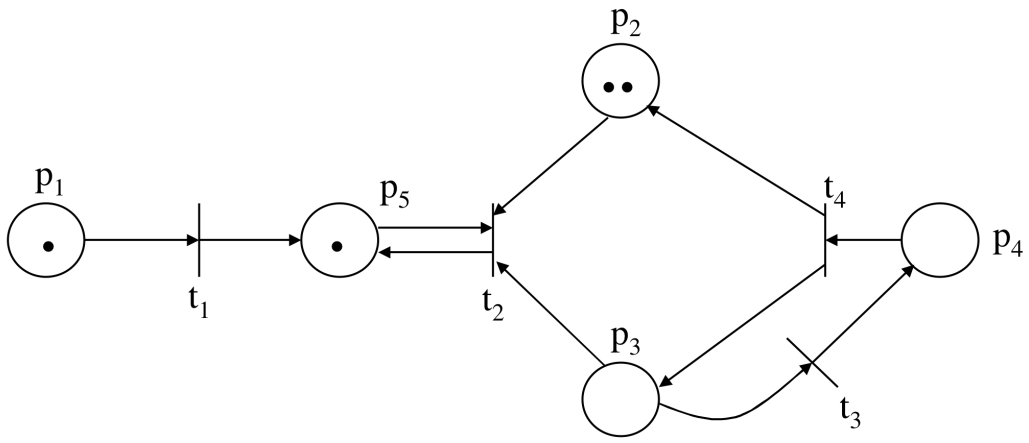


Figura 3.2 – Rede Petri com fichas

Uma rede de Petri executa através do **disparo de transições**. Uma transição dispara removendo fichas dos seus lugares de entrada e criando novas fichas, que são distribuídas nos seus lugares de saída.

Uma transição pode disparar se estiver **habilitada**. Uma transição está habilitada se cada um dos seus lugares de entrada tem, pelo menos, tantas fichas quanto arcos do lugar para a transição (**peso**).

O **estado** de uma rede de Petri é definido por sua **marcação**. Dada uma rede de Petri **C = {P, T, I, O}** e uma marcação inicial **μ0**, pode-se executar a rede de Petri pelo disparo sucessivo de transições.

A análise das marcações de uma rede de Petri busca conhecer todos os estados alcançáveis e as características de sua evolução. O algoritmo básico para esta análise consiste em montar uma árvore de alcançabilidade, onde a partir de uma marcação inicial, realiza-se o disparo de todas as transições possíveis, fazendo a rede evoluir, estado a estado, até haver a repetição de nós já existentes. Ao se analisar uma Rede de Petri, busca-se observar algumas propriedades: se a rede é limitada, viva, estritamente conservativa e reiniciável.

Um rede é limitada para uma marcação inicial M0 se todas as marcas a partir de M0 são limitadas, não apresentando acúmulo de fichas em lugares. Em termos práticos, isso significa que o protocolo atinge um número finito de estados, sendo então fisicamente implementável.

Uma rede é viva se não há estado inicial que exclua uma transição qualquer, garantindo assim que transições são sempre disparáveis. Com essa característica, um protocolo não bloqueia, não possui recepções não especificadas e não contém interações não executáveis.

Uma rede é estritamente conservativa para uma marcação inicial M0 se para toda a marcação sucessiva a M0 o número total de fichas (ou o número de fichas ponderado por um vetor) é constante, impondo limites ao protocolo.

Uma rede é reiniciável para M0 se M0 é acessível a partir de toda marcação, conferindo característica repetitiva ao protocolo, eliminando a possibilidade de *live-locks* e *dead-locks*.

Além destas propriedades gerais, identificadas através da análise clássica, podem também ser avaliadas características específicas como a análise de invariantes de lugar e transição. Os invariantes de transição, por exemplo, podem ser usados para verificar a correção parcial ou total de um protocolo. Um protocolo está parcialmente correto se as

mensagens trocadas na rede são entregues, ou seja, existe um caminho entre uma transição que envia uma mensagem e a transição que recebe esta mensagem, definindo assim seqüências cíclicas das transições. Um protocolo está totalmente correto se esta troca de mensagens ocorre em tempo finito. Já os invariantes de lugar estabelecem uma relação linear entre um subconjunto de lugares, possibilitando a verificação da propriedade de exclusão mútua onde for necessário.

3.4.2 Estelle

A técnica de descrição formal Estelle foi definida dentro da ISO [8] para especificar sistemas de processamento de informação concorrentes e distribuídos, tendo sido especialmente talhada para uma aplicação em particular: protocolos e serviços de comunicação.

A Estelle pode ser descrita como uma técnica que se baseia em um modelo de transição de estados estendida. Mais precisamente, a Estelle pode ser vista como um conjunto de extensões à linguagem Pascal definida pela ISO, que modela um sistema especificado como uma estrutura hierárquica de autômatos que podem ser executados em paralelo e que podem se comunicar através de troca de mensagens e compartilhamento de variáveis (de maneira restrita).

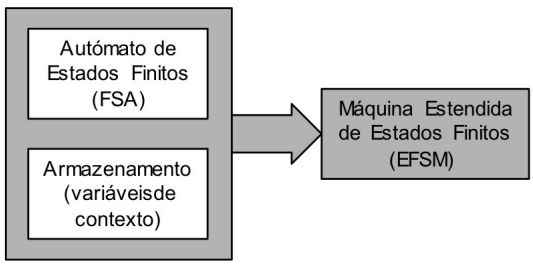


Figura 3.3 – Máquina Estendida de Estados Finitos no Estelle

Outra característica importante da linguagem é que esta permite separar a descrição das interfaces de comunicação entre componentes de um sistema especificado da descrição do comportamento interno de cada um destes componentes. E, assim como no Pascal, todos os objetos manipulados são fortemente tipificados. Esta propriedade possibilita detectar estaticamente as inconsistências da especificação.

Analisaremos, então, resumidamente, os principais conceitos da linguagem Estelle de modo a clarificar o entendimento de sua aplicabilidade e a compreensão de especificações

feitas a partir desta. Detalhes referentes à sintaxe e à semântica de Estelle podem ser encontrados na literatura técnica.

Um sistema distribuído especificado em Estelle é composto de vários componentes comunicantes. Cada componente é especificado em Estelle por uma definição de módulo. Visto que, em um sistema, pode haver mais de um componente definido textualmente pela mesma definição de módulo, é mais apropriado chamar os componentes de um sistema de instâncias de módulos.

A estrutura interna e o comportamento de um módulo são definidos explicitamente ou se deixa para posterior refinamento. A definição de um módulo consiste de um conjunto de ações de um sistema de transição de estados que este pode realizar e/ou de definições de seus sub-módulos (módulos filhos) com suas respectivas interconexões.

Um especial cuidado é tido para especificar a interface de comunicação de um módulo. Tal interface é definida utilizando três conceitos: pontos de interação, canais e interações.

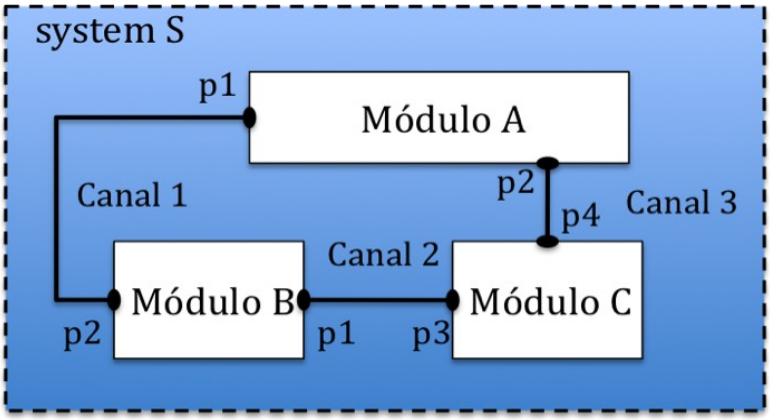


Figura 3.4 – Arquitetura de uma especificação em Estelle

Cada módulo tem um número de pontos de acesso de entrada/saída chamados de pontos de interação. Existem duas categorias de pontos de interação: externo e interno. Para cada interação um canal é associado que define dois conjuntos de interações. Estes dois conjuntos consistem de interações que podem ser recebidas ou enviadas pelo módulo através do ponto de interação. Interações são eventos abstratos (mensagens) intercambiados com o ambiente do módulo (através dos pontos de interação externos) e com os módulos filhos (através dos pontos de interação internos).

Uma definição de módulo em Estelle, como mencionado anteriormente, pode incluir definições de outros módulos. Isto, aplicado repetidamente, resulta numa estrutura em árvore

hierárquica de definições de módulos. A linguagem fornece meios de se criar instâncias de módulos filhos definidos dentro da definição do módulo, sendo que o número instâncias de um mesmo módulo pode variar dinamicamente já que elas podem ser criadas e destruídas.

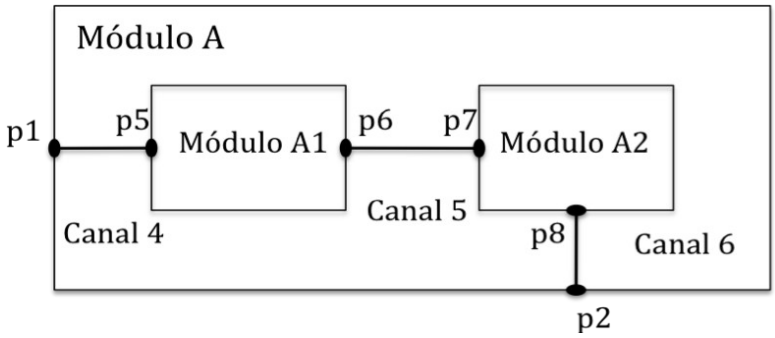


Figura 3.5 – Exemplo de estrutura interna de um módulo

Dentro desta hierarquia de módulos, eles podem se comunicar através de dois mecanismos: a troca de mensagens e o compartilhamento restrito de variáveis.

As instâncias de módulos podem trocar mensagens (chamadas de interações) entre si através de um enlace de comunicação previamente estabelecido entre seus pontos de interação. Uma interação recebida é inserida numa fila FIFO ilimitada associada com seu ponto de interação, podendo esta fila ser individual, pertencente somente a este ponto de interação, ou uma fila comum, compartilhada com outros pontos de interação de um módulo.

Com relação ao outro mecanismo de comunicação, certas variáveis podem ser compartilhadas entre um módulo e seu módulo pai. Estas variáveis têm de ser declaradas como variáveis exportadas pelo módulo e é o único meio de se compartilhar variáveis.

O comportamento dinâmico interno de um módulo Estelle é caracterizado em termos de um sistema de transição de estados não-determinístico; ou seja, definindo o conjunto de estados, os subconjuntos de estados iniciais e de relações de próximo-estado. Um estado é, em geral, uma estrutura complexa composta de muitos componentes tais como o valor de controle do estado, valores das variáveis, conteúdos das filas FIFO associadas aos pontos de interação e um status da estrutura interna do módulo. Estados iniciais de uma instância de módulo são definidos na parte de inicialização da definição do módulo. As relações de próximo-estado de uma instância de módulo são definidas por um conjunto de transições declaradas dentro de uma parte de transição da definição do módulo. Cada definição de transição contém as condições necessárias que habilitam a execução da transição, e uma ação para ser realizada

quando esta é executada. Uma ação pode mudar o estado da instância de módulo descrita acima e pode gerar interações de saída para o ambiente do módulo.

Capítulo 4

Ferramentas para TDFs

4.1 Introdução

O objetivo primordial dos métodos formais é o de ajudar engenheiros a construir sistemas mais confiáveis. Progresso nesta área depende de pesquisa de base, invenção de novos métodos e desenvolvimento de novas ferramentas, integração de diferentes métodos para trabalharem juntos, e de se concentrar esforços para que pesquisadores trabalhem com a indústria para transferência de tecnologia efetiva.

Desta forma, para disseminação do uso das TDFs para além do meio acadêmico e de pesquisa, toda abordagem formal de desenvolvimento de sistemas deve ser complementada com um suporte adequado de ferramentas.

Conforme pode ser observado na literatura, vários tipos de ferramentas vem sendo desenvolvidas para as diversas categorias de TDFs apresentadas na seção anterior:

- **editores**: como todas essas TDFs possuem uma representação textual, editores de texto convencionais podem ser utilizados para escrever especificações. Visando facilitar o trabalho do especificador foram concebidos editores orientados à sintaxe, que detectam erros e orientam online a produção de especificações, e editores gráficos, que representam especificações na forma de diagramas;
- **compiladores**: detectam erros léxicos, sintáticos e de semântica estática no código fonte de uma especificação. Na ausência de tais erros, esse código fonte é geralmente traduzido para um formato intermediário, que pode ser usado por outras ferramentas ou para gerar código em alguma linguagem de programação, auxiliando assim na implementação do sistema especificado;
- **verificadores e simuladores**: um simulador executa simbolicamente uma especificação, permitindo a aplicação de sequências de testes, as quais buscam detectar erros lógicos no projeto. Geralmente, o usuário tem como opção a execução interativa, onde a interação com o simulador pode ser feita a cada passo da execução, ou automática, onde a execução apenas é interrompida ao alcan-

çar um estado pré-determinado. Um verificador ajuda a provar as características de uma especificação analisando as propriedades a estas requeridas.

Na tabela 4.1 a seguir, ilustramos como algumas suítes de ferramentas foram desenvolvidas para dar suporte ao uso das TDFs normatizadas internacionalmente – Estelle, LOTOS e SDL --, a partir dos tipos principais de ferramentas disponibilizados [9].

Linguagem	Suíte	Editor	Compilador	Verificador/Simulador
Estelle	Estelle Development Toolset (EDT)	X	Ec (Estelle-to-C), que traduz especificações Estelle para o código C e gera uma forma intermediária (IF) para o simulador	Edb (Estelle Debugger), que é um simulador/depurador simbólico e interativo
LOTOS	MiniLITE Lotos Integrated Tools Environment (para plataforma Sun)	X	Topo (compilador e verificador de semântica estática) Colos (converte para código C)	Smile (análise parcial ou total, de forma interativa ou automática)
	Caesar/Aldebaran Development Package (CADP) (engenharia de protocolos)	X	CAESAR (parte comportamental) CAESAR.ADT (parte de dados)	Verificador ALDEBARAN
SDL	Sinderella SDL (ferramenta comercial)	Ferramenta de modelagem visual, com um editor orientado à sintaxe	X	Um simulador que dispensa a pré-compilação da especificação
	SDL Integrated Tool Environment (SITE)	* pode se integrar ao editor gráfico do Sinderella	Dois compiladores independentes capazes de traduzir uma especificação SDL para as linguagens C++ e Java.	* simulador QUEST (análise de desempenho de sistemas escritos em SDL)

Tabela 4.1 – Lista de suítes de ferramentas para TDFs

Nas seções a seguir, apresentaremos as ferramentas de suporte aos métodos formais que serão utilizados na seqüência deste trabalho.

4.2 ARP – Analisador de Redes de Petri

O Analisador/Simulador de Redes de Petri (ARP) é um programa para o auxílio ao projeto com redes de Petri, desenvolvido no LCMI/UFSC de 1985 a 1990, contando com várias ferramentas para redes de Petri ordinárias, com temporização e com temporização estendida.

O ARP foi construído em Pascal, de forma modular e permitindo uma interface com o usuário simples e de fácil aprendizado, utilizando textos e janelas. A modularidade do programa permite facilmente projetar e conectar novas ferramentas de análise ou tratamento de redes de Petri, razão pela qual o mesmo está em constante evolução.

O programa roda sobre o sistema operacional DOS na versão 3.00 ou superior, em computadores compatíveis com o IBM-PC, com um mínimo de 256 Kbytes de memória disponível. É composto de dois arquivos:

1. ARP.EXE - Programa executável, principal.
2. ARP.OVR - overlay com rotinas diversas.

A carga do programa se faz a partir do sistema operacional, digitando-se o comando arp <enter>. A partir dai será apresentada a tela principal do analisador, com as diversas opções disponíveis, e o sistema entra na janela de edição.

A primeira ferramenta do ARP é o Editor de Redes, que consiste num editor de textos simples para a edição das redes de Petri, que ainda disponibiliza funções para manipulação dos arquivos que contém as redes.

Os arquivos de rede são arquivos ASCII normais, com extensão RDP, que contém a descrição da rede de Petri numa linguagem que possui uma sintaxe semelhante à do Pascal, permitindo a identificação de lugares e transições por nomes quaisquer com até 20 caracteres.

O Compilador, que é disparado a partir da janela de Edição de Redes, carrega a estrutura da rede de Petri. Caso o texto apresente algum erro o compilador indica seu tipo e posiciona o cursor sobre o local onde o mesmo foi detectado, já pronto para a correção.

Para a análise da rede, dois métodos são disponibilizados pelo ARP:

i. Enumeração de estados - é feita através da árvore de alcançabilidade/cobertura: é dinâmica, simula a execução da RdP a partir da marcação inicial e encontra

todas as marcações possíveis. É o método mais utilizado para verificação das propriedades da RdP;

ii. Análise Estrutural - baseia-se na matriz de incidências e na equação de estados, que constituem uma tentativa de utilização de álgebra linear na análise de RdP: é estática, isto é, depende apenas da topologia da rede, e busca encontrar conjuntos de lugares e de transições com características especiais, os invariantes lineares.

São disponibilizadas uma ferramenta de Verificação, para a análise da RdP automática a partir de uma marcação inicial, e uma ferramenta de Simulação, para a análise da evolução do funcionamento da rede de maneira interativa.

Maiores detalhes a respeito do Analisador de Redes de Petri, o ARP, podem ser encontrados no manual do software em sua versão 2.3 [20].

4.3 O Sistema de Auxílio ao Projeto de Protocolos – CAD-Estelle

Como objeto de estudo deste trabalho, iremos agora fazer uma breve descrição do conjunto de ferramentas que constitui o Sistema de Auxílio ao Projeto de Protocolos de Comunicação, também chamado de CAD-Estelle, por se basear na técnica de descrição formal Estelle para a especificação dos protocolos e arquiteturas de rede a serem tratados pelo sistema.

O CAD-Estelle é um conjunto de ferramentas de projeto de protocolos em Estelle que foi desenvolvido dentro do Grupo de Teleinformática e Automação da UFRJ (GTA) e vem sendo utilizada há alguns anos como instrumento de diversos trabalhos de pesquisa dentro do grupo, tais como : Protocolo para gerenciamento hierárquico de redes de telecomunicações [31], Arquitetura para suporte a mobilidade e segurança na Internet [30], entre outros.

Em geral, a seqüência lógica de etapas percorridas ao longo de um projeto de um protocolo de comunicação obedece ao esquematizado no fluxograma a seguir.

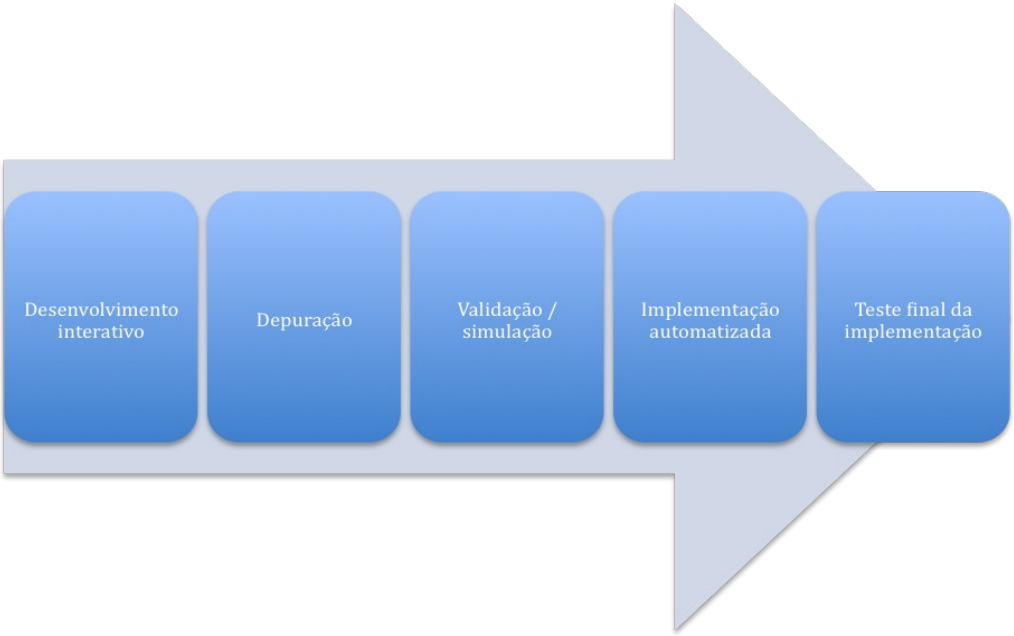


Figura 4.1 – Fluxo de projeto de um protocolo de comunicação

A arquitetura do sistema, portanto, foi inicialmente concebida de forma a contemplar o usuário com ferramentas que lhe proporcionassem maior conforto e praticidade em cada uma destas etapas.

Para a etapa da especificação formal em Estelle, foi proposto um Editor dedicado que ofereceria facilidades gráficas segundo uma forma gráfica proposta para Estelle [1, 3], assim como possibilidade de edição assistida por um editor orientado para a sintaxe e de edição de texto em Estelle.

De posse de uma especificação pronta em Estelle, o próximo passo consiste em traduzir o conteúdo desta para uma forma que lhe permita a depuração automatizada. A ferramenta Compilador para Estelle foi desenvolvida, portanto, com este intuito. O compilador traduz de um código-fonte especificado em Estelle todos os conceitos da linguagem expressos neste, descrevendo-os de forma estruturada adequada numa Forma Intermediária: os diferentes objetos de Estelle são organizados em conjuntos e a parte procedural deduzida da especificação encontra-se escrita na linguagem Modula-2. Esta Forma Intermediária estará contida em dois arquivos gerados pelo compilador, que servirão como entrada padronizada para os demais módulos de tratamento da especificação tais como o simulador de protocolos e o gerador de seqüências de testes.

A simulação funcional do protocolo, já com a especificação corretamente compilada, busca, através da execução do modelo em Estelle, permitir amplo acesso a todos os objetos que representam a dinâmica do protocolo. A ferramenta do sistema de auxílio ao projeto de

protocolos, Simulador de protocolos, cumpre esta tarefa a partir da interpretação da Forma Intermediária, o que permite o disparo de transições no protocolo, verificando-se o fluxo das trocas de mensagens (interações) entre todas as instâncias de módulos da especificação. Desta forma, pode-se analisar a correção funcional do protocolo com base no modelo proposto, observando-se a existência ou não de bloqueios e o correto tratamento de dados do protocolo.

A partir da etapa de simulação, podem ser geradas automaticamente e simuladas seqüências de teste para o protocolo, com o objetivo de submeter os sistemas de transição de estados que descrevem o comportamento dos módulos que o compõem a uma busca intensiva do espaço de estados possíveis. Tal funcionalidade é provida pelo Gerador automático de seqüências de teste, que é uma ferramenta que gera um conjunto bastante extenso de combinações de disparos possíveis a ser utilizado durante as fases de simulação e teste final da implementação de um protocolo. O modelo de informação de entrada aceito pelo gerador de seqüências de teste é o próprio conjunto de sistemas de transição de estados (facilmente convertidos em máquinas de estados finitas) extraído do corpo textual da especificação em Estelle.

Por último, para a implementação efetiva do protocolo, temos um módulo Implementador que gera um código executável transparente partindo das informações contidas na Forma Intermediária.

4.3.1 Breve histórico do CAD-Estelle

Como foi mencionado anteriormente, o CAD-Estelle foi desenvolvido no âmbito do laboratório GTA/UFRJ. Mais especificamente, ele foi fruto de uma série de trabalhos de projetos finais, teses de mestrado e de doutorado que foram sendo elaborados ao longo dos últimos anos (aproximadamente, desde 1989). Em função de tal conjuntura temporal, convém, neste ponto, traçar um breve histórico das plataformas operacionais que suportaram o CAD.

O CAD-Estelle foi, originalmente, concebido para uma plataforma de PC's baseado no sistema operacional DOS. Contudo, com o desenvolvimento dos recursos computacionais e, conseqüentemente, dos sistemas operacionais, o processo de obsolescência foi inevitável, tornando imperiosa a migração para novos ambientes que foram despontando com maior expressividade na comunidade científica. Desta forma, procedeu-se, em anos anteriores, com a adaptação e recompilação dos módulos componentes do sistema para o Sun-OS. O CAD, então, encontrava-se rodando razoavelmente sob este novo sistema operacional. Mas, novamente, o progresso bateu à nossa porta e a Sun lançou o sistema operacional Solaris, agregando vantagens atraentes, como uma interface visual amigável, entre outras que

estimulavam a migração para este sistema. Quando já se planejavam os procedimentos e estratégias para portar o CAD para o Solaris, começou a despontar no horizonte o Linux.

O Linux era, e ainda o é, um sistema operacional aberto, ou seja, você tem livre acesso ao código fonte, o que lhe dá maior flexibilidade para adequá-lo a alguma peculiaridade da sua aplicação específica. Além disso, tem sua distribuição gratuita através da Internet, já estando amplamente difundido. Aliado a isso, convém ressaltar que o sistema operacional em questão roda em plataforma de PC's, de custo significativamente inferior às estações Sun e com poder de processamento similar, ou até mesmo superior. Outro aspecto importante considerado foi o que concerne à vasta oferta dos mais variados softwares gratuitos (freewares) para Linux também pela Internet, situação bastante distinta à do Solaris por este se tratar de um sistema operacional pago.

Em função do que foi exposto acima, com a reforma do laboratório, as estações Sun foram substituídas por PC's com o sistema operacional Linux. Conseqüentemente, havia a necessidade urgente de se portar o CAD para esta nova plataforma, sob o risco de se perder todo um trabalho desenvolvido ao longo dos últimos 10 anos. A medida que algumas teses iam se desenrolando envolvendo o sistema do CAD, alguns módulos foram sendo portados para Linux, como o compilador e o implementador. Em reuniões com todos os envolvidos/interessados no assunto do CAD-ESTELLE, chegou-se ao consenso de que era prioritária a migração do simulador para Linux.

Capítulo 5

Migração do Simulador de Protocolos Descritos em Estelle – SPDE

5.1 Introdução

Em certos casos, como no caso do laboratório do GTA mencionado anteriormente, movimentos de consolidação e modernização da infraestrutura computacional envolvem migrar de um sistema operacional antigo para um que possa fornecer manutenção do sistema mais fácil ou outros benefícios de Custo Total de Propriedade.

Para o planejamento da migração de uma plataforma SunOS/Sparc para o Linux/x86, como é o caso a ser apresentado, quase todos os códigos customizados (incluindo C, *shell scripts*, Perl, etc.) precisarão de algum tipo de *correção*, um "realinhamento" de atributos de mapeamento devido às diferenças das plataformas.

A migração de softwares de médio e grande porte para Linux pode apresentar alguns desafios interessantes. Cuidados têm que ser observados em todas as etapas - desde a definição de escolhas de desenvolvimento para fazer o pacote de código-fonte funcionar até conseguir que a parte do código específica ao SO coopere com o Linux.

Mesmo considerando as semelhanças entre as diversas variantes do Unix e o Linux, deve-se, antes de começar a migração de uma grande aplicação baseada em Unix para Linux, identificar e separar para análise o código dependente do sistema operacional. Diferenças devem surgir em áreas que dependem da arquitetura, do mapeamento de memória, *threading*, entre outros aspectos específicos.

Para a discussão dessas diferenças e estruturar a metodologia de migração do SunOS na arquitetura Sparc 32-bit/64-bit para o Linux na arquitetura x86, abordaremos as seguintes questões:

- Planejamento da migração
- Ambiente de desenvolvimento (compiladores, depuração, entre outros)
- Diferenças vinculadas à arquitetura e ao sistema

5.2 Planejamento da migração

Para o planejamento da migração, listamos um roteiro com uma seqüência de passos comuns a migrações de um sistema operacional para outro, de modo a viabilizar uma execução bem sucedida:

- Preparação
- Mudanças no ambiente e de *makefile*
- Correções de compilador
- Testes e depuração
- Ajuste de desempenho
- Tipo de pacote e distribuição

Como primeiro passo, para uma preparação adequada, é essencial estudar as diferenças entre as plataformas em áreas como: chamadas do sistema (*system calls*), suporte ao sistema de arquivos, código dependente da máquina, *threading* e mapeamento de memória.

Quando da migração de uma aplicação, assegure-se de que os pacotes de terceiros estão disponíveis na plataforma-alvo. Para aplicações de 32 bits, considere se é necessário migrar para uma versão de 64 bits. Também decida que compilador a ser usado na plataforma-alvo. Para plataformas Linux baseadas em x86, o projeto gcc ("GNU Compiler Collection") é uma possível referência amplamente empregada.

Neste passo, arranja-se o ambiente de desenvolvimento, que inclui definir as variáveis de ambiente, implementar mudanças em *makefiles*, e realizar quaisquer outras alterações ao ambiente. Ao fim desta etapa, deve-se estar pronto para iniciar a construção da aplicação. Este passo pode requerer várias iterações antes de que se esteja pronto para a próxima etapa.

Nesta altura, o próximo passo é lidar efetivamente com os erros de compilação, os erros de *linker* e afins; enfim, o processo de construção da aplicação propriamente dito. Este passo também pode requerer diversas iterações até que se produza uma aplicação compilada (*build*) limpa.

O passo seguinte é o de testes e de depuração. Uma vez que a aplicação foi construída com sucesso, testes devem ser feitos para observar possíveis erros em tempo de execução. Algumas áreas de atenção durante os testes são a comunicação entre cliente/servidor, os formatos para troca de dados, conversões de dados do código como a conversão de páginas de código de byte único para páginas de múltiplos bytes, e armazenamento persistente.

Uma vez que o código foi migrado e está executando na plataforma-alvo, a monitoração de desempenho é realizada para garantir que o código apresenta o desempenho esperado. Caso contrário, o ajuste de desempenho é necessário.

Como último passo, temos a montagem de pacotes e a distribuição do código resultante migrado. O Linux oferece diversos métodos para empacotar aplicações, tais como um arquivo de compressão *tar* (*tarball*), *shell scripts* de instalação automática, ou RPM. O RPM é o sistema de gerência de pacotes amplamente utilizado no Linux. A plataforma Sun, por sua vez, usa o *pkgadmin* como gerenciador de pacotes. Há diferenças entre eles nos arquivos de modelo de especificação de pacotes. Existem soluções de empacotamento de softwares, tal como *InstallShield for Multiplatforms* (ISMP), capazes de entregar um pacote de software comum através de ambos sistemas operacionais (no caso, SunOS e Linux) que permitem reduzir o esforço de migração.

Enfim, o esforço para migração de uma plaforma SunOS para Linux sobre x86, em muitos casos, envolve somente a recompilação do código ou pequenas mudanças em parâmetros nos compiladores/*linkers*. Algumas modificações específicas à plataforma em áreas como bloqueios de recursos, mapeamento de memória, *threading* e afins podem também ser necessárias. Portanto, deve-se estudar as diferenças e planejar a migração de modo a reduzir o tempo total necessário para atingir o resultado final.

5.3 O Simulador

Conforme apresentado no capítulo anterior, a ferramenta do CAD-Estelle que é objeto de estudo deste trabalho para migração para a plataforma Linux é o Simulador de Protocolos Descritos em Estelle (SPDE) [2].

O Simulador de Protocolos, dentro do conjunto de ferramentas do CAD-Estelle, é encarregado de exercitar o conjunto de módulos cooperantes que constituem o sistema especificado em Estelle. O simulador oferece ao usuário, entre outras, as seguintes facilidades:

- Acesso a todas as instâncias de objetos definidos na especificação Estelle;
- Inserção de pontos de parada e obtenção de traços da simulação;
- Introdução de seqüências de teste durante uma parada programada da simulação.

Durante uma sessão de simulação, de acordo com a opção selecionada pelo usuário, instâncias de módulos são criadas, podendo ser eventualmente destruídas e recriadas. Ao se criar um módulo, deve-se lembrar que isto será diretamente associado à criação de um conjunto de instâncias de objetos que o compõem, tais como as filas correspondentes aos pontos de interação, as variáveis e os módulos declarados em seu interior.

Para compreender a arquitetura do simulador e o fluxo de informação entre os módulos que o compõem e a Forma Intermediária (FI), convém descrever em maiores detalhes o conteúdo desta.

A Forma Intermediária, que é o código gerado pelo compilador Estelle, consiste em 2 arquivos nos quais está descrita a estrutura da especificação numa forma que pode ser utilizada pelos demais módulos do Sistema de Auxílio. Esses dois arquivos são os seguintes: FI90 (correspondente à Forma Intermediária Estática) e FIModula.mod (correspondente à Forma Intermediária Dinâmica).

Forma Intermediária Estática

É um arquivo texto, contendo diversas listas onde estão descritas as estruturas dos objetos de Estelle da especificação. Esses objetos são, tipicamente: cabeçalhos de módulos, corpos de módulos, canais, interações, pontos de interação, nomes de transições e cláusulas de transições.

Forma Intermediária Dinâmica

É um arquivo texto em linguagem Modula-2 que contém a informação relativa ao comportamento da especificação em tempo de execução, ou seja: as cláusulas “provided” das transições, as ações das transições e as estruturas típicas da linguagem Pascal que se encontram no interior da especificação (constantes, tipos, variáveis, funções e procedimentos).

A partir desta descrição da FI, temos o seguinte conjuntos de módulos que constituem o simulador:

- Rotinas de usuário: módulo formado pelos conjuntos de procedimentos extraídos da especificação, obtidos da FI;
- Rotinas do sistema: módulo formado pelos conjuntos de procedimentos gerados pelo compilador, obtidos da FI;

- Topologia: módulo constituído pelo conjunto de objetos estáticos que formam a parte de dados do simulador, obtidos diretamente da FI;
- Contexto da simulação: módulo constituído pelo conjunto de objetos dinâmicos que são constantemente atualizados durante a simulação. Refletem o estado de uma simulação: conjuntos de transições sensibilizadas, de transições disparáveis e estruturas que definem as ligações entre os módulos;
- Primitivas Estelle: contém as primitivas que são descritas no ambiente de simulação;
- Motor do simulador.

O simulador é composto por 59 módulos mais uma rotina principal. Os dados encontram-se organizados em conjuntos que admitem um número restrito de operações, segundo o conceito de estrutura abstrata de dados.

A linguagem de desenvolvimento empregada para o mesmo foi a Modula-2 [12] devido a suas características adequadas a sistemas de médio porte, caso em que se enquadra o simulador. Dentre estas características, cabe destacar o elevado nível de decomposição possibilitado, o que permite a redução de um problema em um conjunto de problemas menores, mais fáceis de serem administrados e resolvidos. Outro trunfo do Modula-2 foi o de aliar conceitos importantes como a programação modularizada à simplicidade e à clareza do Pascal. O Modula-2 é, na verdade, um descendente do Pascal de Niklaus Wirth.

5.4 Descrição básica de Modula-2

O Modula-2 é uma notação de programação que corrige algumas das deficiências do Pascal. É particularmente adequado para o aprendizado de programação, para grandes projetos escritos e mantidos dentro de uma metodologia de engenharia de software profissional, e para sistemas embarcados de tempo real. O Modula-2 é compacto, expressivo e de fácil leitura.

Trata-se de um descendente do Pascal e do Modula, e um predecessor do Modula-2+, Modula-2*, Modula-2, Oberon, Oberon-2, e várias outras versões orientadas a objeto destes. As últimas linguagens não são substitutos para o Modula-2, sendo meramente notações posteriores dentro da mesma família, tendo suas próprias fraquezas e virtudes. O Modula-2 é, às vezes, classificado juntamente com a Ada e o C++ como o trio de linguagens modernas,

tendo em vista o expressivo poder destas. O Modula-2 é mais compacto e mais legível que ambas. A linguagem é descrita em *Programming in Modula-2 3rd edition* por Springer-Verlag em 1985 [24].

O Modula-2 foi desenvolvido para ser uma linguagem de programação de sistemas (por exemplo, compiladores, sistemas operacionais, sistemas embarcados etc) e uma sucessora do Pascal para suportar programação em larga escala. Não é uma linguagem orientada a objetos. Contudo, tem muitas características que a tornam adequada para grandes projetos de programação. As principais características do Modula-2 incluem o suporte a:

- Compilação separada através do uso de interfaces e implementações de módulos separadas;
- Corrotinas para processamento de interrupções e *multithreading*;
- Tipos "opacos" de apontadores para tipagem abstrata de dados;
- Variáveis procedimento e tipos procedimento;
- *Arrays* abertos;
- *Bit sets*.

O Modula-2 tem módulos de biblioteca compilados separadamente, e faz muito menos uso de blocos (tais como "begin .. end") que o Pascal padrão. Os identificadores são "case-sensitive", não há cláusula "go-to", e procedimentos de I/O estão contidos em bibliotecas em vez de embutidos na notação. A instrução "If" é mais versátil e há facilidades para programação concorrente via corrotinas.

Trata-se de uma linguagem com tipificação segura, o que permite, portanto, que os compiladores identifiquem muitos erros que poderiam aparecer somente durante a execução.

Suporta um desenvolvimento modular que reduz erros e tempo de manutenção. Esta característica permite também que dependências de plataformas sejam isoladas, aumentando a portabilidade. Procedimentos de I/O são encontrados em diversos módulos específicos por tipo, de modo que os *linkers* somente incluem o código de I/O que é necessário, produzindo programas menores e mais rápidos.

Um programa em Modula-2 consiste de um ou mais módulos. Existem quatro tipo de módulos:

- Módulo principal – ou programa principal (por exemplo, a função *main()* no C, ou a classe *main* no Java);

- Sub-módulo – um módulo que se aninha dentro de um outro módulo principal, sub-módulo ou módulo de implementação;
- Módulo de definição – a interface de definição de um componente; e
- Módulo de implementação – um módulo que implementa algum módulo de definição.

Todo programa deve, obrigatoriamente, ter um módulo principal apenas. Por convenção, o nome de um módulo principal deve ser o mesmo que o nome do programa.

```
MODULE Hello;
  (* declarations  of Hello *)
BEGIN
  (* main body of Hello (* with nested comments *) *)
END Hello.
```

Todas as palavras-chave reservadas em Modula-2 devem estar em letras maiúsculas. Comentários devem estar contidos por "(*" e "*)", que podem estar aninhados (desde que todos os comentários em questão estejam dentro dos delimitadores).

Componentes pré-definidos (tais como uma biblioteca de gráficos, uma biblioteca de IO, um tipo de dados abstratos de fila, um gerenciador de janelas etc) são construídos a partir de módulos de definição e de implementação. Um módulo de definição especifica uma API acessível publicamente de um componente, enquanto que o módulo de implementação fornece a implementação de fato do componente. Para uma dada API (um módulo de definição), podem ser fornecidas diferentes implementações desde que todas satisfaçam aos requisitos da mesma API.

Um módulo de definição pode ser entendido com um “contrato” entre os usuários e os implementadores de um componente, ele define a que um usuário pode ter acesso e o que o implementador deve entregar. Uma vez que a API esteja definida, usuários e implementadores de um componente podem seguir diferentes caminhos. O usuário pode fazer uso do componente *importando* sua interface. É responsabilidade do implementador garantir que a implementação entregue satisfaça a API. Construir componentes a partir de interfaces e implementações bem definidas é um dos mais importantes princípios da decomposição em módulos na programação em larga escala.

Um módulo de definição especifica os procedimentos, variáveis e tipos de dados de acesso público de um componente, logo, não deve conter nenhum código executável. Tudo aquilo especificado dentro de um módulo de definição é de acesso público. Por sua vez,

procedimentos, variáveis e tipos de dados privados devem ser fornecidos no módulo de implementação.

Um tipo de dado abstrato (opaco) pode ser encapsulado por um módulo de definição, sendo sua representação interna ocultada de seus usuários. O Modula-2 não suporta herança nem modelos genéricos. Desta forma, não se pode definir facilmente um módulo de tipo de dado abstrato genérico no Modula-2, independente dos valores de seus dados (por exemplo, INTEGER).

```
DEFINITION MODULE IntStack;
  TYPE
     anIntStack; (* an opaque pointer type *)
   PROCEDURE New( VAR s : anIntStack ); (* allocate a new
anIntStack *)
  PROCEDURE Push( VAR s : anIntStack; i : INTEGER );
  PROCEDURE Pop( VAR s : anIntStack ) : INTEGER;
  PROCEDURE Empty( s : anIntStack ) : BOOLEAN;
END IntStack.
```

Portanto, para completar o "contrato" mencionado anteriormente, é necessário fornecer uma implementação. As decisões de implementação não são objeto de preocupação dos usuários, sendo os detalhes ocultados dentro do próprio módulo de implementação, que normalmente não está disponível no formato fonte para nenhum usuário.

```
IMPLEMENTATION MODULE IntStack;
  CONST
      MAX = 10;
  TYPE
       (* our opaque type is represented and defined here *)
     anIntStack = POINTER TO aStack; (* an opaque type must be
a POINTER type *)
     aStack = RECORD stk : ARRAY [1..MAX] OF INTEGER; count :
INTEGER; END;
  PROCEDURE New( VAR s : anIntStack );
  BEGIN
     (* allocate a new RECORD for "s" and then initialize
"count" *)
```

```
    ...
  END New;
  PROCEDURE Push ...  (* code for Push *)
  PROCEDURE Pop ...  (* code for Pop *)
  PROCEDURE Empty( s : anIntStack ) : BOOLEAN;
  BEGIN
    RETURN ( s^.count = 0 );
  END Empty;
BEGIN (* IntStack *)
  (* nothing *)
END IntStack.
(Note: There is an IMPLEMENTATION keyword in front of MODULE.)
```

Quando se compila um módulo de implementação, o compilador automaticamente procura pelo módulo de definição de mesmo nome. Cada módulo de implementação é compilado em um módulo objeto realocável, que está pronto para ligação.

Para usar um componente existente, a API é necessária portanto. No Modula-2, importa-se a definição de um módulo/componente. A palavra "importar", no caso, significa "usar". Freqüentemente, não se necessita do componente inteiro, apenas de alguns poucos tipos ou procedimentos de acesso fornecidos pela API. Pode-se, então, importar de maneira seletiva o que é necessário.

Em muitos compiladores de Modula-2, existem diversos componentes de biblioteca pré-construídos, tais como:

- Módulo IO – um módulo de input/output de console;
- Módulo FIO – um módulo de input/output de arquivo;
- Módulo *Storage* – módulo de alocação e desalocação de memória;
- Módulo *Strings* -- um módulo de conversão e de manipulação de *strings* de propósito geral;
- Módulo *SYSTEM* – módulo de suporte ao sistema em execução;
- Módulo *Window* – um módulo gerenciador de janelas baseadas em texto, etc.

De modo a trabalhar numa descrição padrão do Modula-2 e definir um conjunto de módulos de biblioteca padrão, um comitê da ISO (JTC1/SC2/WG13) foi organizado a partir de 1987. O Modula-2 ISO, aprovado em 1996, resolveu, então, a maioria das ambigüidades

do Modula-2 clássico. Ele adiciona o tipo de dados COMPLEX e LONGCOMPLEX, exceções, terminação de módulos (cláusula FINALLY) e uma biblioteca padrão de I/O completa.

Abordamos aqui, com um pouco mais de detalhe, as principais características e notações de sintaxe da linguagem Modula-2 suficientes para viabilizar a leitura e a compreensão geral do código do simulador a ser portado para a plataforma Linux/x86. Maiores informações sobre outros aspectos de programação em Modula-2 podem ser encontrados em vários livros, como por exemplo [23] e [24].

5.5 Ambiente de desenvolvimento para migração

A tarefa de migração para a plataforma operacional Linux consistiu, de acordo com o roteiro de planejamento de migração apresentado anteriormente, em recompilar o código-fonte do simulador: neste caso, todos os módulos de definição, de implementação e de programa que o compõem. O primeiro passo de preparação foi a escolha de um ambiente de desenvolvimento na linguagem Modula-2 para a plataforma Linux – ajuste de variáveis de ambiente, escolha de um editor, compilador e depurador.

Convém registrar, a esta altura, que a distribuição de Linux escolhida para o desenvolvimento deste trabalho foi a Fedora versão 14, da comunidade Red Hat, por esta distribuição ser a utilizada nas máquinas do laboratório do GTA quando se iniciou o trabalho. De modo a evitar qualquer contratempo futuro, tomou-se o cuidado de escolher um ambiente de desenvolvimento que pudesse ser portado de forma transparente para outras distribuições de expressividade no mercado.

Iniciou-se o trabalho de recompilação com o único compilador de distribuição gratuita encontrado após algum período de busca pela Internet. Após a instalação e a verificação do funcionamento do mesmo em algumas máquinas do laboratório do GTA, com o decorrer das recompilações dos diversos módulos do simulador, diversos problemas de incompatibilidade entre as bibliotecas oferecidas pelo compilador de Modula-2 utilizado para a plataforma Sun-OS e as do compilador em questão foram enfrentados. Estudando as diferenças e as restrições apresentadas pelas bibliotecas do compilador para Linux notamos que estas concentravam-se, principalmente, nas que disponibilizavam procedimentos para a manipulação de arquivos, texto/*strings* e números.

Compiladores mais antigos de Modula-2, como é o caso do compilador para a plataforma SunOS [21] utilizado para a construção do simulador, implementam a linguagem descrita em "Programming in Modula-2" (PIM) de [24] – o livro que foi um padrão "de facto" por vários anos. Devido à falta de um padrão oficial, desenvolvedores de compiladores PIM também introduziram várias extensões à linguagem. Uma outra questão a considerar durante a análise dos erros de compilação são os tamanhos dos tipos base: INTEGER, CARDINAL e BITSET tem 16 bits em compiladores antigos de 16 bits, enquanto que em compiladores mais modernos estes tipos tem 32 bits. Outro problema, que foi observado na prática como relatado anteriormente, se deve ao fato de não existir nenhum padrão de bibliotecas definido para a linguagem Modula-2 conforme foi inicialmente proposta (PIM), de modo que cada implementação tem o seu próprio conjunto de bibliotecas.

Tais dificuldades apontavam para soluções que envolviam a adaptação e a reescrita de trechos de código de diversos módulos e, também, a criação de bibliotecas adicionais – implementadas em outra linguagem, por exemplo – agrupando vários procedimentos/funções não encontrados nas bibliotecas do compilador para Linux.

Fortuitamente, pouco depois deste início nebuloso, um link na Internet para um outro compilador de Modula-2 de distribuição gratuita para Linux foi encontrado. Já com uma visão geral mais clara a respeito de todo o processo de migração, seria mais fácil avaliar objetivamente se este compilador responderia aos problemas encontrados até então.

Trata-se do compilador Mocka (MOdula Compilter KArlsruhe), mantido pela Universidade de Karlsruhe [22]. O compilador implementa a linguagem Modula-2, conforme definida por Niklaus Wirth, com umas poucas extensões, tais como a inclusão de alguns tipos (SHORTINT, LONGINT, SHORTCARD e LONGCARD), detalhes de acesso a procedimentos em módulos estrangeiros e regras de compatibilidade e operações em tipos mais relaxadas em função dos tipos adicionais implementados; além de algumas restrições de implementação. O Mocka é um compilador Modula-2 compatível com PIM3 (*Programming in Modula-2* 3ª edición). É de livre distribuição, gera executáveis compactos e eficientes e sua instalação é limpa e segura. A documentação é suficiente.

O Mocka inclui uma pequena biblioteca de sistema que compreende, entre outros, procedimentos de acesso a dispositivos padrão de entrada/saída (InOut), de acesso formato/não-formatado de entrada/saída via arquivos (TextIO, ByteIO), interfaces para o sistema Unix (Clock, SysLib), um módulo para funções matemáticas (MathLib), módulos para conversões (NumConv, RealConv) e um módulo para manipulação de strings (Strings1). Analisando os procedimentos disponibilizados por estas bibliotecas acima, confirmou-se que

a substituição dos procedimentos utilizados das bibliotecas do compilador para Sun-OS por estes seria mais fácil de ser efetivada devido à maior interseção funcional entre os conjuntos.

Para a operação básica do compilador, temos a opção de compilar os módulos um por um, diretamente via linha de comando do Linux, ou através de um ambiente interativo abrindo-se uma sessão no sistema do compilador. A principal vantagem desta segunda opção é que oferece uma facilidade de *make* que suporta o desenvolvimento e a manutenção de grandes programas compostos de muitas unidades de compilação. Desta forma, o usuário não precisa se preocupar com a ordem de compilação dos módulos de um programa, tarefa esta que fica a cargo do próprio ambiente de programação disponibilizado.

Para fins de registro e referência, uma lista bem abrangente de outros compiladores já desenvolvidos para Modula-2 para as diversas plataformas operacionais pode ser encontrada neste link - http://freepages.modula2.org/compi.html -, onde são descritas sucintamente todas as opções.

Para complementar o ambiente de desenvolvimento em Modula-2, inicialmente utilizou-se a facilidade de edição dos módulos disponibilizada pelo próprio Mocka em modo de sessão interativa, em conjunto com o editor "vi". Posteriormente, devido a interdependência de vários dos numerosos 59 módulos do simulador, optou-se por uma ferramenta de edição mais avançada – o Emacs, mantida pelo projeto GNU, de distribuição gratuita. Havia a necessidade de se analisar e manipular vários arquivos de módulos concomitantemente, e o Emacs facilita tal tarefa por disponibilizar um ambiente de janelas. Além disso, e também de fundamental importância, o Emacs suporta a função de destaque de sintaxe para o Modula-2 (com cores diferenciadas), o que tornava a leitura e a edição bem mais confortável e eficiente.

Com o avanço das iterações de compilação e depuração de erros, e também após pesquisas mais aprofundadas a respeito das ferramentas existentes, chegou-se a uma proposta de um ambiente de desenvolvimento e depuração que propiciaria maiores conforto e eficiência para este processo, combinando o editor de texto Kate [26], o compilador Mocka e o depurador de erros DDD [25].

O Kate é um editor de texto com opções avançadas: navegador de arquivos e emulador de terminal integrados, suporte a múltiplos arquivos, destaque de sintaxe para múltiplos tipos de arquivos, integração de ferramentas e plugins etc. Vem instalado com pacote do kdebase.

Não existe um entorno gráfico integrado para o Mocka. O mais próximo disso poderia se obter usando o editor Kate dentro do ambiente KDE, que é o diferencial desta proposta.

O Kate dispõe de uma extensão que permite integrar um pequeno terminal debaixo da janela de edição e, a partir deste terminal, pode-se fazer tudo o que não é possível pelo Kate ou pelos *scripts*.

Sem sair do Kate, portanto, pode-se editar qualquer módulo em um dado diretório. Usando os scripts de Kate pode-se compilar, revisar erros, gerar um programa executável e, por que não, executá-lo, tudo isso sem sair do Kate.

Os scripts disponíveis encontrados permitem:

⇨ Mocka: Compilar módulo

Compila qualquer módulo mocka em seu diretório (seja um módulo de definição ou de implementação) e, caso se produza qualquer erro, divide a janela em duas, coloca o cursor na linha e coluna em que detecta e mostra o erro ocorrido na parte inferior.

⇨ Mocka: Executar programa

Somente se o módulo que se está editando é um programa e está corretamente gerado, parâmetros de execução serão solicitados e um terminal será aberto com a saída gerada pelo programa

⇨ Mocka: Gerar executável

Se o módulo que se está editando é um programa, tentará gerar o executável correspondente, realizando o relacionamento de todos os módulos. A janela de mensagem da compilação se dividirá ou será reutilizada para informar os possíveis erros.

⇨ Mocka: Erro →

Se a compilação produziu mais de um erro, pode-se utilizar esta opção para avançar ao erro seguinte. O cursor se colocará na linha e coluna em que se detectou o erro.

⇨ Mocka: Erro ←

Permite revisar o erro anterior, caso exista. Também moverá o cursor até a posição em que se detectou o erro.

Para depuração, o DDD é uma interface gráfica de usuário para o gdb - suportado pelo Mocka -, que é o depurador GNU. O DDD pode fazer o mesmo que o gdb, porém de maneira mais cômoda, já que dispõe de menus, barras de ferramentas, janelas etc. Pode-se executar um programa, estabelecer pontos de parada, examinar variáveis ou, inclusive, modificá-las para localizar erros rapidamente.

Enfim, uma vez cumpridas as etapas de preparação e de definição de ambiente de desenvolvimento, vamos apresentar a seguir o processo de compilação e depuração

propriamente ditos do código do simulador, elencando as modificações necessárias de maneira estruturada.

5.6 Migração do SPDE

Um primeiro detalhe a ser observado diz respeito à convenção de nomenclatura dos arquivos que compõem o programa em Modula-2. O arquivo contendo um módulo deve consistir da concatenação do nome do módulo em questão e uma extensão identificando o tipo de arquivo. Em especial, a convenção de extensão adotada para os módulos de implementação e de programa é ".mi"; e para os módulos de definição, ".md". As demais extensões de arquivos de programas em Modula-2 são dos arquivos gerados durante e após a compilação, podendo ser consultadas no Manual do compilador [22].

A primeira medida tomada para a recompilação do simulador, portanto, foi trocar a extensão dos nomes dos arquivos de todos os módulos, já que a convenção utilizada pelo compilador para Sun-OS era diferente: para os módulos de implementação e de programa, trocou-se ".mod" por ".mi", e para os de definição, ".def" por ".md".

A seguir, então, estão descritas todas as modificações e soluções implementadas nos módulos para a migração do simulador. Primeiro, os módulos de biblioteca afetados, e suas respectivas variáveis, tipos e procedimentos, foram agrupados de acordo com o perfil funcional visando facilitar a análise de possível impacto das mudanças na funcionalidade deste e do sistema como um todo. Na seqüência, modificações relacionadas com incompatibilidades de tipos, restrições do compilador ou afins são apresentadas indicando em quais módulos foram implementadas.

Foram usadas as bibliotecas disponíveis no diretório ./lib sob o diretório de instalação do compilador Mocka.

5.6.1 Manipulação de Strings

I.1) Strings.mod

O módulo Strings.mod consta do pacote do simulador e define quais procedimentos de manipulação de *strings* serão utilizadas pelos demais módulos do software.

O procedimento “Delete” acrescenta uma lógica para limitar o número de caracteres a ser removido ao número máximo permitido considerando a posição inicial de onde se quer começar a apagar e o tamanho da *string* em questão.

Os procedimentos “Compare”, “Position” e “Substring” foram renomeados como “CompareStr”, “Pos” e “Copy”, respectivamente, para facilidade de leitura e evitar conflitos com procedimentos de outros módulos.

A biblioteca String.def do compilador para SunOS é utilizada (importada) por este módulo e não existe no Mocka. Solução: usar a biblioteca Strings.md.

Em função do nome da biblioteca do Mocka se chamar “Strings”, foi necessário renomear o módulo “Strings” para “Strngs” para evitar erros de compilação. Esta modificação foi propagada para todos os demais módulos que importam o módulo “Strings”.

O tipo TYPE CompareResult = (less, equal, greater) definido na biblioteca String.def da Sun não é utilizado e, portanto, não precisou ser portado para o Mocka.

A seguir, temos uma tabela indicando a correspondência entre os procedimentos da bibliotecas String (SunOS) e Strings (Mocka).

SunOS (String)	Linux/Mocka (Strings)
PROCEDURE Length (str: ARRAY OF CHAR) (* in this string *) : CARDINAL; (* count of data chars *)	PROCEDURE Length (VAR str: ARRAY OF CHAR):CARDINAL; (* returns the number of significant characters. *)
PROCEDURE Assign (source : ARRAY OF CHAR; (* copy from this str *) VAR dest : ARRAY OF CHAR; (* to this string *) VAR success : BOOLEAN);	PROCEDURE Assign (VAR dst, src: ARRAY OF CHAR); (* assign string 'src' to string 'dst'. 'src' must be terminated by 0C *)
PROCEDURE Insert (source : ARRAY OF CHAR; (* insert this string *) VAR dest : ARRAY OF CHAR; (* into this string *) index : CARDINAL; (* before this char pos*) VAR success : BOOLEAN);	PROCEDURE Insert (substr: ARRAY OF CHAR; VAR str: ARRAY OF CHAR; inx: CARDINAL); (* Inserts 'substr' into 'str', starting at str[inx] *)
PROCEDURE Delete (VAR str: ARRAY OF CHAR; (* from this string *) index : CARDINAL; (* 1st char to delete *) len : CARDINAL; (* cnt of chars to del *) VAR success : BOOLEAN);	PROCEDURE Delete (VAR str: ARRAY OF CHAR; inx, len: CARDINAL); (* Deletes 'len' characters from 'str', starting at str[inx] *)
PROCEDURE Position (pattern: ARRAY OF CHAR; (* search for this one *) source : ARRAY OF CHAR; (* within this one *) VAR index : CARDINAL; (* 1st char of match *) VAR found : BOOLEAN);	PROCEDURE pos (substr: ARRAY OF CHAR; str: ARRAY OF CHAR): CARDINAL; (* Returns the index of the first occurrence of 'substr' in 'str' or *) (* HIGH (str) + 1 if 'substr' not found. *)
PROCEDURE Substring (source : ARRAY OF CHAR; (* copy from this str *) index : CARDINAL; (* at this char postn *) len : CARDINAL; (* for this char count *) VAR dest : ARRAY OF CHAR; (* into this string *) VAR	PROCEDURE Copy (str: ARRAY OF CHAR; inx, len: CARDINAL; VAR result: ARRAY OF CHAR); (* Copies 'len' characters from 'str' into 'result', *) (* starting at str[inx] *)

success : BOOLEAN);	
PROCEDURE Concat (source1 : ARRAY OF CHAR; (* this string with *) source2 : ARRAY OF CHAR; (* this str on the end *) VAR dest : ARRAY OF CHAR; (* assigned to this str*) VAR success : BOOLEAN);	PROCEDURE Concat (s1, s2: ARRAY OF CHAR; VAR result: ARRAY OF CHAR); (* Returns in 'result' the concatenation of 's1' and 's2' *)
PROCEDURE Compare (string1 : ARRAY OF CHAR; (* compare this string *) string2 : ARRAY OF CHAR) (* to this string *) : CompareResult;	PROCEDURE compare (s1, s2: ARRAY OF CHAR): INTEGER; (* Compares 's1' with 's2' and returns -1 if s1 < s2, 0 if s1 = s2, *) (* or 1 if s1 > s2 *)

Tabela 5.1 – Correspondência String=>Strings

5.6.2 Módulos de Entrada e Saída

II.1) InOut

As bibliotecas InOut para o SunOS e para o Linux, apesar de possuírem um razoável conjunto de funções idênticas, apresentam estratégias diferentes de abordagem para a disponibilização de serviços de I/O.

No compilador para o SunOS, a biblioteca fornece acesso aos canais de entrada e saída padrão do programa em Modula-2 de maneira genérica (compatível com o PIM), contendo procedimentos para leitura e escrita de strings, caracteres e números. Por padrão, opera nos canais de arquivo de entrada e saída padrão do Unix (stdin, stdout), porém, diferentemente da biblioteca do Mocka, permite que estes possam ser modificados (procedimentos OpenInput e OpenOutput).

A biblioteca InOut do Mocka, por outro lado, se propõe a oferecer serviços SOMENTE para os canais padrão de arquivo de entrada e saída, porém apresentando um conjunto de serviços mais completo, incluindo leitura e escrita do tipo REAL. Já a biblioteca para o Sun disponibiliza outras duas bibliotecas, SimpleIO e RealIO, para os serviços de leitura e escrita de texto e de números do tipo REAL, respectivamente, para os canais padrão de arquivo texto de entrada e saída especificamente (normalmente o terminal de console).

Comparação de serviços entre as bibliotecas:

- Para ambos: Read, ReadString, ReadCard, ReadInt, Write, WriteString, WriteInt, WriteCard, WriteOct, WriteHex, WriteLn;
- Somente para Linux: ReadReal, ReadLongReal, WriteReal, WriteLongReal, WriteBf, Done, EOF;
- Somente para Sun: OpenInput, OpenOutput, CloseInput, CloseOutput, Done(VAR), EOL(CONST = 12C).

Obs.: Na biblioteca InOut do Mocka, a função WriteBf deve ser invocada após uma escrita para que o conteúdo possa ser enviado para stdout.

II.2) SimpleIO

O módulo SimpleIO.mi foi incluído para substituir os serviços importados das bibliotecas SimpleIO e RealIO da Sun pelos demais módulos do simulador.

Todos os procedimentos e funções oferecidos pelas duas bibliotecas da Sun foram implementados utilizando os equivalentes funcionais da biblioteca InOut do Mocka. O único procedimento da Sun que não teve equivalência direta com um procedimento do Mocka foi o "ReadLn", que foi implementado utilizando o procedimento "Read(VAR ch: CHAR)" e a constante EOL.

II.3) Terminal

Esta biblioteca oferece acesso ao dispositivo terminal de controle do usuário de um programa em Modula-2 na plataforma SunOS. As facilidades disponibilizadas são similares às das bibliotecas InOut e StandardIO da Sun, porém sem o redirecionamento de I/O que estes módulos permitem. O módulo permite interagir diretamente com o dispositivo controlador de teclado e de *display*. Esta é uma biblioteca específica do compilador para SunOS devido a características desta plataforma computacional.

Para a implementação com o compilador Mocka em Linux, os procedimentos ligados a operações de I/O para terminal podem ser suportados pelas bibliotecas InOut e SimpleIO (apresentadas anteriormente). Devido a isto, e por ser uma biblioteca com serviços mais específicos para o SunOS, no contexto da migração do simulador para o ambiente Linux, a biblioteca Terminal não será utilizada.

5.6.3 Módulos de Acesso ao ambiente

III.1) Clock

A biblioteca Clock no compilador SunOS oferece acesso ao relógio em tempo real do sistema através de funções que convertem o tempo do Unix (ou *epoch*) em estruturas de dados que permitiram a conversão deste tempo em um formato legível de hora e data (formato de *string*). O tempo do Unix representa o número de segundos que se passou desde 1º de janeiro de 1970, 0:00h (GMT/UTC); ou seja, é literalmente o tempo zero (t0) do Unix.

No compilador Mocka, a biblioteca Clock apresenta uma abordagem distinta, oferecendo procedimentos para obter o tempo do usuário e o tempo do sistema desde a última vez em que foi executado o procedimento de reset do relógio (ResetClock). É possível,

contudo, obter o tempo atual em formato do tempo do Unix a partir de procedimentos da biblioteca SysLib que permitem acesso ao sistema (dependente do tipo de máquina): PROCEDURE time (VAR t: INTEGER);

Como esta biblioteca só é utilizada pelo módulo Random para geração de números aleatórios, os procedimentos disponíveis pela biblioteca do Mocka são suficientes para fornecer números pseudo-aleatórios para o algoritmo de geração randômica.

5.6.4 Módulos de funções matemáticas

IV.1) MathLib

A biblioteca MathLib do compilador para o SunOS também está disponível no Mocka para o ambiente Linux. Os mesmos procedimentos são encontrados em ambas as bibliotecas, sendo a passagem de argumentos feita da mesma forma também. O único cuidado a se observar, porém, é a necessidade de se converter os nomes dos procedimentos para que iniciem com letra minúscula ao invés de maiúscula. Ex.: Sqrt(SunOS) -> sqrt(Mocka).

IV.2) Random

Trata-se de um módulo gerador de números randômicos criado com base em uma biblioteca do compilador Logitech de Modula-2 para o ambiente DOS.

Na solução no ambiente SunOS, utilizou-se a biblioteca Clock para fornecer uma entrada pseudo-aleatória a partir de informações de hora e data do sistema para o algoritmo gerador randômico. Os tipos de dados de ponteiro para o tempo do sistema e o tempo local foram importados (ptrLocalTime e ptrSystemTime), assim como os procedimentos para obter estes tempos (GetLocalTime e GetSystemTime).

Para portar para ambiente Linux com o Mocka, os procedimentos UserTime e SystemTime, que retornam um valor pseudo-aleatório do tipo INTEGER, satisfazem a necessidade de entrada do algoritmo randômico e, portanto, foram utilizados em substituição.

IV.3) FloatingUtilities

O módulo FloatingUtilities implementa funções matemáticas de ponto flutuante envolvendo o tipo REAL que não são disponibilizadas por bibliotecas do compilador para o SunOS. As duas funções principais para a implementação deste módulo são:

- PROCEDURE Float (x : INTEGER) : REAL; que converte um número inteiro em um número real; e

⇨ PROCEDURE Trunc (real : REAL) : INTEGER; que converte um número real em um número inteiro.

Originalmente, no SunOS, o módulo importava as bibliotecas MathLib, Convert e ConvertReal para conseguir implementar estas duas funções, fazendo uma conversão intermediária para string antes de converter para real ou inteiro, dependendo da função em questão.

No caso do compilador Mocka, temos a biblioteca LREAL, que disponibiliza as seguintes funções:

PROCEDURE LTRUNC (x: LONGREAL): LONGINT

PROCEDURE LFLOAT (x: LONGINT): LONGREAL

Considerando as regras de compatibilidade de atribuição de tipos de variáveis, podemos utilizar estas funções com os tipos REAL e INTEGER, sem restrições de implementação.

5.6.5 Módulos de Conversão

V.1) Convert

A biblioteca Convert deve ser substituída pela biblioteca NumConv. Deve-se observar as seguintes modificações:

SunOS (Convert)	Linux/Mocka (NumConv)
PROCEDURE IntToStr (int : INTEGER; VAR str : ARRAY OF CHAR; width : CARDINAL; VAR success : BOOLEAN);	PROCEDURE Num2Str (num: LONGCARD; base: tBase; VAR str: ARRAY OF CHAR; VAR done: BOOLEAN); (* Convert 'num' to 'str' using 'base' *)
PROCEDURE CardToStr(card : CARDINAL; VAR str : ARRAY OF CHAR; width : CARDINAL; VAR success : BOOLEAN);	
PROCEDURE NumToStr (num : CARDINAL; VAR str : ARRAY OF CHAR; base : CARDINAL (* [2..36] *); width : CARDINAL; VAR success : BOOLEAN);	
PROCEDURE StrToInt (str : ARRAY OF CHAR; VAR int : INTEGER; VAR success : BOOLEAN);	PROCEDURE Str2Num(VAR num: LONGCARD; base: tBase; str: ARRAY OF CHAR; VAR done: BOOLEAN); (* Convert 'str' to 'num' using 'base' *)
PROCEDURE StrToCard(str : ARRAY OF CHAR; VAR card : CARDINAL; VAR success : BOOLEAN);	
PROCEDURE StrToNum (str : ARRAY OF CHAR; VAR num : CARDINAL; base : CARDINAL (* [2..36] *); VAR success : BOOLEAN);	

Tabela 5.2 – Correspondência Convert=>NumConv

V.2) ConvertReal

A biblioteca ConvertReal deve ser substituída pela biblioteca RealConv. Deve-se observar as seguintes modificações:

SunOS (ConvertReal)	Linux/Mocka (RealConv)
PROCEDURE RealToStr (real : REAL; VAR str : ARRAY OF CHAR; width : CARDINAL; (* width of field in char *) decPlaces: INTEGER; (* neg -> sci; 0 -> no point *) VAR success : BOOLEAN);	PROCEDURE Real2Str(x : REAL; n : CARDINAL; k : INTEGER; VAR s: ARRAY OF CHAR; VAR done: BOOLEAN); (* Convert real 'x' into external representation. *) (* IF k > 0 use k decimal places. *) (* IF k = 0 write 'x' as integer. *) (* IF k < 0 use scientific notation. *)
PROCEDURE StrToReal (str : ARRAY OF CHAR; VAR real : REAL; VAR success : BOOLEAN);	PROCEDURE Str2Real(s: ARRAY OF CHAR; VAR done: BOOLEAN): REAL; (* Converts the string 's' to real 'x'. *) (* s has to be of the form: *) (* real = num ['.' {digit}] ['E' num]. *) (* num = ['+'|'-'] digit {digit}. *)

Tabela 5.3 – Correspondência ConvertReal=>RealConv

V.3) NumberConversion

O módulo NumberConversion reúne procedimentos para conversão de strings em números do tipo INTEGER e CARDINAL e vice-e-versa.

Na implementação em SunOS, a biblioteca de conversão Convert era utilizada e disponibilizava procedimentos separados para tratar de números do tipo INTEGER e do tipo CARDINAL.

Já a solução adotada pelo Mocka é através da biblioteca NumConv, que define um par de procedimentos apenas (Num2Str e Str2Num), tratando todos os tipos de números como números do tipo LONGCARD. Devido às regras de compatibilidade de atribuição de tipos de variáveis, podemos utilizar estes procedimentos com os tipos INTEGER e CARDINAL, sem restrições de implementação.

5.6.6 Módulos de manipulação de arquivos

VI.1) Files

A biblioteca Files, em conjunto com as bibliotecas Text e Binary, do compilador para a Sun oferece serviços de I/O formatados (texto) e não formatados (binário) via arquivos com

controle de erros embutido nos procedimentos, através de uma variável que retorna o estado do arquivo. Um tipo enumerado “FileState” é criado com tipos de erros mais comuns.

No SunOS, a biblioteca Files é responsável pelas operações de criação e deleção, abertura e fechamento, verificação de fim de arquivo, e alguns outros procedimentos para posicionamento do ponteiro para leitura e escrita de arquivos. Os procedimentos de leitura e escrita formatados e não formatados são disponibilizados pelas bibliotecas Text e Binary, respectivamente.

No compilador Mocka para o Linux, a abordagem para oferecer o mesmo grupo de serviços é um pouco diferente. As bibliotecas BasicIO, ByteIO e TextIO são disponibilizadas. A biblioteca ByteIO, como o próprio nome sugere, oferece procedimentos de I/O não formatados, enquanto que a biblioteca TextIO oferece serviços I/O formatados em texto. Contudo, as operações de criação, abertura, fechamanto e deleção de arquivos estão definidas em todas as três bibliotecas.

Para definir o tipo de permissão de acesso ao arquivo a ser manipulado - leitura ou escrita -, na biblioteca Files do SunOS há um tipo enumerado “ReadWriteMode”, enquanto que na biblioteca do Mocka é disponibilizado um procedimento “Accessible”, que permite confirmar se a permissão é de leitura ou escrita.

Para o tratamento de estado de arquivo no Mocka, uma função “Done” é disponibilizada para confirmar se a última operação solicitada foi realizada com sucesso ou não. Em caso de erro, é necessário utilizar a biblioteca ErrNumbers, que retorna o erro gerado a partir do procedimento “ErrNo()”. Os códigos de erro do sistema estão definidos através de constantes nesta biblioteca.

VI.2) FileSystem

O módulo FileSystem reúne os procedimentos de manipulação de arquivos que foram definidos para uso pelos demais módulos que compõem o simulador. Este grupo de procedimentos foi implementado originalmente a partir de procedimentos da biblioteca Files do SunOS. Para portar para Linux, utilizamos os procedimentos equivalentes da biblioteca TextIO conforme tabela a seguir:

Procedimento Módulo FileSystem	Procedimento Biblioteca TextIO (Mocka)
PROCEDURE ReadChar(VAR f:Files.File; VAR ch:CHAR);	TextIO.GetChar(f: File, VAR ch: CHAR);
PROCEDURE Lookup (VAR f:Files.File; filename:ARRAY OF CHAR; newfile:BOOLEAN);	TextIO.OpenInput(f: File, filename: ARRAY OF CHAR, newfile: BOOLEAN);
PROCEDURE WriteChar(VAR f:File; ch:CHAR);	TextIO.PutChar(f: File, ch: CHAR); TextIO.PutBf(f: File);
PROCEDURE Close(VAR f:Files.File);	TextIO.Close(f: File);

Tabela 5.4 – Implementação do módulo FileSystem com a biblioteca TextIO

5.6.7 Módulos de Gerenciamento gráfico

VII.1) Telas

O módulo Telas disponibilizava, no pacote de software original do simulador escrito para o sistema operacional DOS, procedimentos que se baseavam em bibliotecas para gerenciamento de janelas em modo texto. Contudo, estas bibliotecas não estão disponíveis nos compiladores para o SunOS e para o Linux.

Desta forma, os blocos de código referentes a este grupo de procedimentos exportado pelo módulo Telas foi colocado como comentário (entre os marcadores '(*' e '*)').

Para os demais procedimentos deste módulo, as bibliotecas InOut, para serviços de I/O em stdin/stdout, e Files/FileSystem, para manipulação de arquivos texto, são utilizadas.

5.6.8 Módulos da arquitetura do simulador

Em alguns módulos que implementam as funcionalidades do software do simulador propriamente dito, modificações nos algoritmos dos procedimentos tiveram que ser feitas, ocasionalmente, em função das mudanças de paradigma nas bibliotecas de suporte do sistema discutidas até este ponto.

Listamos, a seguir, alguns dos módulos que se enquadram neste caso e uma breve explicação sobre as modificações realizadas.

1) SPDE

No módulo principal do programa, em função das modificações feitas no módulo FileSystem a partir da biblioteca TextIO do compilador Mocka, onde o controle de execução de procedimento não é mais retornado através da variável File, mas sim através do procedimento *Done*, a seguinte construção REPEAT teve de ser adaptada:

```
REPEAT
      Lookup ( FileNotation, "SPDE.NOT", TRUE ) ;
UNTIL Done; (*  FileNotation.res = done ; -- trecho original em SunOS
*)
```

2) Notation

Neste módulo de implementação, além de um caso parecido com a cláusula REPEAT mostrado anteriormente, temos o exemplo do procedimento WriteLn, definido dentro do procedimento principal exportado EscreveNomes, que, como o nome sugere, tem a função de escrever o caracter de fim de linha (EOL) a um arquivo texto. Originalmente implementado com o procedimento WriteChar da biblioteca FileSystem e a constante EOL da biblioteca Files, agora o procedimento foi reimplementado com o procedimento PutLn da biblioteca TextIO, conforme a seguir:

```
PROCEDURE WriteLn;
        BEGIN
          TextIO.PutLn(FileNotation);
          (*WriteChar(FileNotation, EOL) – trecho original em SunOS *)
        END WriteLn;
```

5.6.9 Módulos de entrada do simulador – a Forma intermediária

Conforme visto na introdução sobre as características do simulador (SPDE), os dois arquivos que compõem a Forma Intermediária, entrada de informação para o simulador, são gerados pelo compilador Estelle (CE). A versão de compilador utilizada neste trabalho é a que roda sobre o sistema operacional DOS e que gera como saída os seguintes arquivos: FIMODULA.MOD e FI90.

No caso do arquivo FIMODULA.MOD, este precisa ser renomeado para FIModula.mi, para atender à regra de nomenclatura do compilador Mocka.

Uma modificação no código padrão gerado pelo compilador para este módulo também é necessário no que diz respeito à declaração de tipos. Isto se deve ao fato de o tipo “set” ter uma restrição de implementação no compilador Mocka para o Linux: os tipos set devem conter elementos cujos números ordinários estão na faixa de 0 a 31.

Desta forma, as seguintes declarações de tipo devem ser removidas ou colocadas como comentário no código do módulo FIModula antes da execução do simulador:

```
(* SETCHAR = SET OF CHAR;
  SETBYTE = SET OF [-128..127]; *)
```

Obs.:

(i) Não se pode declarar a interface de exportação no módulo de definição pois o compilador não reconhece a diretiva "export".

(ii) Uma variável do tipo procedimento não poderá ser importada por um módulo de implementação e ali ser executado o procedimento, pois poderá ocorrer erro de compilação ou de execução.

5.7 Testes e depuração do simulador

Uma vez concluída a compilação de todos os módulos componentes do software do simulador e a conseqüente obtenção de um *build*, procedeu-se com os testes em tempo de execução e a depuração do pacote obtido. Para tal, fez-se necessária a escolha de um protocolo de comunicação descrito em Estelle para explorar as facilidades do simulador.

Dentro da linha abordada neste trabalho para definir uma metodologia para o projeto de serviços de rede baseada no modelo de referência OSI da ISO, utilizaremos para fins de testes um protocolo padronizado em conjunto pela ISO e pela ITU apresentado no relatório técnico ISO/IEC TR 10167 [10], em que são apresentadas orientações para o uso de Estelle.

O protocolo em questão é o Abracadabra, que convenientemente contém diversos conceitos OSI e, portanto, se mostra bastante adequado para fins ilustrativos e de testes neste contexto. O Abracadabra implementa um serviço confiável, orientado à conexão, de transferência de dados entre dois usuários. O protocolo opera sobre um meio que oferece um serviço de transferência de dados não confiáveis.

A especificação do protocolo Abracadabra testada possui cinco instâncias de módulo: Usuário A, Usuário B, Abracadabra A, Abracadabra B e Meio DG. Com relação ao comportamento das entidades Abracadabra (A e B), temos os seguintes estados:

- CLOSED – estado inicial em que o protocolo encontra-se com a conexão fechada;
- WFCC – após o envio de pedido de abertura de conexão à entidade, o protocolo espera uma PDU de confirmação de abertura dessa conexão (CC – connect confirm);
- OPEN – estado em que o protocolo encontra-se com a conexão aberta;
- WFAK – nesse estado, o protocolo espera o reconhecimento do envio de uma PDU de dado, durante a fase de transmissão de dados;
- CLOSING – estado onde o protocolo encontra-se encerrando a conexão;
- WFUR – estado em que o protocolo espera a resposta de seu usuário a um pedido de abertura de conexão da entidade remota.

Os testes feitos com o Abracadabra buscaram simular algumas situações, como o estabelecimento de conexão com e sem sucesso, a transferência de dados com e sem sucesso e o encerramento da conexão com e sem sucesso entre os usuários. Convém ressaltar que o objetivo foi o de explorar as opções de execução passo-a-passo da especificação, verificação e modificação dos estados das instâncias, de variáveis, das filas, reiniciar uma sessão; enfim, testar o uso do simulador observando se o comportamento do protocolo seria de acordo com o que está definido no padrão ISO.

A seguir, temos os modelos de 3 camadas da especificação com a indicação em ordem numérica do fluxo de mensagens esperado para três dos cenários de teste executados. Os traços coletados do simulador correspondentes são apresentados e brevemente analisados para indicar os resultados dos testes.

1. Solicitação de abertura de conexão

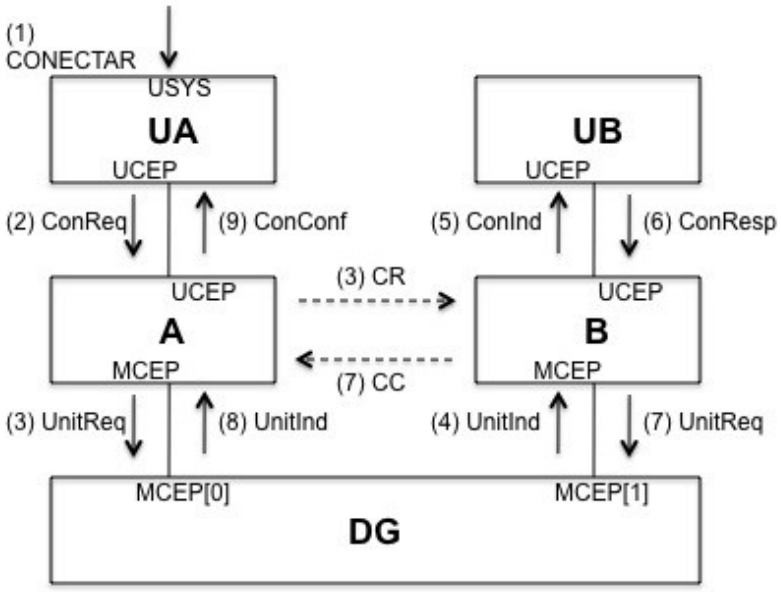


Figura 5.1 – Cenário de abertura de conexão

Traços do simulador:

```
0   | corpo especif.            | INITSENTakb          |     0
=============================================================
 0   | DG[ 0]                    | LINHA608ajsajt       |     0
=============================================================
 0   | UA[ 0]                    | LINHA452ahbahc       |     0
=============================================================
 0   | UB[ 0]                    | LINHA452ahbahc       |     0
=============================================================
 0   | A[ 0]                     | INITAENTady          |     0
=============================================================
 0   | B[ 0]                     | INITAENTady          |     0
=============================================================
 1   | INTERACAO DO USUARIO ->                          |     3
     |                           |----------------------|
     |  --> USYS[ 0]             | CONECTAR             |
=============================================================
 1   | UA[ 0]                    | CONDOUSERahe         |     3
     |                           |----------------------|
     |  -->  [ 0]                | CONECTAR             |
=============================================================
 1   | UA[ 0]                    | U1MANDACONREQaic     |     3
     |                           |----------------------|
     |  --> UCEP[ 0]             | CONREQ               |
=============================================================
 0   | A[ 0]                     | A1MANDACRaea         |     3
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
=============================================================
 0   | DG[ 0]                    | PARABajk             |     3
     |                           |----------------------|
     |  --> MCEP[ 1]             | UNITIND              |
=============================================================
```

```
 0   | B[ 0]                    | A8RECCRaeo           |     3
     |                          |----------------------|
     |  --> UCEP[ 0]            | CONIND               |
=============================================================
 2   | UB[ 0]                   | U13RECCONINDajb      |     3
=============================================================
 2   | UB[ 0]                   | U10MANDACONRESPaiv   |     3
     |                          |----------------------|
     |  --> UCEP[ 0]            | CONRESP              |
=============================================================
 0   | B[ 0]                    | A11MANDACCaeu        |     3
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITREQ              |
=============================================================
 0   | DG[ 0]                   | PERDIDOPARAAajq      |     3
=============================================================
 0   | A[ 0]                    | A6REPETECRaek        |     7
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITREQ              |
=============================================================
 0   | DG[ 0]                   | PERDIDOPARABajm      |     7
=============================================================
 0   | A[ 0]                    | A6REPETECRaek        |    11
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITREQ              |
=============================================================
 0   | DG[ 0]                   | PERDIDOPARABajm      |    11
=============================================================
 0   | A[ 0]                    | A6REPETECRaek        |    15
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITREQ              |
=============================================================
 0   | DG[ 0]                   | PERDIDOPARABajm      |    15
=============================================================
 0   | A[ 0]                    | A6REPETECRaek        |    19
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITREQ              |
=============================================================
 0   | DG[ 0]                   | PARABajk             |    19
     |                          |----------------------|
     |  --> MCEP[ 1]            | UNITIND              |
=============================================================
 0   | B[ 0]                    | A20CRATRREPETECCafs  |    19
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITREQ              |
=============================================================
 0   | DG[ 0]                   | PARAAajo             |    19
     |                          |----------------------|
     |  --> MCEP[ 0]            | UNITIND              |
=============================================================
 0   | A[ 0]                    | A2RECCCCRaec         |    19
     |                          |----------------------|
     |  --> UCEP[ 0]            | CONCONF              |
=============================================================
 1   | UA[ 0]                   | U5RECCONCONFaik      |    19
=============================================================
```

O fluxo de mensagens na seqüência prevista no diagrama se confirma, porém é interessante observar as retransmissões de dados solicitadas por ambas entidades do

Abracadabra (A, como iniciador; e B, como respondedor) em função do meio de transporte não ser confiável. Os intervalos de retransmissão são diferentes para cada entidade: a cada 4 segundos, para a entidade iniciadora A; e 16 segundos, para entidade respondedora B.

2. Solicitação de transferência de dados (envio)

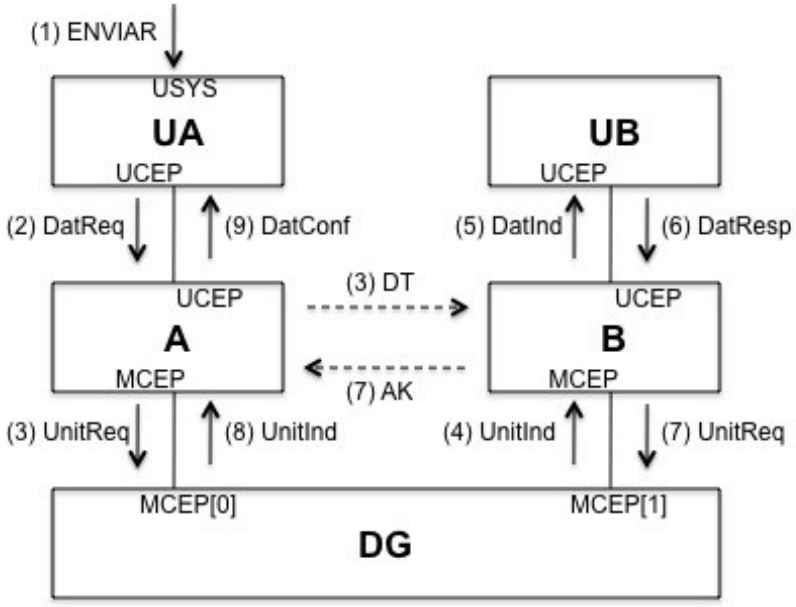
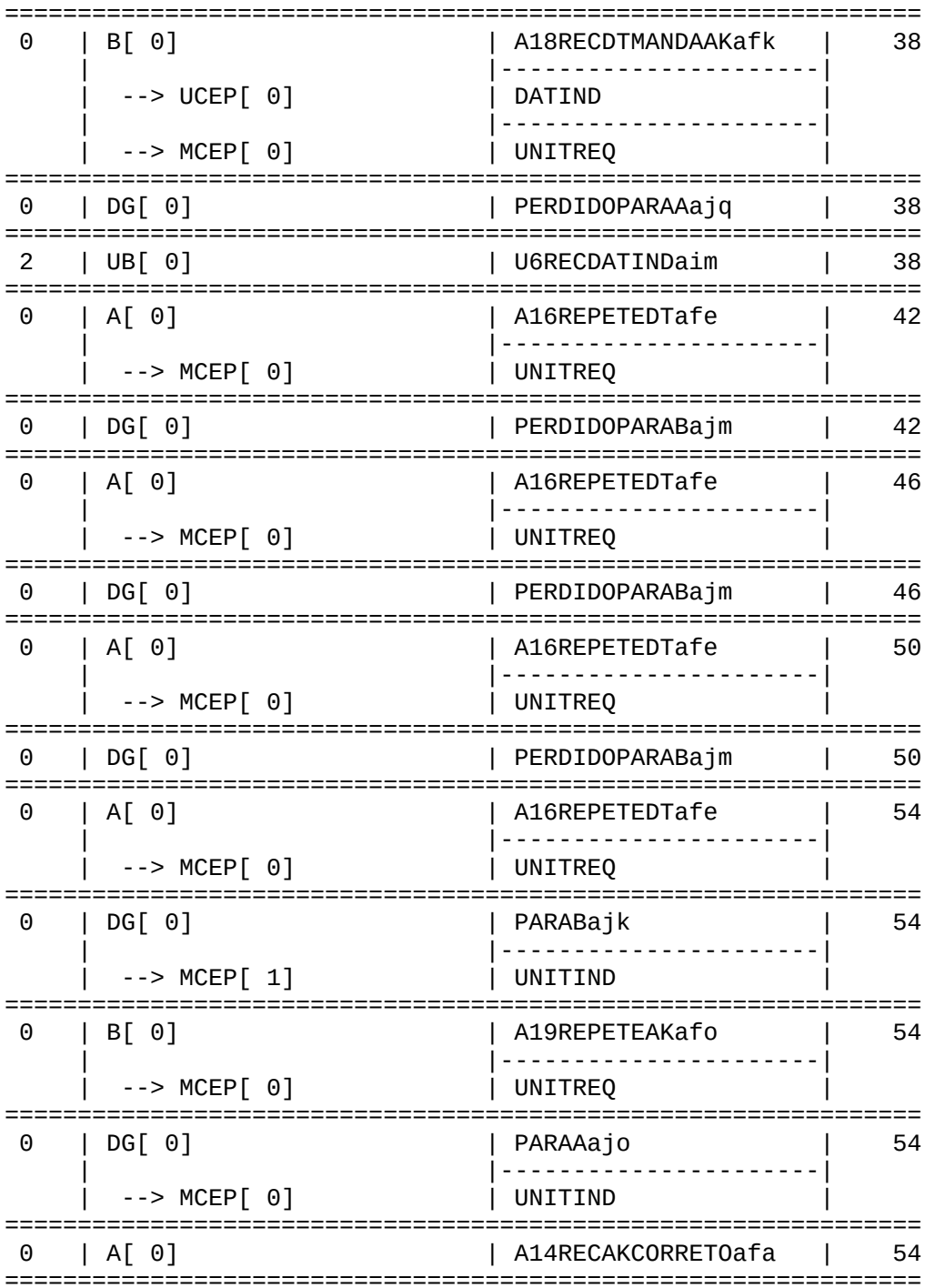


Figura 5.2 – Cenário de envio de dados

Traços do simulador:

```
================================================================
 1   | INTERACAO DO USUARIO ->                          |    34
     |                              |-------------------|
     |  --> USYS[ 0]                | ENVIAR            |
================================================================
 1   | UA[ 0]                       | ENVIADOUSERaia    |    34
     |                              |-------------------|
     |  -->  [ 0]                   | ENVIAR            |
================================================================
 1   | UA[ 0]                       | U7MANDADATREQaip  |    34
     |                              |-------------------|
     |  --> UCEP[ 0]                | DATREQ            |
================================================================
 0   | A[ 0]                        | A13MANDADTaey     |    34
     |                              |-------------------|
     |  --> MCEP[ 0]                | UNITREQ           |
================================================================
 0   | DG[ 0]                       | PERDIDOPARABajm   |    34
================================================================
 0   | A[ 0]                        | A16REPETEDTafe    |    38
     |                              |-------------------|
     |  --> MCEP[ 0]                | UNITREQ           |
================================================================
 0   | DG[ 0]                       | PARABajk          |    38
     |                              |-------------------|
     |  --> MCEP[ 1]                | UNITIND           |
```



O fluxo de mensagens na seqüência prevista no diagrama se confirma, porém é interessante observar que, nesta fase de troca de dados, a mensagen DatResp não é entregue pelo usuário respondedor (UB), tampouco a mensagem DatConf é enviada para o usuário iniciador (UA), sendo somente necessário o envio do AK pela entidade respondedora do Abracadabra para cada solicitação de envio de dados pela entidade par iniciadora através da PDU DT.

3. Solicitação de desconexão

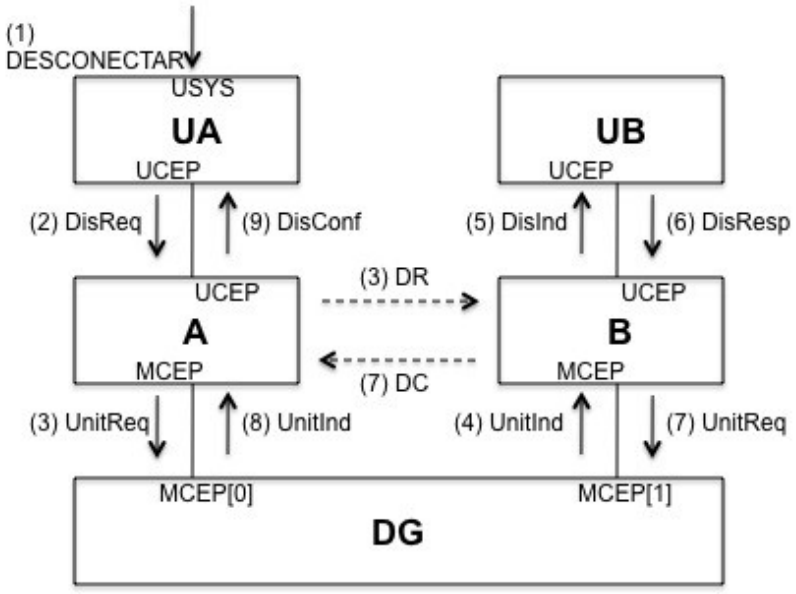


Figura 5.3 – Cenário de desconexão

Traços do simulador:

```
================================================================
 1   | INTERACAO DO USUARIO ->                          |    77
     |                           |----------------------|
     |  --> USYS[ 0]             | DESCONECTAR          |
================================================================
 1   | UA[ 0]                    | DISCONDOUSERahs      |    77
     |                           |----------------------|
     |  -->  [ 0]                | DESCONECTAR          |
================================================================
 1   | UA[ 0]                    | U9MANDADISREQait     |    77
     |                           |----------------------|
     |  --> UCEP[ 0]             | DISREQ               |
================================================================
 0   | A[ 0]                     | A23USUMANDADRagc     |    77
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
================================================================
 0   | DG[ 0]                    | PARABajk             |    77
     |                           |----------------------|
     |  --> MCEP[ 1]             | UNITIND              |
================================================================
 0   | B[ 0]                     | A21RECDRMANDADCafw   |    77
     |                           |----------------------|
     |  --> UCEP[ 0]             | DISIND               |
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
================================================================
 0   | DG[ 0]                    | PERDIDOPARAAajq      |    77
================================================================
 2   | UB[ 0]                    | U8RECDISINDair       |    77
================================================================
 0   | A[ 0]                     | A26REPETEDRagk       |    81
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
```

```
============================================================
 0   | DG[ 0]                    | PARABajk             |    81
     |                           |----------------------|
     |  --> MCEP[ 1]             | UNITIND              |
============================================================
 0   | B[ 0]                     | A9RECDRMANDADCaeq    |    81
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PERDIDOPARAAajq      |    81
============================================================
 0   | A[ 0]                     | A26REPETEDRagk       |    85
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PARABajk             |    85
     |                           |----------------------|
     |  --> MCEP[ 1]             | UNITIND              |
============================================================
 0   | B[ 0]                     | A9RECDRMANDADCaeq    |    85
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PERDIDOPARAAajq      |    85
============================================================
 0   | A[ 0]                     | A26REPETEDRagk       |    89
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PERDIDOPARABajm      |    89
============================================================
 0   | A[ 0]                     | A26REPETEDRagk       |    93
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PERDIDOPARABajm      |    93
============================================================
 0   | A[ 0]                     | A26REPETEDRagk       |    97
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PARABajk             |    97
     |                           |----------------------|
     |  --> MCEP[ 1]             | UNITIND              |
============================================================
 0   | B[ 0]                     | A9RECDRMANDADCaeq    |    97
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PERDIDOPARAAajq      |    97
============================================================
 0   | A[ 0]                     | A26REPETEDRagk       |   101
     |                           |----------------------|
     |  --> MCEP[ 0]             | UNITREQ              |
============================================================
 0   | DG[ 0]                    | PARABajk             |   101
     |                           |----------------------|
     |  --> MCEP[ 1]             | UNITIND              |
============================================================
 0   | B[ 0]                     | A9RECDRMANDADCaeq    |   101
```

```
     |                              |----------------------|
     |  --> MCEP[ 0]                | UNITREQ              |
=============================================================
 0   | DG[ 0]                       | PERDIDOPARAAajq      |   101
=============================================================
 0   | A[ 0]                        | A26REPETEDRagk       |   105
     |                              |----------------------|
     |  --> MCEP[ 0]                | UNITREQ              |
=============================================================
 0   | A[ 0]                        | A27NAOREPETEDRagm    |   105
=============================================================
 0   | DG[ 0]                       | PERDIDOPARABajm      |   105
```

Assim como nos outros dois cenários simulados, o fluxo de mensagens na seqüência prevista no diagrama se confirma. Mais uma vez é explorada o comportamento de retransmissões por parte de ambas as entidades, porém desta vez, o número máximo de retransmissões por parte da entidade iniciadora (A) é atingido e esta não repete mais o envio da PDU de pedido de desconexão (DR).

5.8 Distribuição e uso do simulador no Linux

Finalizadas as etapas de compilação e de testes e depuração do software do simulador, uma modalidade de empacotamento dos módulos foi escolhida para facilitar a distribuição do simulador.

Como não há nenhum requisito específico de instalação do software do simulador, bastando que o compilador de Modula-2 (no caso, o Mocka) esteja funcionando com todas as variáveis de ambiente ajustadas sempre que o mesmo for iniciado, todos os arquivos que o compõem serão agrupados na pasta SPDE e esta será condensada e comprimida no arquivo de distribuição SPDE.tar.gz.

Para executar o simulador, basta lançar o programa SPDE compilado através do Mocka com os arquivos da Forma Intermediária gerados pelo compilador Estelle – FI90 e FIModula.mod – devidamente já renomeados (conforme seção 5.6.9) e copiados para o diretório de instalação do simulador.

Capítulo 6

Estudo de caso: especificação, verificação e simulação do protocolo RTSP

6.1 Introdução

Até este ponto, foram abordados inicialmente o projeto e os princípios propostos para o desenvolvimento de serviços de rede, quando então foram apresentados os conceitos e terminologia do modelo de referência OSI da ISO, comumente mais utilizado para a descrição e estruturação de projetos do que como um padrão para interconexão de sistemas.

Na seqüência, um estudo sobre métodos formais aplicados ao desenvolvimento de sistemas, de modo a garantir maior correção e confiabilidade, foi feito, discutindo aspectos sobre especificação e verificação e as diversas técnicas e vertentes encontradas na literatura e indústria. Dentre estes formalismos, apresentamos em mais detalhes as Redes de Petri e a linguagem Estelle, que particularmente mostram características interessantes para o suporte ao projeto de serviços de rede.

Uma avaliação sobre as ferramentas disponibilizadas para o emprego de métodos formais foi apresentada, assim como uma breve descrição das funcionalidades das ferramentas ARP, para a análise de Redes de Petri, e do CAD-Estelle, para o projeto de protocolos descritos em Estelle.

Por último, apresentamos o estudo e o resultado de um dos objetivos deste trabalho que foi a migração de uma das ferramentas do CAD-Estelle – o simulador de protocolos SPDE – para a plataforma operacional Linux/x86 visando, assim, mantê-lo disponível em um ambiente computacional mais amplamente utilizado e facilitar a integração deste com as demais ferramentas do CAD-Estelle.

Neste capítulo, por fim, a proposta é a de experimentar todos estes temas que concernem o projeto de um serviço de rede e fazer uso das ferramentas mencionadas, em especial do simulador SPDE, através de um estudo de caso. A escolha foi por um protocolo de comunicação cujas características se mostrassem adequadas à metodologia de projeto apresentada e que fizesse parte de aplicações em rede em voga. Dentro destes critérios, a escolha foi pelo protocolo RTSP [27], que apresentamos a seguir.

6.2 Descrição do RTSP

O RTSP, "Real Time Streaming Protocol", é um protocolo do nível de aplicação para controle de transferência de dados com propriedades de tempo real. O RTSP torna possível a transferência, sob demanda, de dados em tempo real, como áudio e vídeo. Ele serve para estabelecer e controlar um único ou vários *streams* sincronizados de mídias contínuas pertencentes a uma apresentação.

O conjunto de *streams* a ser controlado é definido por uma descrição de apresentação, normalmente um arquivo, que pode ser obtido por um cliente usando HTTP ou outro meio, como e-mail, e pode não necessariamente estar armazenado em um servidor de mídia.

Uma descrição de apresentação contém informações sobre um ou mais *streams* que compõe a apresentação, como endereços de rede e informações sobre o conteúdo da apresentação (por exemplo, assunto, e-mail do responsável pela sessão, tempo da apresentação), além de parâmetros que tornam possível ao cliente escolher a combinação mais apropriada das mídias. Na descrição da apresentação, cada *stream* é individualmente identificado por uma URL RTSP, a qual aponta para um servidor de mídia que trata aquele *stream* particular e dá um nome ao *stream* armazenado naquele servidor. Vários *streams* (áudio e vídeo) podem estar localizados em servidores diferentes para compartilhamento de carga. Além disso, a descrição da apresentação também descreve quais métodos de transporte o servidor é capaz de oferecer. Vários modos de operação são utilizados, como *unicast* e *multicas*t.

Em relação ao funcionamento do RTSP, não existe a noção de uma conexão RTSP; ao invés disso, um servidor mantém uma sessão indicada por um identificador. Uma sessão RTSP não está ligada à uma conexão no nível de transporte, como acontece numa conexão TCP. Durante uma sessão RTSP, um cliente RTSP pode abrir e fechar conexões de transporte para o servidor para emitir requisições RTSP, sendo que, normalmente, o controle RTSP pode acontecer em uma conexão TCP, enquanto o fluxo de dados se dá via UDP ou RTP; porém, a operação do RTSP não depende do mecanismo de transporte utilizado para o transporte das mídias contínuas.

O protocolo suporta as seguintes operações: recuperação de mídia de um servidor de mídia, convite de um servidor de mídia para uma conferência (apresentação ou gravação de uma mídia, ou de um subconjunto, na conferência), e adição de mídias a uma apresentação existente.

O RTSP é bastante similar em sintaxe e operação ao HTTP, o que o permite utilizar diversas extensões e recursos desenvolvidos para este. Assim como o HTTP, existem 2 tipos de mensagem, *request* e *response*, só que novos métodos foram disponibilizados: SETUP, DESCRIBE, PLAY, PAUSE, RECORD, TEARDOWN, REDIRECT, entre outros. A mensagem "*response"* retorna um código de status, indicando se o método solicitado pela mensagem *"request"* foi atendido (característica também herdada do HTTP).

No esquema abaixo, temos um cenário de operação básica do RTSP:

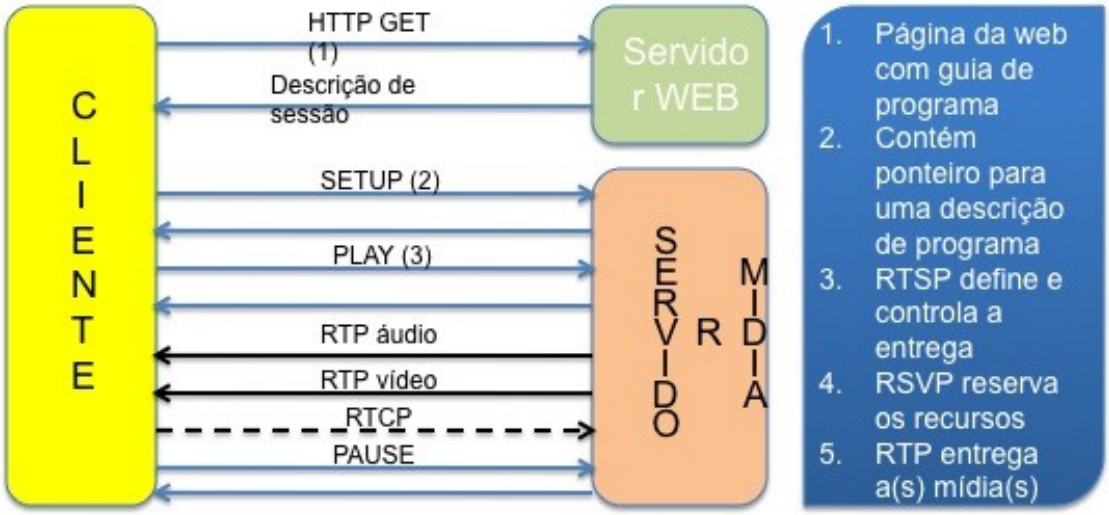


Figura 6.1 – Fluxo de mensagens de operação do RTSP

Definição dos métodos do RTSP:

- SETUP – estabelece o transporte
- DESCRIBE – solicita a descrição de mídia
- PLAY – inicia reprodução, reposicionamento
- RECORD – inicia gravação
- PAUSE – suspende entrega, mas mantém estado
- TEARDOWN – remove estado, desaloca os recursos para o transporte
- REDIRECT – redireciona o cliente para novo servidor

Um único *stream* de mídia, durante uma sessão RTSP, pode ser controlado por pedidos RTSP disparados seqüencialmente em conexões TCP diferentes. Desta forma,

diferentemente do HTTP, o servidor RTSP precisa manter o "estado da sessão" para ser capaz de correlacionar os pedidos RTSP com uma *stream*.

Dentre os métodos supralistados, somente o método DESCRIBE não afetará as máquinas de estados que definem o comportamento do protocolo no cliente e no servidor desde a iniciação da sessão RTSP até sua finalização. Já os métodos SETUP, PLAY, RECORD, PAUSE e TEARDOWN tem um papel central na definição de alocação e de uso de recursos de *stream* no servidor.

6.3 Arquitetura em camadas segundo modelo OSI-ISO

A partir da descrição "informal" apresentada na seção anterior compilada a partir das informações contidas na RFC2326 [27], estruturamos a especificação do protocolo RTSP usando o modelo RM-OSI da ISO como ferramenta.

O primeiro passo é definir a arquitetura do serviço seguindo o modelo hierárquico de três camadas, sendo quatro entidades propostas: usuário, RTSP cliente, RTSP servidor e Meio de transporte. Para o desenvolvimento desta especificação, consideraremos que o serviço da entidade Meio de transporte é confiável, com garantia de entrega de dados (o RTSP normalmente utiliza o protocolo TCP).

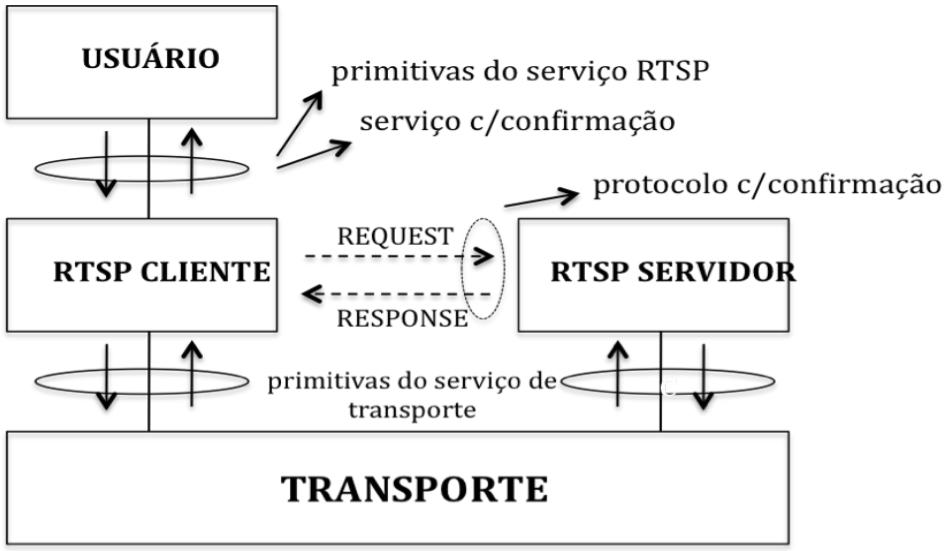


Figura 6.2 – Arquitetura em 3 camadas do RTSP

Para definir os serviços fornecidos por cada camada da arquitetura proposta, listamos as primitivas de serviço do RTSP e do meio de transporte. O serviço provido pela camada

RTSP é do tipo com confirmação, usando apenas dois tipos de primitivas (de acordo com o HTTP): request e response. O serviço da camada de transporte também é fornecido através de dois tipos de primitivas: request e indication.

Seguindo a regra de nomenclatura de primitivas proposta pela ISO (ver seção 1.2), adotamos a sigla RT para a camada RTSP, e a sigla T para a camada de transporte.

Primitivas do RTSP

- RT_Desc.req
- RT_Desc.resp(status, dado)
- RT_Open.req(dado)
- RT_Open.resp(status)
- RT_Play.req
- RT_Play.resp(status)
- RT_Pause.req
- RT_Pause.resp(status)
- RT_Close.req
- RT_Close.resp(status)

O campo "status" carrega o código de resposta ao método solicitado, conforme definido na RFC. De acordo com o objetivo deste trabalho, iremos simplificar a extensa lista proposta da seguinte forma para fins de representação:

- '+' – "2xx" – método realizado com sucesso
- '-' – "4xx" -- método não realizado
- 'RED' – "3xx" – servidor redirecionado

Primitivas do transporte

- T_Data.req(PDU)
- T_Data.ind(PDU)

Detalhando o protocolo da camada em estudo, o RTSP, temos as seguintes unidades de dados do protocolo (PDUs) trocadas entre entidades pares nos dois sentidos, conforme mostrado no diagrama de 3 camadas, de acordo com os métodos definidos na RFC:

Tipos de PDU	
REQUEST	RESPONSE
SETUP	SETUP(+)
	SETUP(-)
	SETUP(RED)
DESC	DESC(status)
PLAY	PLAY(status)
PAUSE	PAUSE(status)
TEAR	TEAR(status)
RED	RED(status)

Tabela 6.1 – Tipos de PDU do RTSP

Uma vez definidas a arquitetura do contexto das especificações das entidades, as primitivas dos serviços e os tipos de PDU do protocolo RTSP, apresentamos alguns cenários de uso do serviço em questão indicando o fluxo de mensagens trocadas.

1. Estabelecimento de transporte para apresentação de mídia (abertura de sessão)

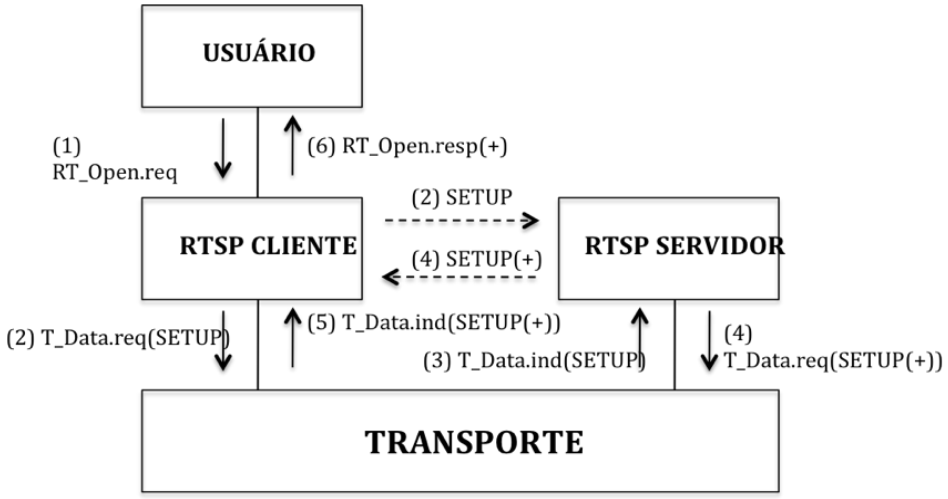


Figura 6.3 – Cenário de estabelecimento de transporte para abertura de sessão RTSP

2. Reprodução de apresentação de mídia

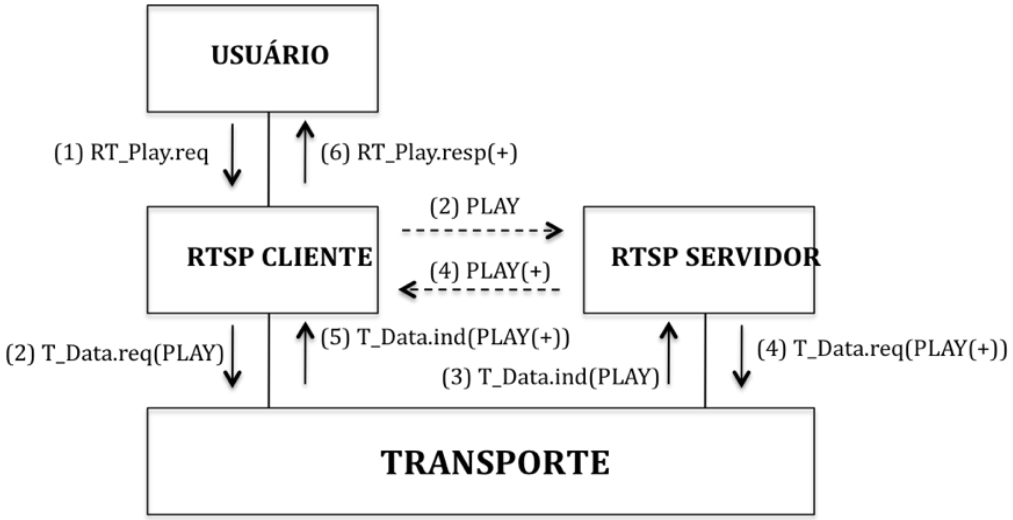


Figura 6.4 – Cenário de pedido de reprodução de mídias

3. Pausa na reprodução da apresentação de mídia (não realizada)

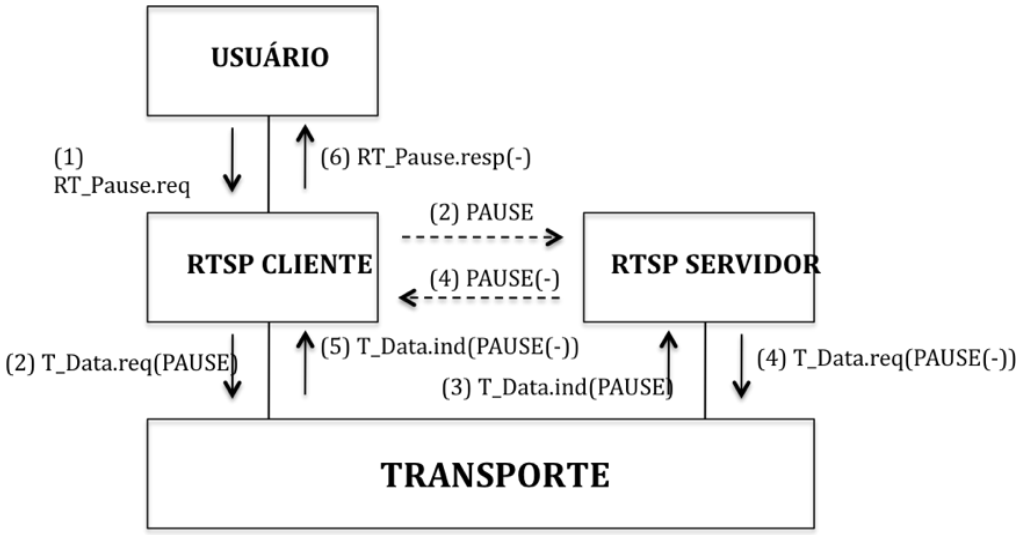


Figura 6.5 – Cenário de pedido de pausa na reprodução de mídias (não realizado)

4. Encerramento da sessão (servidor redirecionado)

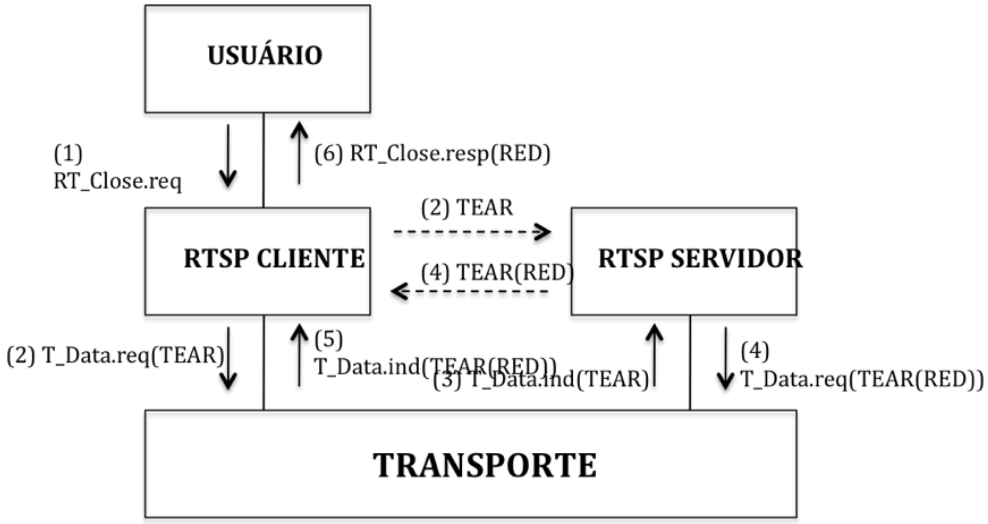


Figura 6.6 – Cenário de pedido de encerramento de sessão (com redirecionamento de servidor)

A especificação destes diagramas auxilia na compreensão do comportamento das entidades do protocolo em cada fase do serviço fornecido, facilitando a posterior definição das máquinas de estado. As tabelas de máquinas de estado do cliente e do servidor apresentadas a seguir são baseadas no que está definido na RFC [27]. Convém ressaltar, a esta altura, que, como este estudo tem como objetivo exercitar os conceitos discutidos de projeto de protocolos de comunicação e explorar as características de modelagem formal mais focada no aspecto comportamental do protocolo, através dos métodos Redes de Petri e Estelle, não iremos levar em consideração o tratamento de dados, como a inclusão e a análise dos diversos campos do cabeçalho das mensagens do protocolo descritos na norma, e o impacto destes em possíveis transições de estado.

Máquina de estados do cliente

Estado inicial	Mensagem enviada	Próximo estado
Repouso	SETUP	Pronto
	TEAR	Repouso
Pronto	PLAY	Ativo
	TEAR	Repouso
	SETUP	Pronto
Ativo	PAUSE	Pronto
	TEAR	Repouso
	PLAY	Ativo
	SETUP	Ativo (muda o transporte)

Tabela 6.2 – Máquina de estados do cliente RTSP

O cliente, em geral, muda de estado ao receber respostas dos pedidos enviados. A coluna de "próximo estado" indica o estado que a entidade assume após uma resposta bem sucedida (2xx). Caso um pedido retorne um código de status 3xx, a máquina de estado volta para o estado de repouso; enquanto que um código de status igual a 4xx resulta em nenhuma modificação no estado inicial. Receber uma mensagem RED (REDIRECT) do servidor tem o mesmo efeito que receber um código de status 3xx de redirecionamento do servidor.

Máquina de estados do servidor

Estado inicial	Mensagem recebida	Próximo estado
Repouso	SETUP	Pronto
	TEAR	Repouso
Pronto	PLAY	Ativo
	SETUP	Pronto
	TEAR	Repouso
Ativo	PLAY	Ativo
	PAUSE	Pronto
	TEAR	Repouso
	SETUP	Ativo

Tabela 6.3 – Máquina de estados do servidor RTSP

O servidor, por sua vez, modifica seu estado ao receber pedidos. A coluna de "próximo estado" indica o estado assumido após o envio de uma resposta bem sucedida (2xx).

Se um pedido resultar em um código de status de 3xx, então o próximo estado da máquina é o de repouso. Um código de status 4xx não modifica o estado inicial.

6.4 Modelagem e validação do comportamento das entidades com Redes de Petri

De posse desta especificação mais estruturada, "semi-informal", do serviço do RTSP, iremos aplicar métodos formais visando identificar ambigüidades e obter uma especificação clara, completa, e tratável; ou seja, uma especificação a qual se possa verificar as propriedades desejadas.

Desta forma, escolhemos as Redes de Petri, que são uma técnica de modelagem para verificação do comportamento especificado para as entidades do protocolo através de máquinas de estados. A transformação de máquinas de estados finitas em Redes de Petri é bastante direta, sendo os estados convertidos em lugares, e as condições para mudanças de estado em predicados para o disparo de transições.

O comportamento das entidades cliente e servidor do RTSP foi modelado em uma única Rede de Petri, explorando-se o fato de esta técnica ser uma extensão às máquinas de estados. Para tal, tomou-se o cuidado de modelar o comportamento do sistema de modo a se conseguir o encadeamento temporal das transições desejadas. Desta forma, foi possível modelar duas máquinas de estado separadamente (cliente e servidor) e, posteriormente, sincronizá-las.

Para a representação gráfica mostrada a seguir, utilizou-se de Redes Predicado-ação, que são uma extensão às Redes de Petri, sendo que uma para o cliente e outra para o servidor, visto que a representação em Redes de Petri ficaria graficamente confusa (24 lugares e 51 transições).

Rede Predicado-Ação para o Cliente e para o Servidor

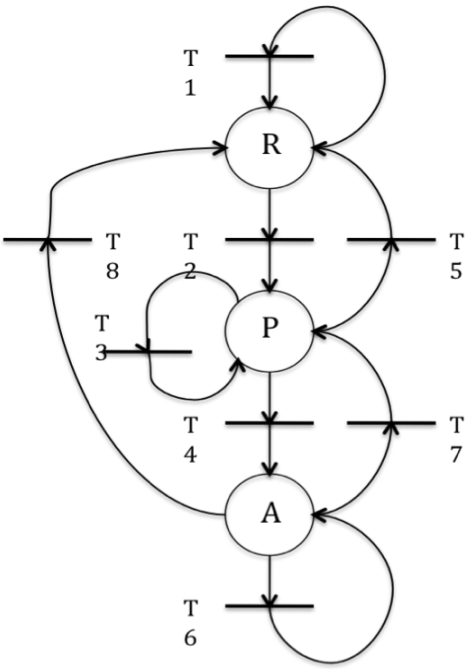


Figura 6.7 – Rede predicado-ação dos comportamentos cliente e servidor do RTSP

Tabela de transições para o comportamento Cliente:

Transição	Predicado/Ação correspondente
T1	?TR_Desc.req|!DESC ?TR_Open.req|!SETUP ?TR_Close.req|!TEAR ?DESC(+)|!TR_Desc.resp(+), ?DESC(-)!TR_Desc.resp(-) ?SETUP(-)|TR_Open.resp(-) ?TEAR(+)|!TR_Close.resp(+), ?TEAR(-)!TR_Close.resp(-)
T2	?SETUP(+)|!TR_Open.resp(+)
T3	?TR_Play.req|!PLAY ?TR_Open.req|!SETUP ?TR_Close.req|!TEAR ?PLAY(-)!TR_Play.resp(-) ?SETUP(+)|!TR_Open.resp(+), ?SETUP(-)|TR_Open.resp(-) ?TEAR(-)!TR_Close.resp(-)
T4	?PLAY(+)|!TR_Play.resp(+)
T5	?TEAR(+)|!TR_Close.resp(+) ?RED|!RT_Tear.resp(RED)
T6	?TR_Play.req|!PLAY ?TR_Pause.req|!PAUSE ?TR_Open.req|!SETUP

	?TR_Close.req|!TEAR ?PLAY(+)|!TR_Play.resp(+), ?PLAY(-)!TR_Play.resp(-) ?PAUSE(-)|!RT_Pause.resp(-) ?SETUP(+)|!TR_Open.resp(+), ?SETUP(-)|TR_Open.resp(-) ?TEAR(-)!TR_Close.resp(-)
T7	?PAUSE(+)|!RT_Pause.resp(+) ?range_timeover
T8	?TEAR(+)|!RT_Close.resp(+) ?RED|!RT_Tear.resp(RED)

Tabela 6.4 – Transições para a rede predicado-ação do cliente RTSP

Tabela de transições para o comportamento Servidor:

Transição	Predicado/Ação correspondente
T1	?SETUP|!SETUP(-) ?DESC|!DESC(+), ?DESC|!DESC(-) ?TEAR|!TEAR(+), ?TEAR|!TEAR(-)
T2	?SETUP|!SETUP(+)
T3	?SETUP|!SETUP(+), ?SETUP|!SETUP(-) ?PLAY|!PLAY(-) ?TEAR|!TEAR(-)
T4	?PLAY|!PLAY(+)
T5	?TEAR|!!TEAR(+), ?Redirect_timeout|!RED
T6	?SETUP|!SETUP(+), ?SETUP|!SETUP(-) ?PLAY|!PLAY(+), ?PLAY|!PLAY(-) ?PAUSE|!PAUSE(-) ?TEAR|!TEAR(-)
T7	?PAUSE|!PAUSE(+) ?range_timeover
T8	?TEAR|!TEAR(+) ?Redirect_timeout|!RED

Tabela 6.5 – Transições para a rede predicado-ação do servidor RTSP

O predicado “range_timeover” foi incluído para representar a mudança de estado de ativo para pronto, tanto para o cliente como para o servidor, quando o tempo total de reprodução da apresentação solicitada for alcançado. Já o predicado “Redirect_timeout” foi definido para representar as condições que levam o servidor a requisitar um redirecionamento

de servidor de mídia, seja através do envio do método REDIRECT ou do código de status 3xx a algum pedido do cliente.

A especificação desta rede no formato utilizado pelo ARP (Analisador de Redes de Petri) é razoavelmente complexa, conforme pode ser visualizada no Apêndice I. Com o objetivo de tornar a especificação e os resultados mais intuitivos, foi adotada uma nomenclatura para os lugares e as transições da rede de acordo com o nome dos estados e dos métodos do protocolo, conforme definido na RFC.

A especificação foi submetida à análise do ARP, e os resultados obtidos encontram-se no Apêndice II (a rede foi chamada de "RTSP_basico"). A rede atendeu a todas as boas propriedades gerais (inclusive a de rede conservativa). Isto garante que esta rede não trava em nenhuma hipótese, que sempre retorna à marcação inicial e que mantém constante o somatório do número de fichas em todos os seus lugares. Mais do que isso, todos os lugares desta rede são binários; ou seja, ou contém uma ficha ou não contém ficha alguma para qualquer marcação. Estas propriedades garantem que as mensagens não se acumulam em nenhum estado das entidades do protocolo e, também, que os pedidos do serviço são sempre respondidos. Em relação às propriedades específicas, os resultados mostram a existência de invariantes de lugar, onde se podem destacar os invariantes IL2 e IL3, que representam a máquina de estados do servidor e do cliente respectivamente. O invariante linear IL1 representa a máquina de estados que indica o fluxo de mensagens entre cliente e servidor, garantindo a exclusão mútua entre pedidos de métodos e entre as respectivas respostas positivas e negativas a cada um destes, além de garantir que o cliente está em algum estado definido ou há alguma mensagem em trânsito. Vale a pena ressaltar que todos os resultados apresentados neste trabalho foram retirados diretamente do arquivo gerado pelo ARP sem nenhuma modificação (vide Apêndice II).

6.5 Especificação do protocolo RTSP em Estelle

Uma vez verificadas formalmente as propriedades desejadas do comportamento das entidades que compõem o protocolo RTSP, seguimos com a especificação formal de todo o modelo em camadas hierárquicas do serviço através da linguagem Estelle, que permite uma tradução quase direta do sistema em uma estrutura hierárquica de autômatos que podem ser executados em paralelo e que podem se comunicar através de troca de mensagens e compartilhamento de variáveis.

A arquitetura da especificação em Estelle do sistema é apresentada no diagrama abaixo.

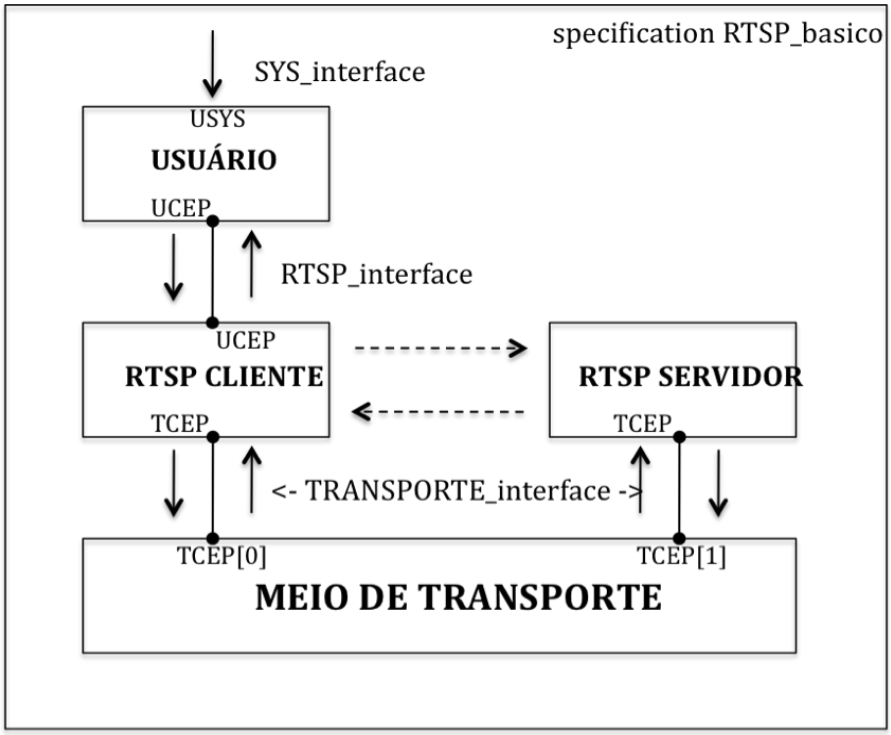


Figura 6.8 – Arquitetura da especificação em Estelle

Nesta figura, podemos ver os módulos propostos para o modelo do serviço, seus respectivos pontos de interação e os tipos de canais de comunicação, de acordo com as construções da linguagem Estelle. Para cada um dos módulos, o comportamento interno, apresentado anteriormente através das máquinas de estados finitos (MEFs), é descrito em código Estelle. Além destes, as interfaces de comunicação são descritas em detalhes, conforme preconiza esta TDF.

Inicialmente, na especificação, após a definição de alguns tipos de variáveis globais, são descritos os canais de comunicação, que apresentam a lista de primitivas de serviço trocadas em ambos sentidos naquela interface de comunicação. Para a identificação das mensagens enviadas em cada sentido no canal em questão, elas são separadas no código utilizando os parâmetros *user* e *provider*.

Um total de três tipos de canais foram especificados, um para a comunicação com cada camada do modelo – SYS (usuário), RTSP e transporte -, conforme abaixo:

```
channel SYS_interface(user,provider);
  by user :
    RT_DESCRIBE;
    RT_SETUP;
```

```
    RT_PLAY;
    RT_PAUSE;
    RT_CLOSE;

channel RTSP_interface(user,provider);
  by user :
    DescReq;
    OpenReq(descricao:frase);
    PlayReq;
    PauseReq;
    CloseReq;
  by provider :
    DescResp(status:integer;descricao:frase);
    OpenResp(status:integer);
    PlayResp(status:integer);
    PauseResp(status:integer);
    CloseResp(status:integer);

channel TRANSPORTE_interface(user,provider);
  by user :
    DataReq(PDU:PDU_tipo);
  by provider :
    DataInd(PDU:PDU_tipo);
```

Na sequência, os tipos de módulos a serem usados são descritos, onde definem-se os pontos de interação externos do módulo associando a estes o tipo de canal de comunicação a ser usado e o sentido das mensagens enviadas neste. São definidos, então, três tipos de módulos – USER, RTSP e TRANSPORTE:

```
module USER_TYPE activity;
  ip
    USYS : SYS_interface(provider);
    UCEP : RTSP_interface(user);
end;

module RTSP_TYPE activity;
  ip
    UCEP : RTSP_interface(provider);
    TCEP : TRANSPORTE_interface(user);
end;

module TRANSPORTE_TYPE activity;
  ip
    TCEP_0 : TRANSPORTE_interface(provider);
    TCEP_1 : TRANSPORTE_interface(provider);
end;
```

Passamos, então, à definição de comportamento para cada tipo de módulo declarado. Para tal, um "corpo" de módulo é descrito a partir da cláusula *body*, indicando-se para qual tipo de módulo este se aplica.

Como a camada RTSP é o objeto de estudo deste trabalho, iremos explorar em mais detalhes a descrição de corpo feita para o módulo RTSP. Para os módulos das camadas de

usuário e de transporte, o comportamento descrito foi o mais simplificado possível de modo a garantir somente a especificação e a verificação do serviço da camada em estudo, observando-se o que foi previsto na RFC [27]. Mais detalhes podem ser encontrados na especificação completa em Estelle no Apêndice III.

A descrição do corpo para o tipo de módulo do RTSP começa com a definição de procedimentos para a montagem dos PDUs de pedido e de resposta trocados entre as entidades pares da camada. A variável PDU é criada a partir do tipo PDU_tipo, que é definido como um registro que contém três campos, com informações do método do serviço RTSP, de status do serviço solicitado e de um corpo (*string* para passar a descrição de mídia). Os métodos do RTSP estão definidos no tipo método_tipo a partir de uma lista enumerada.

Exemplo de par de procedimentos para montagem dos PDUs de pedido e resposta para o método PLAY:

```
procedure BuildPLAY(var PDU: PDU_tipo);
      begin
        PDU.metodo:=PLAY;
      end;
    procedure RespPLAY(var PDU: PDU_tipo; status:integer);
      begin
        PDU.metodo:=PLAY;
        PDU.status:=status;
      end;
```

A partir daí, inicia-se a descrição do comportamento da máquina de estados finita (MEF), definindo-se os estados possíveis e a lista de todas as transições mapeadas. Cabe ressaltar, neste ponto, que são descritas dentro deste único corpo as máquinas de estados do cliente (ou iniciador) e do servidor (ou respondedor) do protocolo RTSP.

Observa-se que os estados declarados neste ponto correspondem aos invariantes de lugar (IL2 e IL3) verificados na análise por Redes de Petri como máquinas de estados.

A tradução do modelo em MEF para a linguagem Estelle corresponderá ao mapeamento das transições listadas, a partir de um estado, conforme a rede de Petri representando a rede predicado-ação do comportamento da entidade RTSP.

Tomemos, como exemplo, a inicialização da máquina de estados para o estado de REPOUSO e as transições para o cliente RTSP, a partir deste, para o estado de ESPERA_REPOUSO; e, posteriormente, as transições do estado de ESPERA_REPOUSO para o estado de REPOUSO ou PRONTO, de acordo com os métodos invocados e o status das respostas recebidas.

```
initialize
```

```
  to REPOUSO
  name init_body :
    begin
    end;

trans
    from REPOUSO
      when UCEP.DescReq
          to ESPERA_REPOUSO
            name CliRmanDESC :
            begin
              BuildDESC(PDU_aux);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from REPOUSO
      when UCEP.OpenReq
          to ESPERA_REPOUSO
            name CliRmanSETUP :
            begin
              BuildSETUP(PDU_aux);
              PDU_aux.corpo:=descricao;
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from REPOUSO
      when UCEP.CloseReq
          to ESPERA_REPOUSO
            name CliRmanTEAR :
            begin
              BuildTEAR(PDU_aux);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from ESPERA_REPOUSO
      when TCEP.DataInd
        provided Metodo(PDU)=DESC
          to REPOUSO
            name CliRrecDESC :
            begin
              status_aux:=Status(PDU);
              descricao_aux:=PDU.corpo;
              output UCEP.DescResp(status_aux,descricao_aux);
            end;

trans
    from ESPERA_REPOUSO
      when TCEP.DataInd
        provided Metodo(PDU)=TEAR
          to REPOUSO
            name CliRrecTEAR :
            begin
              status_aux:=Status(PDU);
              output UCEP.CloseResp(status_aux);
            end;

trans
    from ESPERA_REPOUSO
```

```
          when TCEP.DataInd
            provided ( Metodo(PDU)=SETUP ) and ( (Status(PDU)=4) or
(Status(PDU)=3) )
              to REPOUSO
                name CliRrecSETUPN :
                begin
                  status_aux:=Status(PDU);
                  output UCEP.OpenResp(status_aux);
                end;

    trans
        from ESPERA_REPOUSO
          when TCEP.DataInd
            provided (Metodo(PDU)=SETUP) and (Status(PDU)=2)
              to PRONTO
                name CliRrecSETUP :
                begin
                  output UCEP.OpenResp(2);
                end;
```

Confirmando o comportamento definido na RFC [27] e descrito na MEF, somente o método SETUP é capaz de modificar o estado da MEF do cliente a partir do estado de repouso, caso seja executado com sucesso pelo servidor – quando retorna o código de status ‘2xx’ (simplificado para 2 na especificação).

Com relação à máquina de estados do servidor a partir do estado de REPOUSO, acrescentamos o estado ESPERA_REPOUSO, que não consta na rede de Petri, para que o usuário que executa a simulação da especificação em Estelle possa escolher a resposta do servidor ao método solicitado. Também foram incluídos estados de espera para os estados PRONTO e ATIVO para este mesmo fim no restante da especificação da MEF do comportamento de servidor. Estas modificações são transparentes para fins de verificação das propriedades desejadas, pois o invariante de lugar IL1, que garante exclusão mútua entre as possíveis respostas aos métodos e os estados do servidor conforme visto na análise da rede de Petri, não é ameaçado. Foi apenas uma escolha de desenvolvimento para a especificação, já que a definição de qual status o servidor vai retornar para um método envolve a parte de tratamento de dados que não é abordada neste trabalho.

```
trans
        from REPOUSO
          when TCEP.DataInd
              to ESPERA_REPOUSO
                name ServRrecPDU :
                begin
                  PDU_serv:= PDU;
                  WRITELN;
                  WRITESTRING(' Entre com o status da resposta (2/3/4) :');
                  READINT(status_aux);
                end;
```

```
    trans
        from ESPERA_REPOUSO
            provided Metodo(PDU_serv)=DESC
              to REPOUSO
                name ServRenvDESC :
                begin
                  RespDESC(PDU_aux,status_aux);
                  PDU_aux.corpo:=exemplo;
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_REPOUSO
            provided (Metodo(PDU_serv)=SETUP) and (status_aux=2)
              to PRONTO
                name ServRenvSETUP :
                begin
                  RespSETUP(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_REPOUSO
            provided (Metodo(PDU_serv)=SETUP) and ( (status_aux=3) or
(status_aux=4) )
              to REPOUSO
                name ServRenvSETUPN :
                begin
                  RespSETUP(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);

                end;
```

Por fim, as variáveis módulo são declaradas associando-se o tipo de módulo respectivo de cada uma delas. Dentro da cláusula *initialize*, cada módulo declarado é iniciado com um corpo de módulo escolhido dentre os declarados na especificação e os pontos de interação externos dos módulos são conectados entre si através da clásula *connect*, criando a estrutura hierárquica da especificação.

```
modvar
    USER_A : USER_TYPE;
    RTSP_CLIENTE : RTSP_TYPE;
    RTSP_SERVIDOR : RTSP_TYPE;
    MEIO_DE_TRANSPORTE : TRANSPORTE_TYPE;

  initialize
    name init_modules :
      begin
        init USER_A with USER_BODY;
        init RTSP_CLIENTE with RTSP_BODY;
        init RTSP_SERVIDOR with RTSP_BODY;
        init MEIO_DE_TRANSPORTE with TRANSPORTE_BODY;
        connect USER_A.UCEP to RTSP_CLIENTE.UCEP;
        connect RTSP_CLIENTE.TCEP to MEIO_DE_TRANSPORTE.TCEP_0;
        connect MEIO_DE_TRANSPORTE.TCEP_1 to RTSP_SERVIDOR.TCEP;
```

```
end;
```

6.6 Simulação usando o SPDE

Concluída a especificação do modelo desenhado na linguagem Estelle, utilizou-se o conjunto de ferramentas CAD-Estelle para verificar a sintaxe, a semântica e as propriedades esperadas do sistema especificado.

Através da ferramenta compilador (CE), a análise léxica, sintática e de contexto da especificação é feita e, por fim, a Forma Intermediária é gerada para uso pelas demais ferramentas.

Os resultados obtidos ao submeter a especificação ao compilador Estelle são apresentados no terminal conforme abaixo.

```
     Carregando tabelas de ESTELLE ...
Carregamento das tabelas de tabelas.dat completado.

**********************************************************************

Qual o nome do arquivo ? rtsp2
 1105        linhas compiladas .

So a compilacao gastou aproximadamente  0 min 0 seg e 5 centesimos .

NAO HA ERROS.
```

Após a compilação da especificação sem erros, procedeu-se com a validação da propriedades da especificação; ou seja, verificar se todos os requisitos e características do sistema, por esta descrito, refletem o que foi definido informalmente ou através de técnicas semi-formais. A ferramenta do simulador de protocolos do CAD-Estelle, o SPDE, permite a validação por meio da detecção de erros na especificação através da execução controlada desta.

A entidade usuário da especificação tem como papel possibilitar a interação com o sistema alvo da simulação, fornecendo uma interface que possibilita controlar o envio de primitivas (entradas) e observar as respostas recebidas (saídas). Os testes, portanto, consistem em enviar uma primitiva ao sistema através da interface do usuário e, após o seu processamento, analisar o resultado comparando a resposta esperada para aquela situação.

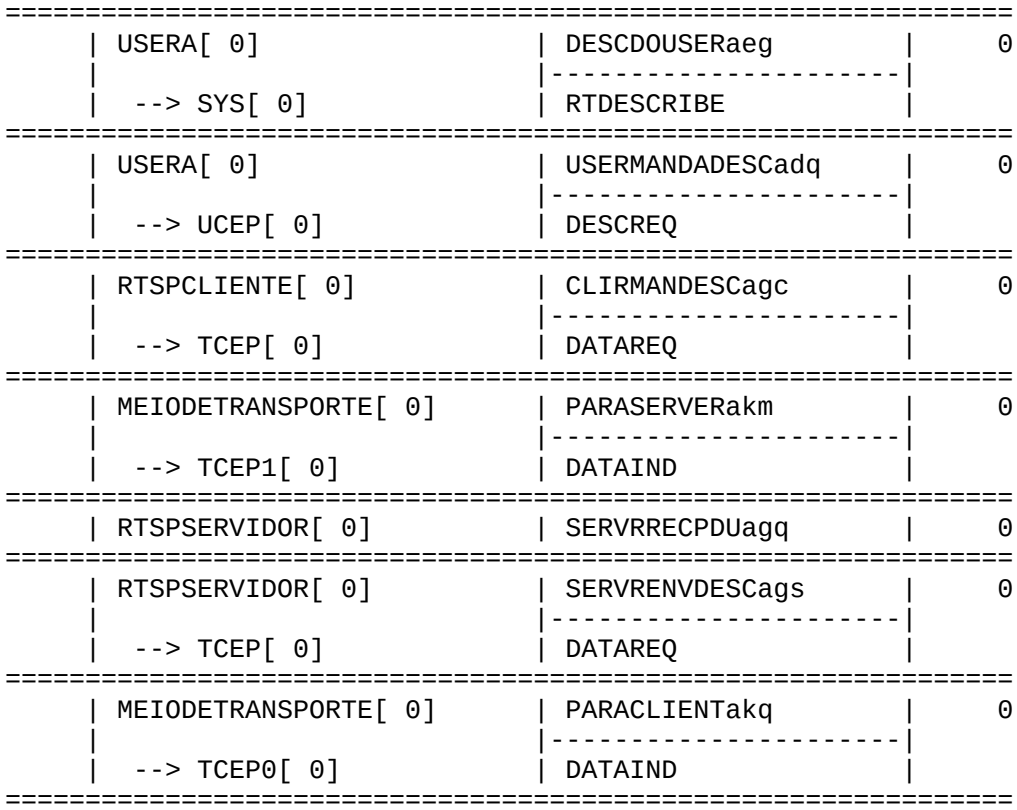
A simulação do serviço envolve a análise dos comportamentos das MEFEs das entidades envolvidas durante o funcionamento da arquitetura, avaliando tanto o

comportamento individual quanto o comportamento conjunto destas. O simulador disponibiliza acesso às interfaces para inserção de interações, à estrutura interna da especificação – verificar e alterar estados de instâncias e de variáveis -, e também simular passos aleatórios, execução passo-a-passo do modelo, reinicializar a simulação, entre outras facilidades. Este acompanhamento da execução do serviço viabiliza a localização mais precisa do ponto onde ocorreu um erro e a identificação mais direta de sua causa.

A estratégia da simulação se baseia em definir um conjunto abrangente de cenários de testes a serem exercitados a partir do simulador, onde cada cenário corresponde à execução, bem sucedida ou não, de uma operação (ou método) do serviço modelado. Desta forma, define-se, para cada cenário, um fluxo de transições em número restrito, facilitando a análise e a identificação do ponto de ocorrência de erros. Para cada operação, o comportamento observado das entidades envolvidas deve ser analisado e comparado com o previsto na especificação e na RFC.

Com base no que foi exposto, realizaram-se testes exaustivos de modo a validar e, ocasionalmente, depurar o código obtido do modelo especificado, coletando-se traços gerados em cada módulo ao longo da execução do serviço, de acordo com os cenários de testes apresentados a seguir.

1. Solicitação de Descrição de mídia para o servidor RTSP

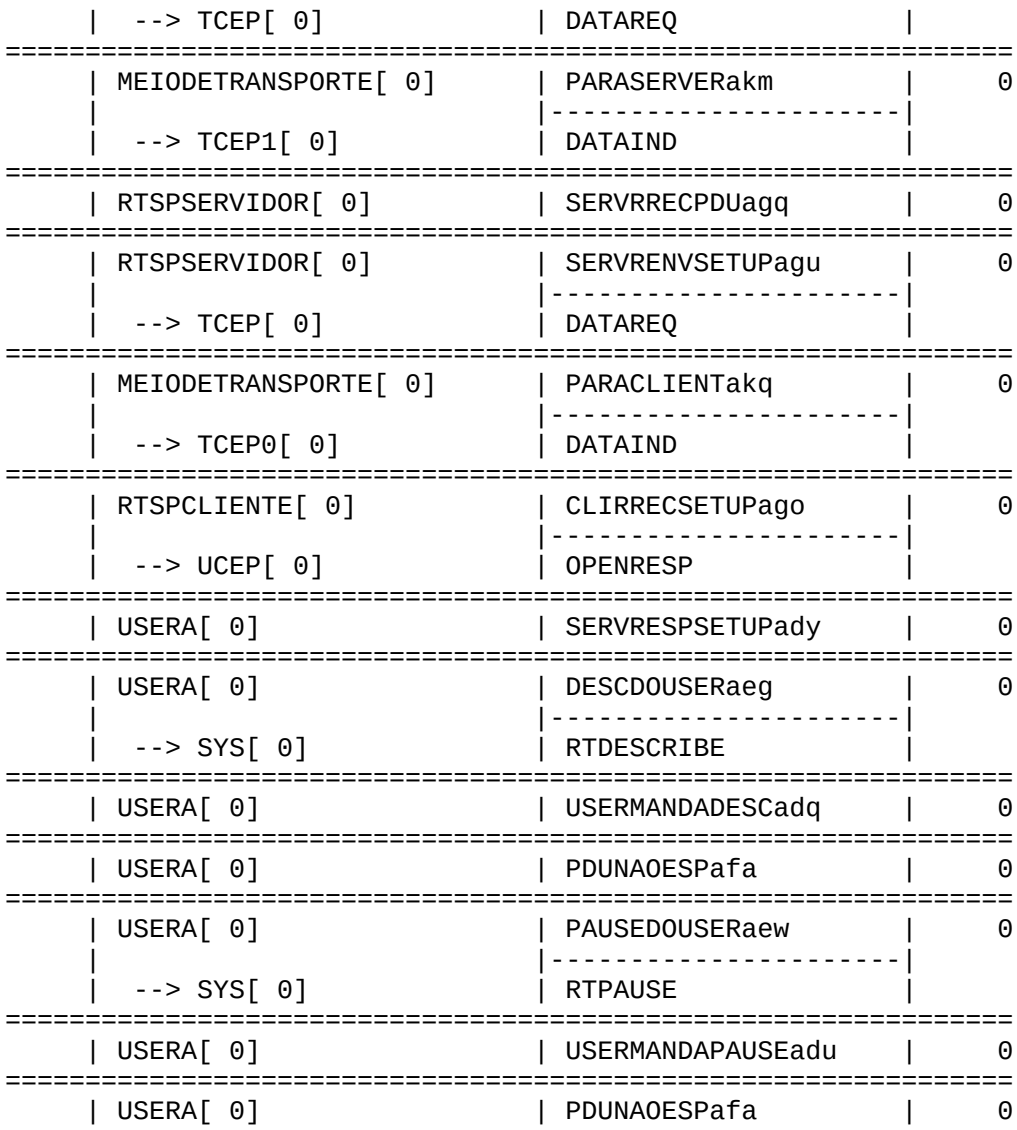


```
     | RTSPCLIENTE[ 0]           | CLIRRECDESCagi       |     0
     |                           |----------------------|
     |  --> UCEP[ 0]             | DESCRESP             |
=============================================================
     | USERA[ 0]                 | SERVRESPDESCaee      |     0
=============================================================
```

Neste cenário, simulamos o envio do método DESCRIBE pelo cliente RTSP, observando todo o fluxo de mensagens esperado até o recebimento da PDU do tipo *response* relativo a este método pela entidade RTSP CLIENTE, sendo então repassada a resposta ao usuário USERA, conforme pode ser observado nos traços acima.

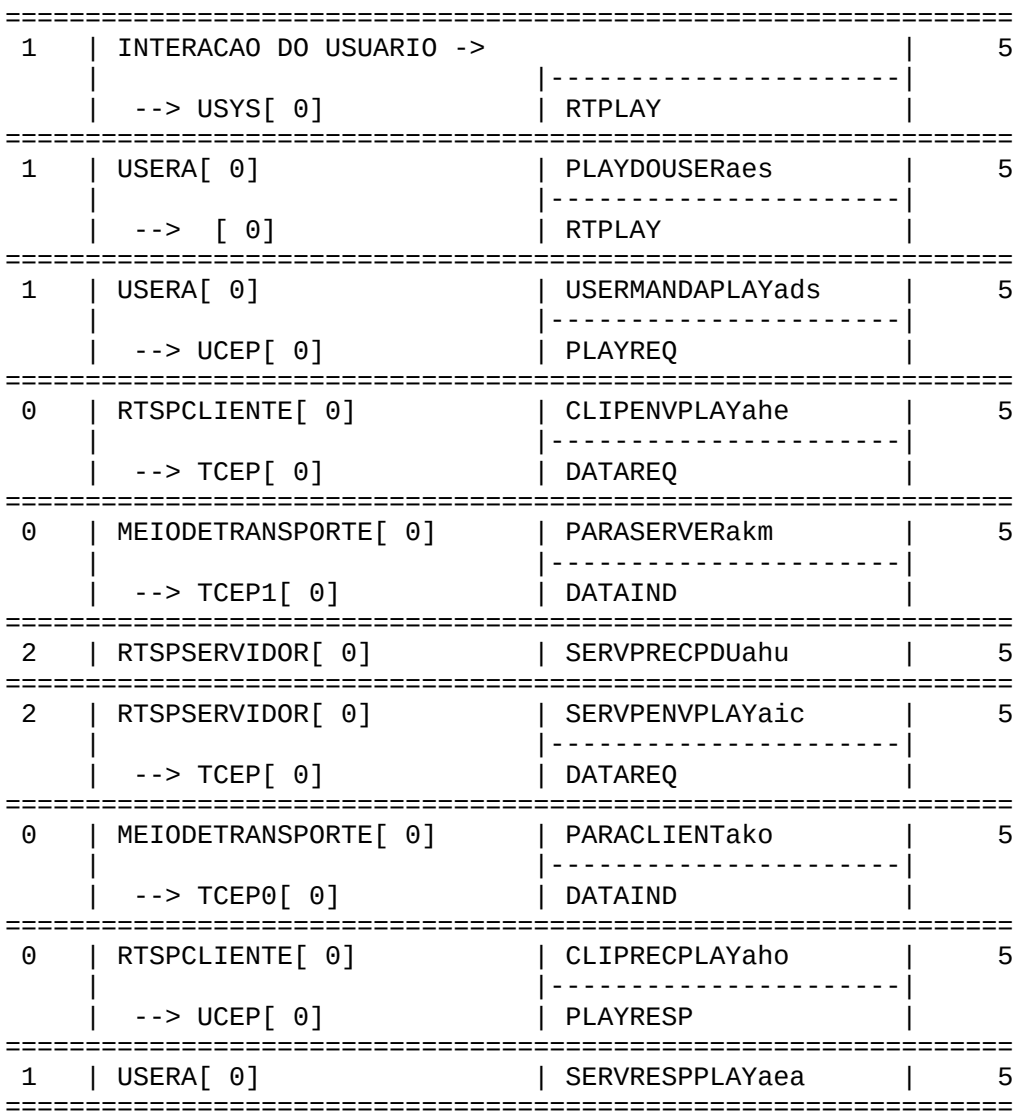
2. Estabelecimento de sessão de mídia no servidor

```
=============================================================
     | USERA[ 0]                 | SETUPDOUSERaek       |     0
     |                           |----------------------|
     |  --> SYS[ 0]              | RTSETUP              |
=============================================================
     | USERA[ 0]                 | USERMANDASETUPado    |     0
     |                           |----------------------|
     |  --> UCEP[ 0]             | OPENREQ              |
=============================================================
     | RTSPCLIENTE[ 0]           | CLIRMANSETUPage      |     0
     |                           |----------------------|
     |  --> TCEP[ 0]             | DATAREQ              |
=============================================================
     | MEIODETRANSPORTE[ 0]      | PARASERVERakm        |     0
     |                           |----------------------|
     |  --> TCEP1[ 0]            | DATAIND              |
=============================================================
     | RTSPSERVIDOR[ 0]          | SERVRRECPDUagq       |     0
=============================================================
     | RTSPSERVIDOR[ 0]          | SERVRENVSETUPNagw    |     0
     |                           |----------------------|
     |  --> TCEP[ 0]             | DATAREQ              |
=============================================================
     | MEIODETRANSPORTE[ 0]      | PARACLIENTakq        |     0
     |                           |----------------------|
     |  --> TCEP0[ 0]            | DATAIND              |
=============================================================
     | RTSPCLIENTE[ 0]           | CLIRRECSETUPNagm     |     0
     |                           |----------------------|
     |  --> UCEP[ 0]             | OPENRESP             |
=============================================================
     | USERA[ 0]                 | SERVRESPSETUPady     |     0
=============================================================
     | USERA[ 0]                 | SETUPDOUSERaek       |     0
     |                           |----------------------|
     |  --> SYS[ 0]              | RTSETUP              |
=============================================================
     | USERA[ 0]                 | USERMANDASETUPado    |     0
     |                           |----------------------|
     |  --> UCEP[ 0]             | OPENREQ              |
=============================================================
     | RTSPCLIENTE[ 0]           | CLIRMANSETUPage      |     0
     |                           |----------------------|
```



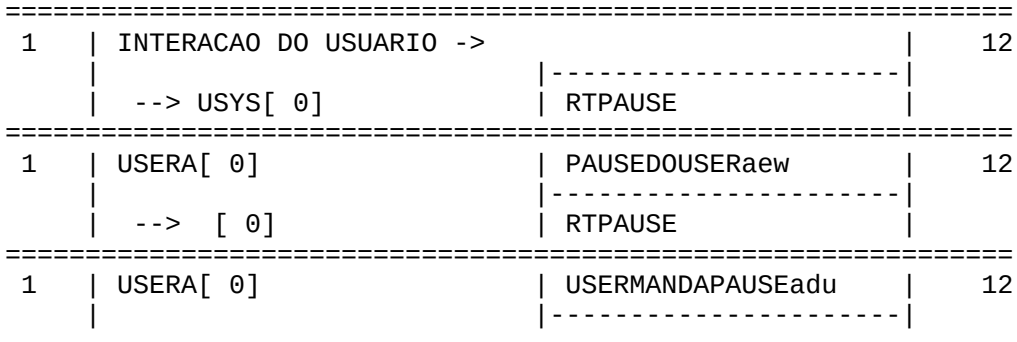
Neste conjunto de traços, explorou-se no simulador o comportamento da especificação considerando algumas tentativas frustradas de estabelecimento de sessão até o recebimento de uma resposta positiva (status 2xx). Também verificou-se o comportamento quando são enviadas pelo usuário solicitações de serviços não esperadas para o estado em questão da entidade RTSP CLIENTE.

3. Início de reprodução de mídia



O fluxo normal para uma solicitação de início de reprodução de mídia é verificado na sequência de mensagens trocadas entre as entidades da especificação.

4. Pausa na reprodução de mídia



```
     |  --> UCEP[ 0]              | PAUSEREQ              |
=============================================================
 0   | RTSPCLIENTE[ 0]            | CLIAENVPAUSEaio       |    12
     |                            |-----------------------|
     |  --> TCEP[ 0]              | DATAREQ               |
=============================================================
 0   | MEIODETRANSPORTE[ 0]       | PARASERVERakm         |    12
     |                            |-----------------------|
     |  --> TCEP1[ 0]             | DATAIND               |
=============================================================
 2   | RTSPSERVIDOR[ 0]           | SERVARECPDUajk        |    12
=============================================================
 2   | RTSPSERVIDOR[ 0]           | SERVAENVPAUSEaka      |    12
     |                            |-----------------------|
     |  --> TCEP[ 0]              | DATAREQ               |
=============================================================
 0   | MEIODETRANSPORTE[ 0]       | PARACLIENTako         |    12
     |                            |-----------------------|
     |  --> TCEP0[ 0]             | DATAIND               |
=============================================================
 0   | RTSPCLIENTE[ 0]            | CLIARECPAUSEaja       |    12
     |                            |-----------------------|
     |  --> UCEP[ 0]              | PAUSERESP             |
=============================================================
 1   | USERA[ 0]                  | SERVRESPPAUSEaec      |    12
=============================================================
```

Assim como no cenário para iniciar a reprodução da sessão de mídia, exploramos aqui o fluxo de mensagens para uma solicitação de pausa através do método PAUSE em que a resposta recebida é positiva.

5. Encerramento de sessão de mídia

```
=============================================================
 1   | INTERACAO DO USUARIO ->                         |    29
     |                            |-----------------------|
     |  --> USYS[ 0]              | RTCLOSE               |
=============================================================
 1   | USERA[ 0]                  | TEARDOUSERaeo         |    29
     |                            |-----------------------|
     |  -->  [ 0]                 | RTCLOSE               |
=============================================================
 1   | USERA[ 0]                  | USERMANDATEARadm      |    29
     |                            |-----------------------|
     |  --> UCEP[ 0]              | CLOSEREQ              |
=============================================================
 0   | RTSPCLIENTE[ 0]            | CLIPENVTEARahc        |    29
     |                            |-----------------------|
     |  --> TCEP[ 0]              | DATAREQ               |
=============================================================
 0   | MEIODETRANSPORTE[ 0]       | PARASERVERakm         |    29
     |                            |-----------------------|
     |  --> TCEP1[ 0]             | DATAIND               |
=============================================================
 2   | RTSPSERVIDOR[ 0]           | SERVPRECPDUahu        |    29
=============================================================
 2   | RTSPSERVIDOR[ 0]           | SERVPENVTEARahy       |    29
     |                            |-----------------------|
```

```
     |  --> TCEP[ 0]             | DATAREQ              |
=============================================================
 0   | MEIODETRANSPORTE[ 0]      | PARACLIENTako        |    29
     |                           |----------------------|
     |  --> TCEP0[ 0]            | DATAIND              |
=============================================================
 0   | RTSPCLIENTE[ 0]           | CLIPRECTEARahi       |    29
     |                           |----------------------|
     |  --> UCEP[ 0]             | CLOSERESP            |
=============================================================
 1   | USERA[ 0]                 | SERVRESPTEARadw      |    29
```

Por fim, para verificar como as entidades cliente e servidor do RTSP se comportam uma vez solicitado o encerramento da sessão, disparamos através da entidade usuário o serviço RTCLOSE, que dispara o método TEAR no cliente RTSP. O resultado apurado nos traços coletados confirma o fluxo de mensagens esperado entre as entidades da especificação.

Capítulo 7

Conclusões

Este trabalho, a partir da motivação inicial que foi a migração da ferramenta de simulação do CAD-Estelle para o ambiente Linux, buscou estudar o projeto de serviços para redes de comunicação a partir da metodologia apresentada e experimentar todo o processo a partir de um estudo de caso simples, porém que permitisse explorar os conceitos principais envolvidos.

Já durante a preparação e a execução da migração do código do simulador do ambiente computacional SunOS/Sparc para o Linux/x86, pôde-se observar, na prática, a importância da estruturação de um adequado ambiente de desenvolvimento para o trabalho com alguma notação de programação – que, no caso, era a linguagem Modula-2. A escolha de um adequado conjunto de ferramentas, tais como um editor com reconhecimento de sintaxe, um compilador com bom desempenho e que ofereça facilidades de compilação, e integração com uma ferramenta de depuração do programa flexível e com interface amigável, é imprescindível para lidar com sistemas de médio e grande porte. Buscou-se propor um ambiente com uma interface amigável e com o máximo de integração possível entre as ferramentas, unindo edição, compilação e depuração numa mesma interface.

Devido à característica de desenvolvimento modular oferecida pelo Modula-2, a estratégia de isolar dependências de serviços de bibliotecas do sistema potencialmente vinculadas à plataforma computacional, aplicada em boa parte do desenvolvimento original do código do simulador, tornou a migração do código mais direcionada. Os módulos que implementavam procedimentos, funções e tipos a partir do conjunto de bibliotecas do compilador para a plataforma operacional em desuso (SunOS), tais como os de manipulação de strings, funções de I/O e funções matemáticas e de conversões numéricas, foram o objeto do esforço inicial. Em função desta análise por grupo de funcionalidades, ficou facilitada a tarefa de buscar a compatibilidade entre as bibliotecas dos compiladores dos sistemas de origem (SunOS) e alvo (Linux), e assim evitar ter que simplesmente implementar os serviços não prontamente encontrados através de uma outra linguagem (C, por exemplo) e disponibilizá-los através de um módulo estrangeiro – recurso disponibilizado pelo Modula-2.

Na etapa dos testes do simulador, já com o código compilado sem erros para o ambiente Linux, ficou evidente, novamente, a importância da questão de um ambiente de desenvolvimento integrado e amigável para suporte ao projeto com uma linguagem – desta vez, a técnica de descrição formal Estelle. Para os testes com o protocolo Abracadabra, além de não contar com nenhuma facilidade de edição, foi necessário utilizar o compilador CE do CAD-Estelle na sua versão para o sistema operacional DOS. A falta de integração entre as ferramentas de compilação e simulação da especificação Estelle é flagrante, sendo necessário, inclusive, renomear um dos arquivos da Forma Intermediária (FIModula.mod) para a correta compilação deste com o código do simulador para a execução da simulação da especificação. Ou seja, a definição e a implementação de uma arquitetura para o CAD-Estelle em um ambiente computacional atual – como o Linux/x86 -, propondo a integração das ferramentas em torno de uma interface mais amigável do que as originalmente propostas para o ambiente DOS (e até mesmo SunOS), são fundamentais para recuperar este conjunto de ferramentas e proporcionar comodidade ao usuário-projetista, seguindo a linha da proposta do ProtCAD [4].

O estudo de caso com o protocolo RTSP, cuja aplicação se encontra cada vez mais difundida em função da crescente popularidade de serviços de assinatura de streaming de conteúdo multimídia via Internet (Netflix, Hulu entre outros), permitiu vivenciar o processo de especificação e verificação formais a partir de uma descrição informal em formato de texto – a RFC 2326 [27] – do protocolo em questão.

Ao aplicar os conceitos e a terminologia propostas pelo modelo OSI da ISO, de modo a traduzir a especificação da RFC em uma descrição mais estruturada, obteve-se um modelo mais próximo do cenário de implementação do protocolo, porém ainda semi-informal. Neste ponto, pôs-se em prática, de acordo com o contexto do estudo deste trabalho, o uso de métodos formais orientados ao comportamento do sistema modelado, focando na especificação e verificação das máquinas de estado das entidades cliente e servidor do RTSP. A tradução da tabela de estados da RFC usando a modelagem de Redes de Petri leva a uma preocupação constante em avaliar as ambigüidades durante o processo de especificação. Como já dito durante o estudo sobre os métodos formais, estes não são uma garantia de correção, mas aumentam a precisão, concisão e não-ambigüidade da especificação. O processo de verificação foi realizado com o suporte da ferramenta ARP, que através de uma interface simples, porém integrada, permitiu compilar o modelo e verificar as propriedades desejadas das redes de Petri do comportamento iniciador e respondedor do protocolo.

A especificação formal em Estelle do modelo em camadas do serviço do RTSP permitiu descrever com mais rigor as interfaces de comunicação das entidades (ou módulos,

na notação do Estelle), definindo os pontos de interação, tipos de canais e grupos de primitivas respectivos de cada um, assim como a hierarquia das entidades comunicantes. Mais uma vez ressaltamos que não era objetivo deste trabalho descrever as estruturas de dados para a montagem das mensagens do protocolo RTSP (cabeçalho e seus diversos campos, e o formato da mensagem), apesar dos recursos de tratamento de dados herdados do Pascal pela linguagem Estelle.

A simulação da especificação permitiu explorar as facilidades de execução do modelo oferecidas pela ferramenta e testar diversos cenários de operação do protocolo RTSP, tentando disparar ao menos uma vez todas as transições previstas nas máquinas de estados previamente verificadas.

Com relação aos resultados obtidos a partir da especificação e verificação do protocolo RTSP com o suporte do sistema de auxílio ao projeto de protocolos do CAD-Estelle, ficam como sugestão de trabalhos futuros o refinamento da especificação descrevendo a estrutura das mensagens do protocolo e inclusão de outros métodos previstos na RFC e, também, submeter o sistema especificado à simulação com uma sequência de testes exaustiva contando com o apoio da ferramenta do GASTPC [28] e, posteriormente, a uma implementação semi-automática através do implementador de protocolos descritos em Estelle [29]. Neste contexto, a migração e integração destas ferramentas numa nova arquitetura atualizada para o CAD-Estelle, conforme mencionado anteriormente, também é proposto.

Referências bibliográficas

[1] A. C. P. Pedroza, C. C. Goulart, P. R. O. Valim e R. de Oliveira, *"Um Sistema de Auxílio ao Projeto de Protocolos de Comuniação para Redes de Computadores"*, in *Seminário Franco-Brasileiro de Sistemas de Informação Distribuídos (SFBSID'89)*, Florianópolis, SC, Brasil, 1989.

[2] P. R. O. Valim, *"Um simulador de protocolos de comunicação para a linguagem Estelle"*, Tese de Mestrado, COPPE/PEE/UFRJ, abril 1990.

[3] R. de Carvalho Roman, *"Um editor visual para o auxílio a especificação de protocolos de comunicação com restrições temporais"*, Tese de Mestrado, COPPE/PEE/UFRJ, junho 2001.

[4] G. M. Delfino, J. V. dos Santos Filho, R. C. Roman, J. L. S. Leão e A. C. P. Pedroza, *"An integrated toolset for the design of protocols with QoS requirements (ProtCAD/RT)"*, in *Protocols for Multimedia Systems - PROMS'2000 Conference*, Krakow, Poland, 2000.

[5] Soares, L. F. G., Souza Filho, G. L., Colcher, S., "Redes de Computadores – Das LANs, MANs e WANs às Redes ATM", 2ª edição, Ed. Campus, 1995.

[6] Tanenbaum, A. S.: *Computer Networks. Third Edition*, Prentice-Hall International, 1996.

[7] M. G. M. Miranda, *"Modelagem e Análise Formal de algumas funcionalidades de um protocolo de transporte através das Redes de Petri"*, Tese de Mestrado, Inatel, dezembro de 2003.

[8] ISO, *"Estelle: A Formal Description Technique Based on an Extended State Transition Model"*, ISO IS 9074, 1989.

[9] U. Mantovan, *"Uma Abordagem, baseada em framework e na técnica de descrição formal Estelle, para o desenvolvimento de sistemas de arquivos distribuídos"*, Tese de Doutorado, Escola Politécnica/USP, 2006.

[10] ISO/IEC, *"Information Technology – Open System Interconnection – guidelines for the application of Estelle, LOTOS and SDL"*, Relatório Técnico ISO/IEC/TR 10167, 1991.

[11] ISO, *"ISO 7498: Information Processing Systems – Open Systems Interconnection – Basic Reference Model"*, 1984.

[12] ISO/IEC, *"Information technology -- Programming languages -- Part 1: Modula-2, Base Language"*, ISO/IEC 10514-1:1996.

[13] X. Logean, F. Dietrich, J.-P. Hubaux, S. Grisouard, P.-A. Etique, *"On Applying Formal Techniques to the Development of Hybrid Services: Challenges and Directions"*, IEEE Communications Magazine, vol. 37, no. 7, pp. 132–. 138, July 1999.

[14] Lamsweerde, Axel, *"Formal Specification: A Roadmap"*, Proceedings of the conference on The Future of Software Engineering, Finkelstein, A. (ed.) ACM Press, 2002.

[15] CCITT, *"CCITT Z.100 Specification and Description Language (SDL)"*, International Consultative Committee on Telegraphy and Telephony. 1988.

[16] ISO/IEC, *"Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behaviour"*, ISO/IEC 8807, International Organisation for Standardisation, Geneve 1989.

[17] Lewerentz, C. and Lindner, T., "*Formal Development of Reactive Systems - Case Study Production Cell"*, Springer-Verlag, Ed. 1995.

[18] Turner, K. J., "*Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL*", John Wiley & Sons, Ed. 1993.

[19] Murata, Tadao, *"Petri Nets: Properties, Analysis and Applications"*, Proceedings of the IEEE, Vol. 77, No. 4, April 1989.

[20] *ARP – Analisador de Redes de Petri*. Laboratório de Controle e de Microinformática do Departamento de Engenharia Elétrica da Universidade Federal de Santa Catarina, 1995

[21] Sun Microsystems, *"Using Modula-2 on the Sun Workstation"*, Version A, October 1986.

[22] Universitat Karlsruhe Manheim, Germany, *GMD Modula System Mocka*, 1999.

[23] Ford, Gary A., Wiener, Richard S.- "*Modula-2, a software development approach"*, John Wiley & Sons, 1985.

[24] Wirth, N., *"Programming in Modula-2"*, 3rd Edition, Springer-Verlag, 1985.

[25] GNU DDD. http://www.gnu.org/software/ddd/

[26] Kate editor. http://kate-editor.org/

[27] Schulzrinne, et. al., *"Real Time Streaming Protocol (RTSP)"*, RFC2326, Abril 1998.

[28] Silva, G. S., Pedroza, A. C. P., *"Um Sistema de Geração Automática de Sequências de Testes para Protocolos de Comunicação"*, Revista Brasileira de Computação, v. 5, n.1 pp. 33-44, 1989.

[29] S.C. de Alcântara, "*Implementação semi-automática de protocolos descritos em Estelle"*, Tese de Mestrado, PEL/COPPE/UFRJ, Rio de Janeiro, março de 1993.

[30] Oliveira, F. S. G., *"Uma Arquitetura para Suporte a Mobilidade e Segurança na Internet"*, Tese de Mestrado, COPPE/PEE/UFRJ, março de 1999.

[31] Fernandes, M. P., *"Protocolo para Gerenciamento Hierárquico de Redes de Telecomunicações"*, Tese de Mestrado, COPPE/PEE/UFRJ, janeiro de 1998.

Apêndice I - Especificação do comportamento das entidades cliente e servidor do RTSP na linguagem de entrada do ARP

```
Rede RTSP_basico;

NODOS

   Repouso_cliente, Repouso_servidor : lugar(1);
   Espera_repouso, Espera_pronto, Espera_ativo : lugar;
   Pronto_cliente, Pronto_servidor   : lugar;
   Ativo_cliente, Ativo_servidor     : lugar;
   Setup, Setup_ack, Setup_nack      : lugar;
   Describe, Describe_ack, Describe_nack : lugar;
   Tear, Tear_ack, Tear_nack         : lugar;
   Play, Play_ack, Play_nack         : lugar;
   Pause, Pause_ack, Pause_nack      : lugar;

   Manda_setup_repouso, Setup_ack_repouso, Setup_nack_repouso : transicao;
   Manda_desc_repouso, Desc_ack_repouso, Desc_nack_repouso : transicao;
   Manda_tear_repouso, Tear_ack_repouso, Tear_nack_repouso : transicao;
   Envia_setup_ack_rep, Envia_setup_nack_rep : transicao;
   Envia_desc_ack_rep, Envia_desc_nack_rep : transicao;
   Envia_tear_ack_rep, Envia_tear_nack_rep : transicao;
   Manda_setup_pronto, Setup_ack_pronto, Setup_nack_pronto : transicao;
   Manda_play_pronto, Play_ack_pronto, Play_nack_pronto : transicao;
   Manda_tear_pronto, Tear_ack_pronto, Tear_nack_pronto : transicao;
   Envia_setup_ack_p, Envia_setup_nack_p : transicao;
   Envia_play_ack_p, Envia_play_nack_p : transicao;
   Envia_tear_ack_p, Envia_tear_nack_p : transicao;
   Manda_setup_ativo, Setup_ack_ativo, Setup_nack_ativo : transicao;
   Manda_play_ativo, Play_ack_ativo, Play_nack_ativo : transicao;
   Manda_tear_ativo, Tear_ack_ativo, Tear_nack_ativo : transicao;
   Manda_pause_ativo, Pause_ack_ativo, Pause_nack_ativo : transicao;
   Envia_setup_ack_a, Envia_setup_nack_a : transicao;
   Envia_play_ack_a, Envia_play_nack_a : transicao;
   Envia_tear_ack_a, Envia_tear_nack_a : transicao;
   Envia_pause_ack_a, Envia_pause_nack_a : transicao;
   Timeout : transicao;

ESTRUTURA

   Manda_setup_repouso : ( Repouso_cliente ), ( Espera_repouso, Setup );
   Setup_ack_repouso : ( Espera_repouso, Setup_ack ), ( Pronto_cliente );
   Setup_nack_repouso : ( Espera_repouso, Setup_nack ),
( Repouso_cliente );
   Manda_desc_repouso : ( Repouso_cliente ), ( Espera_repouso, Describe );
   Desc_ack_repouso : ( Espera_repouso, Describe_ack ),
( Repouso_cliente );
   Desc_nack_repouso : ( Espera_repouso, Describe_nack ),
                       ( Repouso_cliente );
   Manda_tear_repouso : ( Repouso_cliente ), ( Espera_repouso, Tear );
   Tear_ack_repouso : ( Espera_repouso, Tear_ack ), ( Repouso_cliente );
   Tear_nack_repouso : ( Espera_repouso, Tear_nack ), ( Repouso_cliente );
   Envia_setup_ack_rep : ( Repouso_servidor, Setup ),
                         ( Setup_ack, Pronto_servidor );
   Envia_setup_nack_rep : ( Repouso_servidor, Setup ),
                          ( Setup_nack, Repouso_servidor );
```

```
   Envia_desc_ack_rep : ( Repouso_servidor, Describe ),
                        ( Describe_ack, Repouso_servidor );
   Envia_desc_nack_rep : ( Repouso_servidor, Describe ),
                         ( Describe_nack, Repouso_servidor );
   Envia_tear_ack_rep : ( Repouso_servidor, Tear ),
                        ( Tear_ack, Repouso_servidor );
   Envia_tear_nack_rep : ( Repouso_servidor, Tear ),
                         ( Tear_nack, Repouso_servidor );

   Manda_setup_pronto : ( Pronto_cliente ), ( Espera_pronto, Setup );
   Setup_ack_pronto : ( Espera_pronto, Setup_ack ), ( Pronto_cliente );
   Setup_nack_pronto : ( Espera_pronto, Setup_nack ), ( Pronto_cliente );
   Manda_play_pronto : ( Pronto_cliente ), ( Espera_pronto, Play );
   Play_ack_pronto : ( Espera_pronto, Play_ack ), ( Ativo_cliente );
   Play_nack_pronto : ( Espera_pronto, Play_nack ), ( Pronto_cliente );
   Manda_tear_pronto : ( Pronto_cliente ), ( Espera_pronto, Tear );
   Tear_ack_pronto : ( Espera_pronto, Tear_ack ), ( Repouso_cliente );
   Tear_nack_pronto : ( Espera_pronto, Tear_nack ), ( Pronto_cliente );
   Envia_setup_ack_p : ( Pronto_servidor, Setup ),
                       ( Setup_ack, Pronto_servidor );
   Envia_setup_nack_p : ( Pronto_servidor, Setup ),
                        ( Setup_nack, Pronto_servidor );
   Envia_play_ack_p : ( Pronto_servidor, Play ),
                      ( Play_ack, Ativo_servidor );
   Envia_play_nack_p : ( Pronto_servidor, Play ),
                       ( Play_nack, Pronto_servidor );
   Envia_tear_ack_p : ( Pronto_servidor, Tear ),
                      ( Tear_ack, Repouso_servidor );
   Envia_tear_nack_p : ( Pronto_servidor, Tear ),
                       ( Tear_nack, Pronto_servidor );

   Manda_setup_ativo : ( Ativo_cliente ), ( Espera_ativo, Setup );
   Setup_ack_ativo : ( Espera_ativo, Setup_ack ), ( Ativo_cliente );
   Setup_nack_ativo : ( Espera_ativo, Setup_nack ), ( Ativo_cliente );
   Manda_play_ativo : ( Ativo_cliente ), ( Espera_ativo, Play );
   Play_ack_ativo : ( Espera_ativo, Play_ack ), ( Ativo_cliente );
   Play_nack_ativo : ( Espera_ativo, Play_nack ), ( Ativo_cliente );
   Manda_tear_ativo : ( Ativo_cliente ), ( Espera_ativo, Tear );
   Tear_ack_ativo : ( Espera_ativo, Tear_ack ), ( Repouso_cliente );
   Tear_nack_ativo : ( Espera_ativo, Tear_nack ), ( Ativo_cliente );
   Manda_pause_ativo : ( Ativo_cliente ), ( Espera_ativo, Pause );
   Pause_ack_ativo : ( Espera_ativo, Pause_ack ), ( Pronto_cliente );
   Pause_nack_ativo : ( Espera_ativo, Pause_nack ), ( Ativo_cliente );
   Envia_setup_ack_a : ( Ativo_servidor, Setup ),
                       ( Setup_ack, Ativo_servidor );
   Envia_setup_nack_a : ( Ativo_servidor, Setup ),
                        ( Setup_nack, Ativo_servidor );
   Envia_play_ack_a : ( Ativo_servidor, Play ), ( Play_ack,
Ativo_servidor );
   Envia_play_nack_a : ( Ativo_servidor, Play ),
                       ( Play_nack, Ativo_servidor );
   Envia_tear_ack_a : ( Ativo_servidor, Tear ),
                      ( Tear_ack, Repouso_servidor );
   Envia_tear_nack_a : ( Ativo_servidor, Tear ),
                       ( Tear_nack, Ativo_servidor );
   Envia_pause_ack_a : ( Ativo_servidor, Pause ),
                       ( Pause_ack, Pronto_servidor );
   Envia_pause_nack_a : ( Ativo_servidor, Pause ),
                        ( Pause_nack, Ativo_servidor );

   Timeout : ( Ativo_cliente, Ativo_servidor ),
```

```
            ( Pronto_cliente, Pronto_servidor );

FIM.
```

Apêndice II - Resultado da verificação da especificação em Redes de Petri das entidades cliente e servidor do RTSP usando o ARP

```
Registro de evolucao da simulacao da rede RTSP_basico.

*---------------------------------------------------------------------------
*

Enumeracao de estados: rede RTSP_basico (33 estados acessiveis).

Propriedades verificadas:
*---------------------------------------------------------------------------
*
A rede em analise e' binaria.
  Lugares Nulos (M = 0): {}
  Lugares Binarios     : {todos(as)}
  Lugares k-Limitados  : {}
  Lugares Nao Limitados: {}

A rede em analise nao e' estritamente conservativa.

A rede em analise e' viva.
  Tr. vivas         : {todos(as)}
  Tr. quase-vivas   : {todos(as)}
  Tr. nao disparadas: {}

A rede e' reiniciavel.

Nao foram detectados "live-locks" na rede.

Nao foram detectados "dead-locks" na rede.
*---------------------------------------------------------------------------
*

Enumeracao de estados: rede RTSP_basico.

Estados acessiveis pela rede:
*---------------------------------------------------------------------------
*
M0   :{Repouso_cliente, Repouso_servidor}
M1   :{Repouso_servidor, Espera_repouso, Setup}
M2   :{Espera_repouso, Pronto_servidor, Setup_ack}
M3   :{Pronto_cliente, Pronto_servidor}
M4   :{Espera_pronto, Pronto_servidor, Setup}
M5   :{Espera_pronto, Pronto_servidor, Setup_ack}
M6   :{Espera_pronto, Pronto_servidor, Setup_nack}
M7   :{Espera_pronto, Pronto_servidor, Play}
M8   :{Espera_pronto, Ativo_servidor, Play_ack}
M9   :{Ativo_cliente, Ativo_servidor}
M10  :{Espera_ativo, Ativo_servidor, Setup}
M11  :{Espera_ativo, Ativo_servidor, Setup_ack}
M12  :{Espera_ativo, Ativo_servidor, Setup_nack}
M13  :{Espera_ativo, Ativo_servidor, Play}
```

```
M14  :{Espera_ativo, Ativo_servidor, Play_ack}
M15  :{Espera_ativo, Ativo_servidor, Play_nack}
M16  :{Espera_ativo, Ativo_servidor, Tear}
M17  :{Repouso_servidor, Espera_ativo, Tear_ack}
M18  :{Espera_ativo, Ativo_servidor, Tear_nack}
M19  :{Espera_ativo, Ativo_servidor, Pause}
M20  :{Espera_ativo, Pronto_servidor, Pause_ack}
M21  :{Espera_ativo, Ativo_servidor, Pause_nack}
M22  :{Espera_pronto, Pronto_servidor, Play_nack}
M23  :{Espera_pronto, Pronto_servidor, Tear}
M24  :{Repouso_servidor, Espera_pronto, Tear_ack}
M25  :{Espera_pronto, Pronto_servidor, Tear_nack}
M26  :{Repouso_servidor, Espera_repouso, Setup_nack}
M27  :{Repouso_servidor, Espera_repouso, Describe}
M28  :{Repouso_servidor, Espera_repouso, Describe_ack}
M29  :{Repouso_servidor, Espera_repouso, Describe_nack}
M30  :{Repouso_servidor, Espera_repouso, Tear}
M31  :{Repouso_servidor, Espera_repouso, Tear_ack}
M32  :{Repouso_servidor, Espera_repouso, Tear_nack}
*--------------------------------------------------------------------------
*

Enumeracao de estados: rede RTSP_basico.

Grafo de acessibilidade da rede:
*--------------------------------------------------------------------------
*
M0   :(Manda_setup_repouso: M1) (Manda_desc_repouso: M27)
      (Manda_tear_repouso: M30)
M1   :(Envia_setup_ack_rep: M2) (Envia_setup_nack_rep: M26)
M2   :(Setup_ack_repouso: M3)
M3   :(Manda_setup_pronto: M4) (Manda_play_pronto: M7)
      (Manda_tear_pronto: M23)
M4   :(Envia_setup_ack_p: M5) (Envia_setup_nack_p: M6)
M5   :(Setup_ack_pronto: M3)
M6   :(Setup_nack_pronto: M3)
M7   :(Envia_play_ack_p: M8) (Envia_play_nack_p: M22)
M8   :(Play_ack_pronto: M9)
M9   :(Manda_setup_ativo: M10) (Manda_play_ativo: M13)
      (Manda_tear_ativo: M16) (Manda_pause_ativo: M19) (Timeout: M3)
M10  :(Envia_setup_ack_a: M11) (Envia_setup_nack_a: M12)
M11  :(Setup_ack_ativo: M9)
M12  :(Setup_nack_ativo: M9)
M13  :(Envia_play_ack_a: M14) (Envia_play_nack_a: M15)
M14  :(Play_ack_ativo: M9)
M15  :(Play_nack_ativo: M9)
M16  :(Envia_tear_ack_a: M17) (Envia_tear_nack_a: M18)
M17  :(Tear_ack_ativo: M0)
M18  :(Tear_nack_ativo: M9)
M19  :(Envia_pause_ack_a: M20) (Envia_pause_nack_a: M21)
M20  :(Pause_ack_ativo: M3)
M21  :(Pause_nack_ativo: M9)
M22  :(Play_nack_pronto: M3)
M23  :(Envia_tear_ack_p: M24) (Envia_tear_nack_p: M25)
M24  :(Tear_ack_pronto: M0)
M25  :(Tear_nack_pronto: M3)
M26  :(Setup_nack_repouso: M0)
M27  :(Envia_desc_ack_rep: M28) (Envia_desc_nack_rep: M29)
M28  :(Desc_ack_repouso: M0)
M29  :(Desc_nack_repouso: M0)
M30  :(Envia_tear_ack_rep: M31) (Envia_tear_nack_rep: M32)
```

```
M31  :(Tear_ack_repouso: M0)
M32  :(Tear_nack_repouso: M0)
*---------------------------------------------------------------------------
*

Analise de Invariantes da Rede RTSP_basico.
*---------------------------------------------------------------------------
*
Inibicoes usadas para esta analise:

  Lugares Obrigatorios: {}
  Lugares Proibidos   : {}
*---------------------------------------------------------------------------
*
Base invariante linear de lugar encontrada:

BI1  :{Repouso_cliente, Espera_ativo, Espera_pronto, Espera_repouso,
       Pronto_cliente, Ativo_cliente}
BI2  :{Repouso_servidor, Pronto_servidor, Ativo_servidor}
BI3  :{Espera_ativo, Espera_pronto, Espera_repouso, -Setup, -Setup_ack,
       -Setup_nack, -Describe, -Describe_ack, -Describe_nack, -Tear,
       -Tear_ack, -Tear_nack, -Play, -Play_ack, -Play_nack, -Pause,
       -Pause_ack, -Pause_nack}
*---------------------------------------------------------------------------
*
Invariantes lineares de lugar encontrados:

IL1  :{Repouso_cliente, Pronto_cliente, Ativo_cliente, Setup, Setup_ack,
       Setup_nack, Describe, Describe_ack, Describe_nack, Tear, Tear_ack,
       Tear_nack, Play, Play_ack, Play_nack, Pause, Pause_ack, Pause_nack}
IL2  :{Repouso_servidor, Pronto_servidor, Ativo_servidor}
IL3  :{Repouso_cliente, Espera_ativo, Espera_pronto, Espera_repouso,
       Pronto_cliente, Ativo_cliente}
*---------------------------------------------------------------------------
*
Lugares que participaram de todos  os invariantes:
  P = {}

Lugares que nao participaram de nenhum invariante:
  P = {}
*---------------------------------------------------------------------------
*
Teste de existencia de sub-redes (maquina de estados):

  Todos os invariantes encontrados sao maquina de estados.
*---------------------------------------------------------------------------
*

Registro de evolucao da simulacao da rede RTSP_basico.

*---------------------------------------------------------------------------
*
```

Apêndice III - Especificação do protocolo RTSP em Estelle

```
specification rtsp_basico systemactivity;
  default individual queue;
  timescale seconds;

  {$ external NETUSERTMN}

  const
    max_string=90;

  type
    frase = array [ 0..max_string ] of char;
    metodo_PDU_tipo = (DESC,SETUP,PLAY,PAUSE,TEAR);
    PDU_tipo = record
                          metodo: metodo_PDU_tipo;
                          status: integer;
                          corpo: frase;
                       end;


  procedure WRITELN; primitive;
  procedure WRITEINT (i: integer); primitive;
  procedure READINT (var i: integer); primitive;
  procedure READSTRING (var s: frase); primitive;
  procedure WRITESTRING (s: frase); primitive;
  procedure CLRSCR; primitive;

  channel SYS_interface(user,provider);
    by user :
      RT_DESCRIBE;
      RT_SETUP;
      RT_PLAY;
      RT_PAUSE;
      RT_CLOSE;

  channel RTSP_interface(user,provider);
    by user :
      DescReq;
      OpenReq(descricao:frase);
      PlayReq;
      PauseReq;
      CloseReq;
    by provider :
      DescResp(status:integer;descricao:frase);
      OpenResp(status:integer);
      PlayResp(status:integer);
      PauseResp(status:integer);
      CloseResp(status:integer);

  channel TRANSPORTE_interface(user,provider);
    by user :
      DataReq(PDU:PDU_tipo);
    by provider :
      DataInd(PDU:PDU_tipo);

  module USER_TYPE activity;
    ip
```

```
    USYS : SYS_interface(provider);
    UCEP : RTSP_interface(user);
end;

module RTSP_TYPE activity;
  ip
    UCEP : RTSP_interface(provider);
    TCEP : TRANSPORTE_interface(user);
end;

module TRANSPORTE_TYPE activity;
  ip
    TCEP_0 : TRANSPORTE_interface(provider);
    TCEP_1 : TRANSPORTE_interface(provider);
end;

body USER_BODY for USER_TYPE;
  ip
    USER : SYS_interface(provider);
    SYS : SYS_interface(user);

  var
    play, setup, erro : BOOLEAN;
    descricao_aux: frase;

  state
    Esp_pedido,Esp_resposta;

  initialize
    to Esp_pedido
    name init_body :
      begin
        setup:=FALSE;
        play:=FALSE;
        erro:=FALSE;
        connect SYS to USER;
      end;

  trans
      from Esp_pedido
        when USER.RT_CLOSE
            to Esp_resposta
              name UsermandaTEAR :
              begin
                WRITELN;
                WRITESTRING(' Encerrando a sessao ...');
                output UCEP.CloseReq;
              end;

  trans
      from Esp_pedido
        when USER.RT_SETUP
            to Esp_resposta
              name UsermandaSETUP :
              begin
                WRITELN;
                WRITESTRING(' Entre com a descricao da apresentacao :');
                READSTRING(descricao_aux);
                output UCEP.OpenReq(descricao_aux);
              end;
```

```
    trans
        from Esp_pedido
          when USER.RT_DESCRIBE
              to Esp_resposta
                name UsermandaDESC :
                begin
                  if (NOT setup)
                    then
                      begin
                        WRITELN;
                        WRITESTRING(' Buscando Descricao da
Apresentacao ...');
                        output UCEP.DescReq;
                      end
                    else
                      begin
                        WRITELN;
                        WRITESTRING(' Erro! Sessao ja estabelecida!');
                        erro:=TRUE;
                      end;

                end;

    trans
        from Esp_pedido
          when USER.RT_PLAY
              to Esp_resposta
                name UsermandaPLAY :
                begin
                  if (setup)
                    then
                      begin
                         WRITELN;
                         WRITESTRING(' Iniciando a sessao ...');
                        output UCEP.PlayReq;
                      end
                    else
                      begin
                         WRITELN;
                         WRITESTRING(' Erro! Sessao nao estabelecida!');
                        erro:=TRUE;
                      end;
                end;

    trans
        from Esp_pedido
          when USER.RT_PAUSE
              to Esp_resposta
                name UsermandaPAUSE :
                begin
                  if (play)
                    then
                      begin
                         WRITELN;
                         WRITESTRING(' Congelando a sessao ...');
                         output UCEP.PauseReq;
                      end
                    else
                      begin
                        WRITELN;
                        WRITESTRING(' Erro! Sessao nao iniciada!');
```

```
                    erro:=TRUE;
                  end;
            end;

trans
    from Esp_resposta
      when UCEP.CloseResp
          to Esp_pedido
            name ServrespTEAR :
            begin
              if (status=2)
                then
                  begin
                    WRITELN;
                    WRITESTRING(' Sessao encerrada!');
                    setup:=FALSE;
                    play:=FALSE;
                  end
                else if (status=3)
                  then
                    begin
                      WRITELN;
                      WRITESTRING(' Servidor redirecionado!');
                      setup:=FALSE;
                      play:=FALSE;
                  end
                  else if (status=4)
                    then
                      begin
                         WRITELN;
                         WRITESTRING(' Sessao nao foi encerrada!');
                      end;
            end;

trans
    from Esp_resposta
      when UCEP.OpenResp
          to Esp_pedido
            name ServrespSETUP :
            begin
              if (status=2)
                then
                   begin
                     WRITELN;
                     WRITESTRING(' Sessao estabelecida!');
                     setup:=TRUE;
                   end
                else if (status=3)
                  then
                    begin
                      WRITELN;
                      WRITESTRING(' Servidor redirecionado!');
                      setup:=FALSE;
                      play:=FALSE;
                    end
                  else if (status=4)
                    then
                      begin
                        WRITELN;
                        WRITESTRING(' Sessao nao estabelecida!');
                      end;
```

```
            end;

trans
    from Esp_resposta
      when UCEP.PlayResp
          to Esp_pedido
            name ServrespPLAY :
            begin
              if (status=2)
                then
                  begin
                     WRITELN;
                     WRITESTRING(' Sessao iniciada!');
                     play:=TRUE;
                  end
                else if (status=3)
                  then
                    begin
                      WRITELN;
                      WRITESTRING(' Servidor redirecionado!');
                      setup:=FALSE;
                      play:=FALSE;
                    end
                  else if (status=4)
                    then
                      begin
                        WRITELN;
                        WRITESTRING(' A sessao nao foi iniciada.');
                      end;
            end;

trans
    from Esp_resposta
      when UCEP.PauseResp
          to Esp_pedido
            name ServrespPAUSE :
            begin
              if (status=2)
                then
                  begin
                    WRITELN;
                    WRITESTRING(' Sessao congelada!');
                    play:=FALSE;
                  end
                else if (status=3)
                  then
                    begin
                      WRITELN;
                      WRITESTRING(' Servidor redirecionado!');
                      setup:=FALSE;
                      play:=FALSE;
                    end
                  else if (status=4)
                    then
                      begin
                        WRITELN;
                        WRITESTRING(' A sessao nao foi congelada.');
                       end;
            end;
```

```
trans
    from Esp_resposta
      when UCEP.DescResp
          to Esp_pedido
            name ServrespDESC :
            begin
              if (status=2)
               then
                 begin
                    WRITELN;
                    WRITESTRING(' Descricao recebida : ');
                    WRITESTRING(descricao);
                 end
               else if (status=3)
                  then
                     begin
                       WRITELN;
                       WRITESTRING(' Servidor Redirecionado!');
                       setup:=FALSE;
                       play:=FALSE;
                     end
                  else if (status=4)
                    then
                      begin
                         WRITELN;
                         WRITESTRING(' Descricao nao recebida!');
                      end;
            end;

trans
      when USYS.RT_DESCRIBE
            name DescDoUser :
            begin
              output SYS.RT_DESCRIBE;
            end;

trans
      when USYS.RT_SETUP
            name SetupDoUser :
            begin
              output SYS.RT_SETUP;
            end;

trans
      when USYS.RT_CLOSE
            name TearDoUser :
            begin
              output SYS.RT_CLOSE;
            end;

trans
      when USYS.RT_PLAY
            name PlayDoUser :
            begin
              output SYS.RT_PLAY;
            end;

trans
      when USYS.RT_PAUSE
            name PauseDoUser :
```

```
              begin
                output SYS.RT_PAUSE;
              end;

  trans
      from Esp_resposta
          provided erro
            to Esp_pedido
              name PDUnaoesp :
              begin
                erro:=FALSE;
              end;

end;

body RTSP_BODY for RTSP_TYPE;
  const
    exemplo='Star Wars II';

  var
    PDU_aux : PDU_tipo;
    PDU_serv : PDU_tipo;
    status_aux : integer;
    descricao_aux : frase;

  procedure BuildPLAY(var PDU: PDU_tipo);
    begin
      PDU.metodo:=PLAY;
    end;
  procedure RespPLAY(var PDU: PDU_tipo; status:integer);
    begin
      PDU.metodo:=PLAY;
      PDU.status:=status;
    end;
  procedure BuildDESC(var PDU: PDU_tipo);
    begin
      PDU.metodo:=DESC;
    end;
  procedure RespDESC(var PDU: PDU_tipo; status:integer);
    begin
      PDU.metodo:=DESC;
      PDU.status:=status;
    end;
  procedure BuildSETUP(var PDU: PDU_tipo);
    begin
      PDU.metodo:=SETUP;
    end;
  procedure RespSETUP(var PDU: PDU_tipo; status:integer);
    begin
      PDU.metodo:=SETUP;
      PDU.status:=status;
    end;
  procedure BuildPAUSE(var PDU: PDU_tipo);
    begin
      PDU.metodo:=PAUSE;
    end;
  procedure RespPAUSE(var PDU: PDU_tipo; status:integer);
    begin
      PDU.metodo:=PAUSE;
      PDU.status:=status;
    end;
```

```
procedure BuildTEAR(var PDU: PDU_tipo);
  begin
    PDU.metodo:=TEAR;
  end;
procedure RespTEAR(var PDU: PDU_tipo; status:integer);
  begin
    PDU.metodo:=TEAR;
    PDU.status:=status;
  end;
function Metodo(PDU: PDU_tipo): metodo_PDU_tipo;
  begin
    Metodo:= PDU.metodo;
  end;
function Status(PDU: PDU_tipo): integer;
  begin
    Status:= PDU.status;
  end;

state
  REPOUSO,ESPERA_REPOUSO,PRONTO,ESPERA_PRONTO,ATIVO,ESPERA_ATIVO;

initialize
  to REPOUSO
  name init_body :
    begin
    end;

trans
    from REPOUSO
      when UCEP.DescReq
          to ESPERA_REPOUSO
            name CliRmanDESC :
            begin
              BuildDESC(PDU_aux);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from REPOUSO
      when UCEP.OpenReq
          to ESPERA_REPOUSO
            name CliRmanSETUP :
            begin
              BuildSETUP(PDU_aux);
              PDU_aux.corpo:=descricao;
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from REPOUSO
      when UCEP.CloseReq
          to ESPERA_REPOUSO
            name CliRmanTEAR :
            begin
              BuildTEAR(PDU_aux);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from ESPERA_REPOUSO
      when TCEP.DataInd
```

```
            provided Metodo(PDU)=DESC
              to REPOUSO
                name CliRrecDESC :
                begin
                  status_aux:=Status(PDU);
                  descricao_aux:=PDU.corpo;
                  output UCEP.DescResp(status_aux,descricao_aux);
                end;

    trans
        from ESPERA_REPOUSO
          when TCEP.DataInd
            provided Metodo(PDU)=TEAR
              to REPOUSO
                name CliRrecTEAR :
                begin
                  status_aux:=Status(PDU);
                  output UCEP.CloseResp(status_aux);
                end;

    trans
        from ESPERA_REPOUSO
          when TCEP.DataInd
            provided ( Metodo(PDU)=SETUP ) and ( (Status(PDU)=4) or
(Status(PDU)=3) )
              to REPOUSO
                name CliRrecSETUPN :
                begin
                  status_aux:=Status(PDU);
                  output UCEP.OpenResp(status_aux);
                end;

    trans
        from ESPERA_REPOUSO
          when TCEP.DataInd
            provided (Metodo(PDU)=SETUP) and (Status(PDU)=2)
              to PRONTO
                name CliRrecSETUP :
                begin
                  output UCEP.OpenResp(2);
                end;

    trans
        from REPOUSO
          when TCEP.DataInd
              to ESPERA_REPOUSO
                name ServRrecPDU :
                begin
                  PDU_serv:= PDU;
                  WRITELN;
                  WRITESTRING(' Entre com o status da resposta (2/3/4) :');
                  READINT(status_aux);
                end;

    trans
        from ESPERA_REPOUSO
            provided Metodo(PDU_serv)=DESC
              to REPOUSO
                name ServRenvDESC :
                begin
                  RespDESC(PDU_aux,status_aux);
```

```
                PDU_aux.corpo:=exemplo;
                output TCEP.DataReq(PDU_aux);
              end;

    trans
        from ESPERA_REPOUSO
            provided (Metodo(PDU_serv)=SETUP) and (status_aux=2)
              to PRONTO
                name ServRenvSETUP :
                begin
                  RespSETUP(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_REPOUSO
            provided (Metodo(PDU_serv)=SETUP) and ( (status_aux=3) or
(status_aux=4) )
              to REPOUSO
                name ServRenvSETUPN :
                begin
                  RespSETUP(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_REPOUSO
            provided (Metodo(PDU_serv)=TEAR)
              to REPOUSO
                name ServRenvTEAR :
                begin
                  RespTEAR(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from PRONTO
          when UCEP.OpenReq
              to ESPERA_PRONTO
                name CliPenvSETUP :
                begin
                  BuildSETUP(PDU_aux);
                  PDU_aux.corpo:=descricao;
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from PRONTO
          when UCEP.CloseReq
              to ESPERA_PRONTO
                name CliPenvTEAR :
                begin
                  BuildTEAR(PDU_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=TEAR) and (Status(PDU)=4)
```

```
              to PRONTO
                name CliPrecTEARN :
                begin
                  output UCEP.CloseResp(4);
                end;

    trans
        from PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=TEAR) and ((Status(PDU)=2) or
(Status(PDU)=3))

              to REPOUSO
                name CliPrecTEAR :
                begin
                  status_aux:=Status(PDU);
                  output UCEP.CloseResp(status_aux);
                end;

    trans
        from ESPERA_PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=SETUP) and ((Status(PDU)=2) or
(Status(PDU)=4))
              to PRONTO
                name CliPrecSETUP :
                begin
                  status_aux:=Status(PDU);
                  output UCEP.OpenResp(status_aux);
                end;

    trans
        from PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=SETUP) and (Status(PDU)=3)

              to REPOUSO
                name CliPrecSETUPRED :
                begin
                  output UCEP.OpenResp(3);
                end;

    trans
        from ESPERA_PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=PLAY) and (Status(PDU)=2)
              to ATIVO
                name CliPrecPLAY :
                begin
                  output UCEP.PlayResp(2);
                end;

    trans
        from ESPERA_PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=PLAY) and (Status(PDU)=4)
              to PRONTO
                name CliPrecPLAYN :
                begin
                  output UCEP.PlayResp(4);
                end;
```

```
    trans
        from PRONTO
          when TCEP.DataInd
            provided (Metodo(PDU)=PLAY) and (Status(PDU)=3)
              to REPOUSO
                name CliPrecPLAYRED :
                begin
                  output UCEP.PlayResp(3);
                end;

    trans
        from PRONTO
          when TCEP.DataInd
              to ESPERA_PRONTO
                name ServPrecPDU :
                begin
                  PDU_serv:= PDU;
                  WRITELN;
                  WRITESTRING(' Entre com o status da resposta (2/3/4) :');
                  READINT(status_aux);
                end;

    trans
        from ESPERA_PRONTO

            provided (Metodo(PDU_serv)=SETUP) and ( (status_aux=2) or
(status_aux=4) )
              to PRONTO
                name ServPenvSETUP :
                begin
                  RespSETUP(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from PRONTO
            provided (Metodo(PDU_serv)=TEAR) and ( (status_aux=2) or
(status_aux=3) )
              to REPOUSO
                name ServPenvTEAR :
                begin
                  RespTEAR(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_PRONTO
            provided (Metodo(PDU_serv)=TEAR) and ( status_aux=4 )

              to PRONTO
                name ServPenvTEARN :
                begin
                  RespTEAR(PDU_aux,4);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_PRONTO
            provided (Metodo(PDU_serv)=PLAY) and ( status_aux=2 )
              to ATIVO
```

```
            name ServPenvPLAY :
            begin
              RespPLAY(PDU_aux,2);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from PRONTO
        provided (Metodo(PDU_serv)=PLAY) and ( status_aux=3 )
          to REPOUSO
            name ServPenvPLAYR :
            begin
              RespPLAY(PDU_aux,3);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from ESPERA_PRONTO
        provided (Metodo(PDU_serv)=PLAY) and ( status_aux=4 )
          to PRONTO
            name ServPenvPLAYN :
            begin
              RespPLAY(PDU_aux,4);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from PRONTO
      when UCEP.PlayReq
          to ESPERA_PRONTO
            name CliPenvPLAY :
            begin
              BuildPLAY(PDU_aux);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from ATIVO
      when UCEP.OpenReq
          to ESPERA_ATIVO
            name CliAenvSETUP :
            begin
              BuildSETUP(PDU_aux);
              PDU_aux.corpo:=descricao;
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from ATIVO
      when UCEP.CloseReq
          to ESPERA_ATIVO
            name CliAenvTEAR :
            begin
              BuildTEAR(PDU_aux);
              output TCEP.DataReq(PDU_aux);
            end;

trans
    from ATIVO
      when UCEP.PlayReq
          to ESPERA_ATIVO
```

```
              name CliAenvPLAY :
              begin
                BuildPLAY(PDU_aux);
                output TCEP.DataReq(PDU_aux);
              end;

    trans
        from ATIVO
          when UCEP.PauseReq
              to ESPERA_ATIVO
              name CliAenvPAUSE :
              begin
                BuildPAUSE(PDU_aux);
                output TCEP.DataReq(PDU_aux);
              end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=TEAR) and (Status(PDU)=4)
              to ATIVO
              name CliArecTEARN :
              begin
                output UCEP.CloseResp(4);
              end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=TEAR) and ((Status(PDU)=2) or
(Status(PDU)=3))
              to REPOUSO
              name CliArecTEAR :
              begin
                status_aux:=Status(PDU);
                output UCEP.CloseResp(status_aux);
              end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=SETUP) and (Status(PDU)=2)
              to ATIVO
              name CliArecSETUP :
              begin
                output UCEP.OpenResp(2);
              end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=SETUP) and (Status(PDU)=3)
              to REPOUSO
              name CliArecSETUPR :
              begin
                 output UCEP.OpenResp(3);
              end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
```

```
            provided (Metodo(PDU)=SETUP) and (Status(PDU)=4)
              to ATIVO
                name CliArecSETUPN :
                begin
                  output UCEP.OpenResp(4);
                end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=PAUSE) and (Status(PDU)=2)
              to PRONTO
                name CliArecPAUSE :
                begin
                  output UCEP.PauseResp(2);
                end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=PAUSE) and (Status(PDU)=3)
              to REPOUSO
                name CliArecPAUSER :
                begin
                  output UCEP.PauseResp(3);
                end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=PAUSE) and (Status(PDU)=4)
              to ATIVO
                name CliArecPAUSEN :
                begin
                  output UCEP.PauseResp(4);
                end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=PLAY) and (Status(PDU)=3)
              to REPOUSO
                name CliArecPLAYR :
                begin
                  output UCEP.PlayResp(3);
                end;

    trans
        from ESPERA_ATIVO
          when TCEP.DataInd
            provided (Metodo(PDU)=PLAY) and ((Status(PDU)=2) or
(Status(PDU)=4))
              to ATIVO
                name CliArecPLAY :
                begin
                  status_aux:=Status(PDU);
                  output UCEP.PlayResp(status_aux);
                end;

    trans
        from ATIVO
```

```
          when TCEP.DataInd
              to ESPERA_ATIVO
                name ServArecPDU :
                begin
                  PDU_serv:= PDU;
                  WRITELN;
                  WRITESTRING(' Entre com o status da resposta (2/3/4) :');
                  READINT(status_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=PLAY) and ( ( status_aux=2 ) or
( status_aux=4) )
              to ATIVO
                name ServAenvPLAY :
                begin
                  RespPLAY(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=PLAY) and ( status_aux=3 )
              to REPOUSO
                name ServAenvPLAYR :
                begin
                  RespPLAY(PDU_aux,3);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=SETUP) and ( status_aux=2 )
              to ATIVO
                name ServAenvSETUP :
                begin
                  RespSETUP(PDU_aux,2);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=SETUP) and ( status_aux=3 )
              to REPOUSO
                name ServAenvSETUPR :
                begin
                  RespSETUP(PDU_aux,3);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=SETUP) and ( status_aux=4 )
              to ATIVO
                name ServAenvSETUPN :
                begin
                  RespSETUP(PDU_aux,4);
                  output TCEP.DataReq(PDU_aux);
                end;
```

```
    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=TEAR) and ( (status_aux=2) or
(status_aux=3) )
              to REPOUSO
                name ServAenvTEAR :
                begin
                  RespTEAR(PDU_aux,status_aux);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=TEAR) and ( status_aux=4 )
              to ATIVO
                name ServAenvTEARN :
                begin
                  RespTEAR(PDU_aux,4);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=PAUSE) and ( status_aux=2 )
              to PRONTO
                name ServAenvPAUSE :
                begin
                  RespPAUSE(PDU_aux,2);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=PAUSE) and ( status_aux=3 )
              to REPOUSO
                name ServAenvPAUSER :
                begin
                  RespPAUSE(PDU_aux,3);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from ESPERA_ATIVO
            provided (Metodo(PDU_serv)=PAUSE) and ( status_aux=4 )
              to ATIVO
                name ServAenvPAUSEN :
                begin
                  RespPAUSE(PDU_aux,4);
                  output TCEP.DataReq(PDU_aux);
                end;

    trans
        from PRONTO
            provided (Metodo(PDU_serv)=SETUP) and (status_aux=3)
              to REPOUSO
                name ServPenvSETUPR :
                begin
                  RespSETUP(PDU_aux,3);
                  output TCEP.DataReq(PDU_aux);
                end;
```

```
  end;

  body TRANSPORTE_BODY for TRANSPORTE_TYPE;
    initialize
      name init_body :
        begin
        end;

    trans
          when TCEP_0.DataReq
                name para_server :
                begin
                  output TCEP_1.DataInd(PDU);
                end;

    trans
          when TCEP_0.DataReq
                name perdido_server :
                begin
                end;

    trans
          when TCEP_1.DataReq
                name para_client :
                begin
                  output TCEP_0.DataInd(PDU);
                end;

    trans
          when TCEP_1.DataReq
                name perdido_client :
                begin
                end;

  end;

  modvar
    USER_A : USER_TYPE;
    RTSP_CLIENTE : RTSP_TYPE;
    RTSP_SERVIDOR : RTSP_TYPE;
    MEIO_DE_TRANSPORTE : TRANSPORTE_TYPE;

  initialize
    name init_modules :
      begin
        init USER_A with USER_BODY;
        init RTSP_CLIENTE with RTSP_BODY;
        init RTSP_SERVIDOR with RTSP_BODY;
        init MEIO_DE_TRANSPORTE with TRANSPORTE_BODY;
        connect USER_A.UCEP to RTSP_CLIENTE.UCEP;
        connect RTSP_CLIENTE.TCEP to MEIO_DE_TRANSPORTE.TCEP_0;
        connect MEIO_DE_TRANSPORTE.TCEP_1 to RTSP_SERVIDOR.TCEP;
      end;

end.
```