6
Trabalhos Relacionados

Neste capítulo reunimos um conjunto de trabalhos coletados durante o desenvolvimento da tese que tiveram relação direta ou indireta com esta. Dividimos este conjunto em: propostas similares, as quais são ambientes de validação baseadas em alguma arquitetura em camadas; linguagens de especificação, de forma a podermos compará-las com LEP; ferramentas utilizadas, e; trabalhos afins, que de alguma maneira se relacionam com a tese.

6.1
Propostas Similares

O ambiente de verificação IF (BGM02) é uma plataforma de validação para sistemas temporizados assíncronos. Este ambiente tem como elemento central sua linguagem de representação intermediária baseada numa extensão de autômatos temporizados. O ambiente contém ferramentas específicas para a representação intermediária, como compiladores, analisadores estáticos, verificadores de modelos, assim como *front-ends* para várias linguagens de especificação e ferramentas de validação. Excluindo as construções temporais, as quais ainda não existem em LEP, uma diferença principal é o foco das arquiteturas. Enquanto IF tem a representação intermediária como a "espinha dorsal"do ambiente, que interliga as diversas linguagens, a nossa arquitetura prioriza a linguagem de entrada. Com isso, nossa representação intermediária fica em segundo plano, só importando para os módulos que fazem a conversão entre LEP e a linguagem dos verificadores.

CADP (JHA+96) é formado por um conjunto de ferramentas para o projeto de protocolos de comunicação e sistemas distribuídos. Dentre as ferramentas, podemos citar os compiladores para vários formalismos, as ferramentas de verificação de equivalência e de modelos. CADP é projetado de maneira modular e, assim como IF, também dá ênfase à sua representação intermediária para integração com outras ferramentas.

VeriTech (KG02) é um projeto que visa integrar diferentes ferramentas de verificação através da tradução de suas linguagens de entrada. A tradução é realizada em três passos: linguagem fonte para o xml da linguagem fonte;

xml da linguagem fonte para o xml da linguagem destino, e; xml da linguagem destino para a linguagem destino. A criação de um formato xml para LEP pode ser uma maneira de integrarmos e extendermos nossa arquitetura.

SAL (BGL+00) é um framework que combina diferentes ferramentas para abstração, análise de programas, prova de teoremas e verificação de modelos através do cálculo de propriedade de sistemas concorrentes. A parte principal do framework é sua linguagem intermediária para especificação de sistemas concorrentes, que é feita de maneira composicional. Comparando com LEP, a linguagem de especificação de SAL está mais próxima das linguagens dos verificadores do que LEP. Assim como no ambiente IF e no projeto VeriTech, a tradução de LEP para a linguagem intermediária de SAL parece promissora, já que poderíamos tirar proveito dos vários métodos de análise implementados no SAL.

In (McG04), o autor usa uma linguagem abstrata chamada TAP (*Timed Abstract Protocol notation*) e seu modelo de computação para expressar suposições assumidas na verificação de protocolos de rede. O modelo de execução pode ser usado na tradução das especificações dos protocolos em TAP para programas em C como uma forma de tornar a especificação executável. Em nossa arquitetura, as linguagens dos verificadores provêm o ambiente de execução. Apesar de ser uma linguagem abstrata, elementos da rede são referenciados explicitamente em TAP através de seus identificadores. Em vários cenários pudemos perceber quão interessante seria o uso de pronomes neste trabalho. Outro diferencial neste trabalho é o fato da tradução de TAP ser em apenas um (1) único passo.

Em (Win01, CW00) é apresentada uma extensão do framework ASM-WB (Abstract State Machines Workbench) (Cas98) para SMV e MDG (Multiway Decision Graphs). ASM-SL é a linguagem de especificação de entrada, baseada em domínios e funções. ASM-IL é a linguagem intermediária que permite o uso de diferentes ferramentas de verificação. Como diferencial para nossa abordagem temos LEP, com construções de alto nível que a tornam mais amigável. Além disto, elaboramos o retorno dos contra-exemplos no nível na linguagem de entrada, o que não está disponível no ASM-WB.

6.2 Linguagens de Especificação

SDL (Z.102) é uma recomendação ITU-T para sistemas de telecomunicação reativos. Sistemas em SDL são constituídos de uma estrutura de agentes comunicantes. Para cada agente pode-se ter um conjunto de instâncias deste agente. Análogo a alguns pronomes de LEP, SDL tem quatro (4) variáveis

anônimas que podem ser aplicadas aos agentes: *self* que referencia a própria instância (pronome *this*); *parent* que referencia a instância criadora (pronome homônimo em LEP); *offspring* que referencia a instância criada mais recentemente (similar ao pronome *children*), e; *sender* que é exatamente o mesmo que o pronome *sender* em LEP. Quando comparamos estes dois (2) conjuntos de abstrações, as construções de LEP são mais abstratas e, por isso, demandam maior esforço computacional, enquanto que as de SDL podem ser obtidas diretamente. Em linhas gerais, SDL é uma linguagem de especificação bem conhecida, estabelecida e bastante completa, às vezes similar a uma linguagem de programação como Pascal. Contém uma representação gráfica intuitiva, que permite uma visão ampla e por níveis de um sistema a ser validado. Enquanto que LEP foi projetada para ser pequena, simples e com construções de domínio-específico de forma a facilitar sua utilização por projetistas de protocolos e algoritmos distribuídos.

UML (*Unified Modeling Language*) é a linguagem gráfica mais popular para a descrição de sistemas. Como UML está voltada para a descrição de sistemas orientados à objetos, a especificação de protocolos ou sistemas baseados em agentes em UML não é imediata. Na literatura encontramos duas (2) abordagens comuns: a adaptação de construções padrões de UML como diagrama de atividades, como apresentado em (Lin01), e a criação de extensões, como no caso de AUML (OPB00, BMO01). Como diferencial entre LEP e estas linguagens de especificação temos a notação textual e gráfica, respectivamente, que são amigáveis ou não, dependendo da familiaridade do projetista, e temos também os pronomes de LEP.

Estelle (BD87) é um padrão ISO para a especificação formal de sistemas distribuídos e concorrentes. Uma especificação em Estelle é composta por um conjunto de módulos que são aninhados hierarquicamente. Numa especificação temos a definição explícita do conjunto de canais de comunicação, dos módulos internos à especificação e do módulo principal, o qual contém as conexões entre os módulos (ligação um a um (1:1) entre os canais) e o sistema de transição que define o comportamento do módulo principal. Diferentemente de LEP, onde temos uma certa transparência na definição das conexões, em Estelle estas especificações são explícitas. Uma característica também encontrada em LEP é a possibilidade de especificação incompleta dos módulos, permitindo a verificação de protocolos parcialmente especificados.

LOTOS (BB87) foi uma linguagem de especificação originalmente projetada para a descrição formal da arquitetura OSI, embora seja comumente aplicada para especificação de sistemas distribuídos e concorrentes em geral. A essência da especificação em LOTOS é a ordenação de eventos no tempo.

A especificação consiste em duas (2) partes: uma comportamental, baseada e com notação de álgebra de processos, e a outra declarativa, baseada em tipos abstratos de dados. Na parte comportamental, por não ter construções comuns das linguagens procedurais usuais, LOTOS tende a ser menos intuitiva do que LEP. Ambas não têm construções específicas para facilitar a validação, o que torna suas especificações mais limpas.

HLPSL (vO05) é uma linguagem expressiva para a especificação de protocolos de segurança da internet, proposta no escopo do projeto AVISPA (ea05). Esta linguagem tem a mesma proposta que LEP, exceto o escopo de protocolos de segurança: prover construções de domínio-específico e abstratas o suficiente de forma a simplificar as especificações ao ponto destas assemelharem-se às descrições de protocolos encontradas em livros didáticos. Para a verificação formal de especificações em HLPSL no projeto AVISPA, estas são traduzidas para o código intermediário de IF (BGM02). Como é comumente encontrado em outras propostas, não é mencionado como se dá a interpretação do resultado da verificação, o qual não estará no nível de abstração de HLPSL.

6.3 Ferramentas Utilizadas

Spin (Hol97) é uma ferramenta de código-aberto usada para verificação formal de sistemas distribuídos. Spin tem sua linguagem de especificação própria (Promela - *Process Meta Language*) e utiliza LTL (*Linear Temporal Logic*) como linguagem de especificação de propriedades. Promela permite a criação dinâmica de processos e a comunicação entre estes, que é feita via canais de mensagens de forma síncrona ou assíncrona. Vários parâmetros podem ser passados ao verificador, os quais configuram, dentre outros, o comportamento do verificador através dos algoritmos utilizados e do modo de busca. Além destes parâmetros, o Spin automaticamente verifica situações indesejáveis como deadlock, erros nas trocas de mensagens e código morto. Todas essas funcionalidades estão disponíveis via comando de linha ou via XSpin, que é uma interface gráfica em Tcl/Tk (acompanha a distribuição completa do Spin) para a manipulação do verificador. Esta interface, dentre outras funcionalidades, permite a configuração de todos parâmetros necessários para a verificação de uma especificação, a simulação de traços de execução de um protocolo, assim como a reprodução dos passos gerados num contra-exemplo.

NuSMV (CCGR00) é o resultado de uma reengenharia, reimplementação e extensão do SMV (McM93). NuSMV permite a especificação de sistemas de transições de estados síncronos e assíncronos para a verificação de expressões

em CTL (*Computer Tree Logic*) e LTL (*Linear Temporal Logic*). Heurísticas são utilizadas de forma a aumentar a eficiência do verificador e parcialmente controlar o problema de explosão de estados. Tanto Spin como NuSMV foram usados como ferramentas de verificação alvo para a arquitetura desenvolvida.

TXL (Cor04) é uma linguagem de programação híbrida (funcional e baseada em regras) para análise e transformação de código. TXL é baseada em unificação, iteração implícita e reconhecimento de padrões. Cada programa em TXL contém uma descrição das estruturas a serem transformadas e um conjunto de regras de transformação. Internamente, a implementação de TXL está baseada em reescrita formal de estruturas no formato de árvores. TXL foi utilizado na implementação dos módulos que fazem a interligação entre as camadas da arquitetura.

MMC (YRS03) (*Mobility Model Checking*) é um verificador de modelos para sistemas móveis especificados no estilo de $\pi-calculus$. Suas propriedades são fornecidas utilizando um subconjunto de $\pi - \mu-calculus$. MMC foi implementado sobre um sistema de programação em lógica e de banco de dados dedutivo, ou seja, é uma nova abordagem para o analisador de sistemas concorrentes CWB (MS). Utilizamos o MMC inicialmente para ganharmos experiência no uso de linguagens no estilo de $\pi-calculus$ e avaliarmos a possibilidade de basearmos LEP em $\pi-calculus$. Deste estudo extraímos a sobrecarga entre identificadores de processos e canais existentes em LEP.

6.4 Trabalhos Afins

(DAC98) é uma coleção de padrões de especificação de propriedades para sistemas concorrentes. Nesta coleção, temos as propriedades definidas nos formalismos LTL, CTL, GIL (*Graphical Interval Logic*), QRE (*Quantified Regular Expressions*) e em consultas INCA. Neste trabalho, a taxonomia hierárquica para estes padrões é definida segundo a figura 6.1.

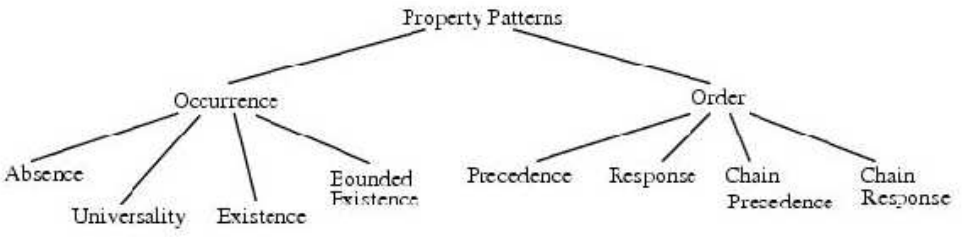

Figura 6.1: Hierarquia para padrões de especificação de propriedades

Na tese, esta coleção de padrões serviu como referência para a especificação das propriedades sobre os modelos.

Em (WPP04) os autores fazem algumas ponderações sobre quais propriedades podem realmente serem verificadas em protocolos ad hoc. Um estudo de caso é apresentado, cuja verificação é feita com os verificadores de modelos Spin (Hol97) e UPPAAL (BLL+95). Relacionado à tese temos algumas decisões tomadas na modelagem em Spin, como a forma de se modelar a movimentação, as quais são automaticamente geradas pela arquitetura. Além disso, na verificação de um protocolo apresentada neste trabalho os autores argumentam que há a necessidade de se definir uma estrutura mais geral para a especificação das topologias. Acreditamos que AGG (*Attribute Graph Grammar*) seja um passo neste sentido.

Em (GV03) é proposto um método para a interpretação de contra-exemplos que se baseia na investigação destes. A idéia básica é identificar pontos em comum nas execuções que geram contra-exemplos, chamadas de execuções *negativas*. À partir destes pontos são identificadas porções da especificação que devem ser cruciais na distinção de execuções que levam à contra-exemplos das execuções que não levam, chamadas de execuções *positivas*. Comparando com nossa abordagem, em (GV03) temos uma análise semântica dos contra-exemplos, baseada em conjuntos de execuções, enquanto que a nossa é essencialmente sintática sobre a especificação do protocolo. Para a arquitetura, os pontos chaves são pré-definidos (pontos de sincronização, onde a variável de controle *codePositionLEP* é atualizada), enquanto que em (GV03) estes pontos de interesse são calculados. Entretanto, ambos tem a mesma meta que é tornar os contra-exemplos mais amigáveis para o projetista inserindo nestes construções existentes na especificação original.

Em (Lau02) foi apresentada a modelagem e verificação formal dos protocolos DSR (JMB01) e GPSAL (CL00) no verificador de modelos Verus (SEM97). Verus, diferentemente de outros verificadores, é voltado para avaliação quantitativa dos modelos permitindo, por exemplos, avaliar o desempenho como tempo de resposta e carga do sistema. Com respeito à arquitetura, de (Lau02) e (WPP04) percebemos que era interessante a definição de Modelos Mínimos para a especificação de protocolos de forma a tornar a definição do tamanho mínimo de uma rede menos empírica.

MobiCS (RE01) é um framework para prototipação e simulação de protocolos para redes móveis estruturadas. A especificação em MobiCS é feita através da extensão de classes básicas em Java, implementação de interfaces e a junção para se construir o simulador. MobiCS tem dois (2) modos de funcionamento: determinístico e estocástico. Para se guiar uma simulação no modo determinístico, é necessário criar um *script* que conterá a criação dos agentes, das mensagens, dos deslocamentos dos agentes e dos pontos de

sincronização da simulação na sequência em que estes ocorrem no *script*. O objetivo é analisar cenários críticos do protocolo que está sendo simulado. Entretanto, dependendo do número de agentes e mensagens, a tarefa de construção do *script* pode ser árdua. No modo estocástico definem-se os padrões de comportamento dinâmico de todos os elementos da rede, como por exemplo a frequência de migrações de um agente, de forma a testar o protocolo num cenário aleatório e mais realístico.

Comparando nossa abordagem com o MobiCS, também temos versões determinísticas e aleatórias das verificações se iniciarmos as mensagens na transição *init* ou, de forma não-determinística, criarmos pelo menos duas (2) transições com pré-condição *true* e com as ações sendo os envios das mensagens iniciais. Dada a exploração exaustiva dos verificadores de modelos, não precisamos definir uma ordem particular no envio das mensagens na transição *init*, uma vez que todas as ordenações serão analisadas. Entretanto, no envio de mensagens não-determinísticas não temos como definir, por exemplo, frequências distintas de migração. No máximo podemos garantir, utilizando o conceito de justiça (*fairness*) dos verificadores de modelos, que sempre que um processo estiver apto para execução, em algum momento ele executará. Ou o contrário, se quisermos permitir que algum processo não execute, ou seja, que algum agente não migre.