UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA

2008.2

Estudo de Viabilidade do Uso da Abdução com Regras de Resolução de Restrições para Sistema Especialista de Diagnóstico de Falha em Redes de Telecomunicação Interna a Empresa do Setor Energético

TRABALHO DE GRADUAÇÃO EM INTELIGÊNCIA ARTIFICIAL

Aluno – Luiz Carlos Assis d´Oleron Barreto (lcadb@cin.ufpe.br)

Orientador – Jacques Robin (jr@cin.ufpe.br)

Novembro de 2008.

“Daria tudo que sei pela metade do que ignoro.”

René Descartes

Agradecimentos

Agradeço à minha mãe Ivanise, por tudo que ela representa para mim;

Agradeço à minha tia Zuleide, por ser uma pessoa fantástica para mim;

Ao Professor Jacques Robin, por sua orientação e inspiradora paixão pela pesquisa e ciência;

À minha esposa Katiane, companheira dos bons e maus momentos;

Ao meu colega de trabalho na Chesf Alexandre Jansen, pelo apoio inquestionável a este trabalho e no dia a dia;

Aos colegas Viviane Souza e Lúcio Jorge, pelos dias que compartilhamos durante o curso;

Ao Professor Jarbas Maciel, por ter me mostrado ainda pequeno, o valor da dedicação e do trabalho.

Resumo

Devido a suas proporções geográficas, a Chesf, Companhia Hidro Elétrica do São Francisco, possui uma infra-estrutura de telecomunicação de grande porte. A fim de manter a continuidade dos serviços de telecomunicação desta infra-estrutura de acordo com os requisitos do setor, é necessário agilidade e efetividade no diagnóstico de problemas. Neste trabalho é verificado a viabilidade de uma abordagem baseada em sistemas especialistas abdutivos que seja capaz de inferir o estado dos equipamentos que suportam os serviços de telecomunicação, identificando os defeitos e suas respectivas causas a partir da lista de alarmes fornecidos. Para isso, é usada a linguagem Constraint Handling Rules - CHR, que permite o uso de abdução com regras de resolução de restrições.

Abstract

Due to its geographic proportions, Chesf, Companhia Hidro Elétrica do São Francisco, holds a big size telecommunication infra-structure. In order to maintain the telecommunication services continuity of such infra-structure according to the requisites of the sector, agility and effectivity on diagnosing problems is necessary. In this work, the viability of an approach based on abduct specialist systems able to infer the state of the equipments which supports the telecommunication services, identifying defects and its respective causes from the provided list of alarms, is verified. For such approach, Constraint Handling Rules – CHR – is used, allowing the use of abduction with restriction resolution rules.

Índice

Índice de figuras

1 Introdução

O presente trabalho propõe a apresentar um estudo de viabilidade do uso da abdução como forma de raciocínio lógico para construção de sistemas especialistas responsáveis por identificar as causas de falhas em redes de telecomunicações industriais.

1.1 Problema prático na CHESF

A Chesf possui uma planta de telecomunicações complexa e relativamente grande o suficiente para tornar o trabalho de indentificação das causas dos seus defeitos difícil o suficiente para ser resolvida através da forma tradicional. Atualmente, a Chesf conta com uma equipe de especialistas disponíveis 24 horas, 7 dias por semana, para atender a esta demanda. A incorporação de novos equipamentos e serviços tem mostrado que esta estratégia é um tanto custosa e pouco escalável, a médio e longo prazo.

1.1.1 A Chesf

A Companhia Hidro Elétrica do São Francisco – Chesf, criada em 03 de outubro de 1945, tem como missão produzir, transmitir e comercializar energia elétrica para a Região Nordeste do Brasil[1]. Além de atender os estados da Bahia, de Sergipe, de Alagoas, de Pernambuco, da Paraíba, do Rio Grande do Norte, do Ceará e do Piauí, com a abertura permitida pelo novo modelo do Setor Elétrico Brasileiro, a Chesf tem contratos de venda de energia em todos os submercados do sistema interligado nacional.

1.1.2 Necessidade de Telecomunicação na Chesf

Devido a suas proporções geográficas, a Chesf possui um infra-estrutura de telecomunicação de grande porte, de modo a atender tipos e quantidades distintas de clientes (funcionários e sistemas informatizados) com diferentes requisistos de qualidade. Por infra-estrutura de telecomunicação se entende as necessidade da empresa com:

- Rede de Dados WAN (Wide Area Network)
- Comutação (telefonia corporativa)
- Teleproteção da rede elétrica
- SAGE – Sistema Aberto de Supervisão e Controle [2]
- Outros sitemas de menor porte

Todas essas necessidades são tratadas pelos especialistas como "Serviços de Telecomunicação". Serviços são necessidades de telecomunicação ponto-a-ponto. Por exemplo, o serviço de "Rede WAN UXG-COS":

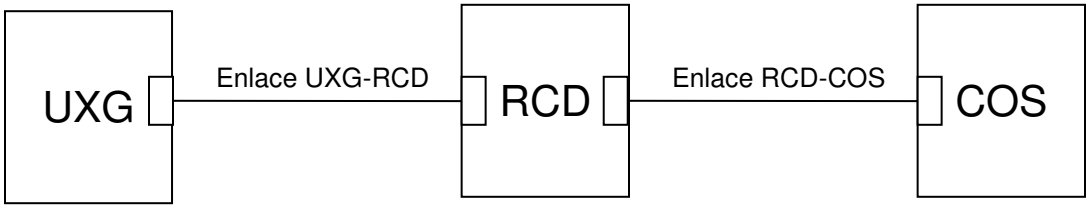

Figura 1: Rede WAN UXG - COS

representa a necessidade de enlace de telecomunicação entre um roteador em UXG (usina Xingó) e outro no COS (Centro de Operações em Recife). Apesar deste serviço trafegar pela instalação de RCD (Recife II) é importante lembrar que todo serviço conecta apenas dois pontos (ponto-a-ponto). Neste caso, a instalação de RCD serve apenas como um nó passante e não consome nem provê o serviço, neste exemplo.

1.1.3 Serviços e Equipamentos de Transmissão

Todo o fluxo dos serviços de telecomunicação na Chesf trafegam através de uma rede chamada de Rede de Transmissão Digital. Esta rede é a responsável por transportar todos os sinais digitais e analógicos entre os pontos da rede Chesf. A rede de transmissão é composta basicamente dos seguintes equipamentos:

- Enlaces de Fibras Óticas
- Enlaces de Rádio Digital
- Enlaces de Cabos Elétricos
- Multiplexadores Digitais (e seus componentes: placas óticas, slots e cross-conexões)

Um serviço de telecomunicação deve trafegar através de vários equipamentos. No nosso exemplo, o serviço “Rede WAN UXG-COS” trafega por multiplexadores nas instalações UXG, RCD e COS, assim como nos respectivos enlaces UXG-RCD e RCD-COS.

Além disso, por um equipamento de transmissão, por exemplo, um multiplexador, podem passar vários serviços.

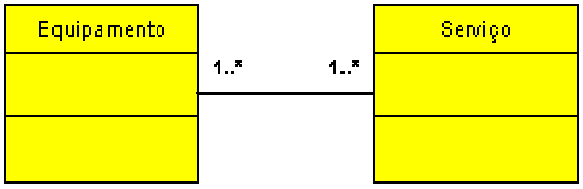

Figura 2: Relacionamento Equipamento – Serviços

Enlaces formam o meio físico por onde a onda portadora dos dados deve trafegar. Nos enlaces de Fibra ótica, a onda é convertida em um sinal de luz e inserida no meio ótico confinado. Na rede da Chesf, enlace óticos permitem o transporte de fluxos de dados da ordem de 2 Mbps, 34 Mbps, 155 Mbps e 622

Mbps. Nos enlaces de rádio digital, o ar é utilzado para propagar a onda eletromagnética portadora dos dados. Na Chesf, o tráfego através de rádio é realizado em uma taxa de 155 Mbps. Enlaces elétricos são utilizados em pequenas distâncias, permitindo taxas que variam entre 2 Mpbs e 155 Mbps.

Multiplexadores são equipamentos responsáveis por receber fluxos de dados através de placas óticas ou elétricas e multiplexar este fluxo em outros fluxos. O transporte dos dados, normalmente é realizado no sentido de transmissão (TX) e recepção (RX) em ambas as placas simultâneamente. Este modelo de transporte é chamado de *full-duplex*.

Para controlar o processo de multiplexação e demultiplexação, o multipexador utiliza um ou mais componentes internos chamados de matrizes de cross-conexão. Nestas matrizes, estão configuradas os ajustes de cross-conexão: as placas de onde os fluxos são obtidos e as placas aonde o fluxo será demultiplexado (ou vice-e-versa).

Em um multiplexador, normalmente várias placas estão dispostas em *slots*. Cada placa possui sua taxa de transmissão e é construída fisicamente para se adaptar ao tipo de enlace físico ao qual está acoplada. Desta forma, tem-se placas óticas (adequadas para uso em enlaces de fibra ótica) e as placas com interface elétrica (adequadas para uso em enlaces elétricos e de rádio digital).

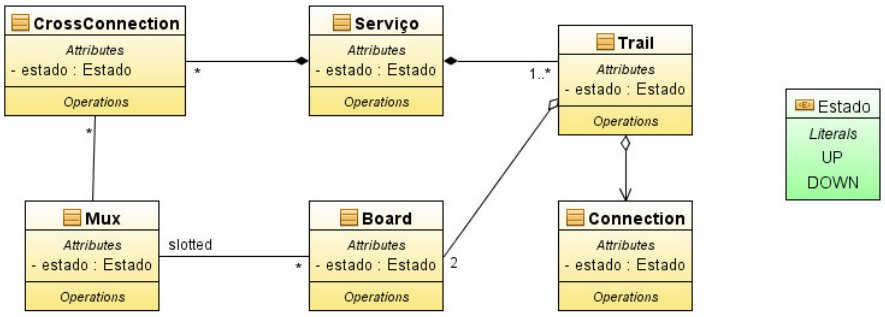

Figura 3: Ontologia do Domínio

1.1.4 Gerências de Telecomunicação

Serviços e equipamentos de telecomunicação devem ser supervisionados diariamente, 24 horas por dia, pelo Centro de Operação e Supervisão de Telecomunicação da Chesf – CSTL. O CSTL executa suas ações de supervisão através de sistemas computacionais chamados de Gerências.

Gerências são sistemas que aquisitam o estado dos serviços de telecomunicação. Através delas é possível supervisionar o fluxo, disponibilidade e configurações dos serviços.

1.1.5 Defeitos, causas, eventos e alarmes

Um defeito em um serviço é caracterizado pela sua execução distinta daquela prevista para ele. Estes defeitos podem ser caracterizados por:

- Perda de performance

- Fornecimento descontínuo
- Diminuição da confiabilidade (perda de redundância)
- Perda completa do serviço

A perda completa do serviço é comumente chamada pelos especialistas como service down e será objeto deste trabalho.

As causas de um defeito normalmente estão associadas a:

- Defeito em um dos equipamentos que provê o serviço
- Defeito em um dos componentes de um dos equipamentos que provê o serviço
- Perda parcial ou total do enlace físico do serviço
- Falha na configuração (intencional ou acidental) do serviço

Um defeito pode ocorrer por uma ou mais causas, aonde estas causas podem ocorrer de forma isolada ou em conjunto. Por outro lado, uma causa pode gerar um ou mais defeitos.

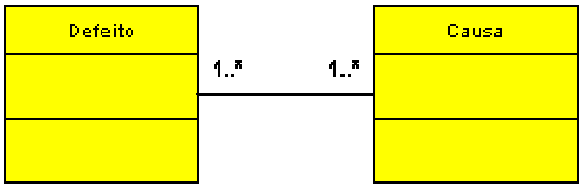

Figura 4: Relacionamento Defeito-Causa

1.1.6 Desafios na supervisão e controle dos ativos de telecomunicação

Como dito anteriormente, as gerências supervisionam os serviços, de modo que nem sempre a identificação da causa da falha é facilmente realizada.

Uma solução para identificação da causa, dado um defeito, seria simular um conjunto de causas possíveis e suas combinações, comparando seus

resultados com o estado dos serviços. Entretanto, esta estratégia ingênua não é razoável, devido a:

- Quantidade de combinações de causas possíveis, gerando um conjunto intratável de possibilidades até mesmo para situações simples;
- Incerteza quanto aos eventos, visto que nem todo defeito pode ser identificado por um ou mais alarmes.

1.1.7 Problema: como diagnosticar as causas a partir do alarmes?

O problema tratado por este trabalho é:

Dados

- a lista de alarmes
- Estado verificado dos serviços
- Estado esperado dos serviços

Inferir o estado dos equipamentos de telecomunicação que suportam este serviço, identificando o(s) defeito(s) e sua(s) respectiva(s) causa(s).

Atualmente, esta atividade é resolvida em tempo real pela intervenção humana de especialistas. Conforme será visto a seguir, propõe-se uma abordagem baseada em sistemas especialistas para resolvê-la automaticamente, sem intervenção humana e em tempo real.

1.2 Abordagem: Sistema Especialista Baseado em Regras

O DENDRAL [3], primeiro sistema especialista desenvolvido, era capaz de interpretar a saída de um espectrômetro de massa e inferir a estrutura molecular da amostra.

A primeira versão do sistema calculava todas as estruturas possíveis para a amostra, comparando o espectro de cada estrutura calculada com o espectro medido. Essa abordagem baseada em força bruta se mostrou ineficaz para resolver moléculas de tamanhos razoáveis. Assim, a equipe do DENDRAL procurou analisar a forma como especialistas realizavam a análise de moléculas e perceberam que eles procuravam por determinados padrões no espectro que sugeriam subestruturas comuns. Estas heurísiticas foram transformadas em regras que transformaram o DENDRAL no "primeiro sistema bem sucedido de conhecimento intensivo" [4].

Desde então, sistemas especialistas foram desenvolvidos nas mais diversas áreas, desde medicina até perícia jurídica, incluindo controle de falhas. Na construção de muitos destes sistemas foi utilizada uma linguagem lógica que se tornou bastante popular, chamada de Prolog.

1.3 Regras causais aplicadas por máquina de inferência abdutiva

Até 1865, o raciocínio lógico estava dividido em duas classes: a de argumentos dedutivos e a de argumentos indutivos. Foi nesta época que Charles Sanders Peirce sugeriu que existiam duas classes distintas de argumentos indutivos. Pierce se referia a essas classes de raciocínio como inferências indutivas e inferências abdutivas [5].

De acordo com [6], dedução, indução e abdução podem ser definidas da seguinte forma:

Dedução: um processo analitico baseado na aplicação de regras gerais para casos particulares (causas), no qual se infere um resultado.

Indução: raciocínio sintético no qual se infere a regra a partir da causa e do resultado.

Abdução: uma outra forma de inferência sintética, na qual se obtém a causa a partir das regras e dos resultados.

Peirce caracterizou a abdução como a adoção plausível de uma hipótese a partir de fatos observados (resultados), respeitando as leis conhecidas (as regras). *"It is however a weak kind of inference, because we cannot say that we believe in the truth of explanation, but only that it may be true"* [7]

Um exemplo [6] de abdução é apresentado a seguir:

Considerando a teoria *T*:

A grama está molhada ← Choveu ontem à noite
A grama está molhada ← O irrigador estava ligado
Os sapatos estão molhados ← A grama está molhada

Caso se observe que os sapatos estão molhados, {Choveu ontem à noite} é uma boa explanação. Entretanto, outras explanações podem ser obtidas, a saber, {O irrigador estava ligado} e {Choveu ontem à noite, O irrigador estava ligado}.

A existência de múltiplas explanações é uma característica comum no raciocínio abdutivo, sendo uma fonte de importantes problemas. Em [8] foi demonstrado que a tarefa de se obter a explanação mais plausível a partir do conjunto de hipóteses que explanam todas as observações tem complexidade NP-difícil na maioria dos casos. A explosão combinacional das hipóteses torna a identificação da melhor explanação um problema geralmente intratável.

1.4 CHR^{v} como máquina de inferência abdutiva

Máquinas de inferência são sistemas capazes de derivar soluções a partir de uma base de conhecimento [9]. Máquinas de inferência abdutivas utilizam o raciocínio lógico abdutivo para solucionar um resultado possível para uma consulta.

Como exemplo[10], segue a base de conhecimento:

Todo albatroz é um pássaro;
Todo pinguim é um pássaro;
Pinguins não voam.

Uma máquina de inferência abdutiva seria capaz de responder a seguinte consulta:

"Existe um pássaro que voa?"

Com uma (única) solução: albatroz.

Uma Máquina de Inferência Abdutiva pode ser escrita utilizando CHR. CHR[11], ou Constraint Handling Rules, é uma linguagem baseada em lógica de primeira ordem utilizada para problemas de resolução de restrições através de re-escrita de restrições. Quando CHR é utilizado para escrever cláusulas abdutivas, ela é designada de CHR^{v}.

2 Diagnóstico de Falha em Redes de Telecomunicação da CHESF

Neste capítulo é descrito o domínio do problema que este trabalho se propõe a tratar.

2.1 Sistema atual de diagnóstico de falha

Situado em Recife, o CSTL opera e supervisiona a rede de telecomunicação da Chesf 24 horas por dia, 365 dias por ano. O trabalho de tempo real é realizado por uma equipe de 14 operadores de telecomunicação, que se revesam em quatro turnos diário de 6 horas. Em cada turno normalmente trabalham dois operadores, todos com escolaridade no mínimo de técnico industrial de nível médio. Além desses operadores, há também três engenheiros que trabalham em horário administrativo suportanto a equipe de operação com apoio técnico e burocrático.

O conjunto de 14 operadores e 3 engenheiros constitui os especilistas de telecomunicação que, dentre outras atividades, tem a resposabilidade de indentificar as causas das ocorrência que contigenciam a rede de telecomunicação.

A identificação das causas se dá por meio indireto. Através das listas de eventos (mudanças de configuração e alarmes de serviços) fornecidas pelas gerências, os especialistas, munidos de diagramas e tabelas auxiliares, inferem as causas de cada ocorrência. Basicamente:

I. Obtém a lista de eventos, através da gerência, identificando a contigência;

II. Obtém dados externos (telefonemas e informações de usuários) osbre a ausência de algum serviço;

III. Obtém a configuração do sistema, através da gerência, identificando possíveis causas triviais;

IV. Obtém a configuração prevista (ou de trabalho) do sistema, através de tabelas e diagramas;

V. Infere a(s) causa(s) da contigência;

VI. Propõem uma solução ou informa órgão de manutenção.

A identificação das causas é normalmente realizada com sucesso, com diferentes tempos de duração (contado a partir do alarme da gerência). De fato, alguns especilistas possuem maior capacidade de identificação das causas do que outros, a depender:

I. Experiência

II. Conhecimento da tecnologia envolvida

III. Autoconfiança

IV. Conhecimento da configuração do sistema

V. Perfil intelectual

Dessa forma, a variação dos tempos de resposta e da efetividade do diagnóstico dependem muito da natureza emocional e psíquica dos envolvidos. Esta característica não determinística na identificação das causa não é interessante do ponto de vista do cliente do recurso de telecomunicação e da gestão do órgão de operação.

2.2 Modelo simplificado da rede

A rede de telecomunicação da Chesf é bastante diversificada, possuindo equipamentos com funcionalidades similares e de fabricantes diversos, equipamentos de naturezas distintas mas do mesmo fabricante ou de fabricantes distintos, serviços de telecomunicação de diferentes níveis de criticidade e clientes de cardinalidade e perfis distintos.

Este trabalho aborda somente a camada mais importante da rede da Chesf, que é a camada de transmissão.

A camada de transmissão da Chesf é responsável por transportar todos os fluxos de dados entre os nós da rede (multiplexadores). Esta camada utiliza duas tecnologias distintas para realizar sua função: SDH, para transmissões acima de 2 mbps (34 mbps, 155 mbps e 622 mbps) e PDH(para taxas de 2 mbps).

2.3 Modelo simplificado de alerta

A maior parte das informações utilizadas pelos especialistas na identificação das causas dos defeitos é obtida através das gerências através da lista de alarmes e eventos. Outra fonte importante de informação é a configuração da rede e dos serviços. Tais informações algumas vezes se encontram dispostas nas gerências de rede, outras vezes exigem contato físico

com o equipamento direto ou mesmo um através de contato telefônico com um técnico em campo.

Defeitos são identificados através de listas de alarmes. Além dos alarmes, mudanças na configuração do serviço (automáticas ou manuais) são sinalizada para o operador da gerência através da lista de eventos.

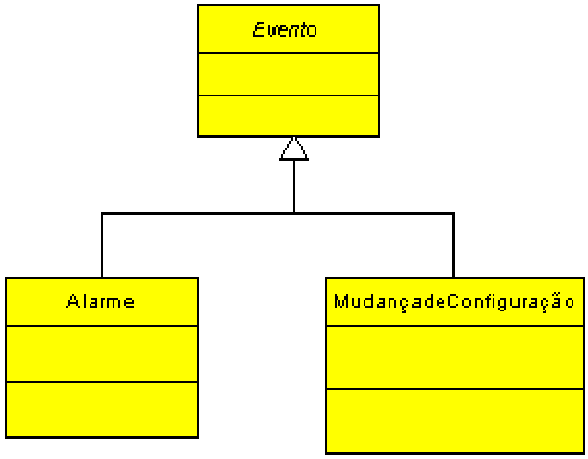

Figura 5: Tipos de Eventos

2.4 Modelo simplificado de diagnóstico

O principal sintoma de falha na rede é a ocorrência de perda de serviço. A perda de serviço normalmente está associada a um ou mais alarmes e eventos. A figura a seguir apresenta a relação de causa e consequência dos defeitos na rede de telecomunicações da Chesf utilizada neste trabalho:

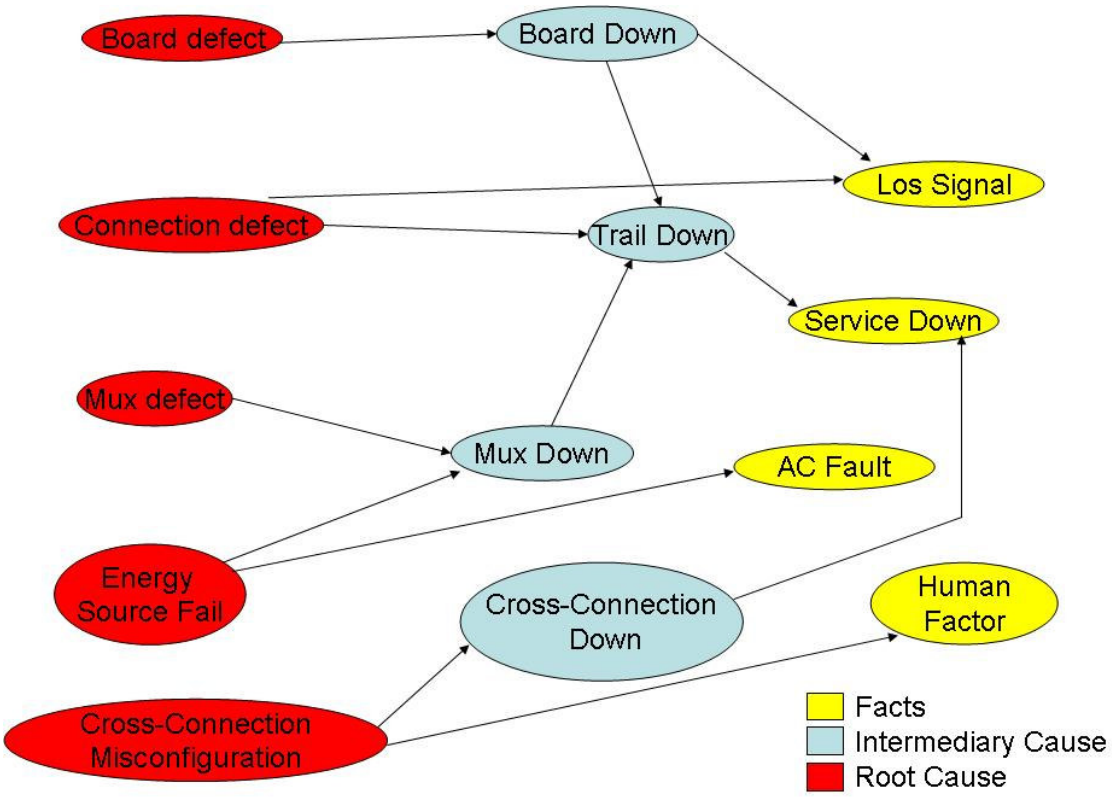

Figura 6: Relação de Causa e conseqüência dos defeitos na rede

Os fatos, em amarelo, são os elementos observáveis. Eles estão disponíveis ao especialista em tempo real, através do sistema supervisório e demais meios de coleta descritos anteriormente. Em azul estão as causas intermediárias, que são explanações parciais do defeito. Finalmente, em vermelho estão as causas raiz, cuja identificação é o objetivo do especialista.

3 Diagnóstico de falha por abdução

Neste capítulo é apresentado o embasamento teórico utilizado no diagnóstico de falha através do raciocínio abdutivo. É apresentada também a linguagem CHR e sua variante CHRv, utilizada na implementação do trabalho.

3.1 Programação em Lógica Abdutiva com Restrições

Dado um conjunto de sentenças T (uma represetação de teoria) e uma sentença G (observação), o raciocínio abdutivo, ou simplesmente, abdução, é caracterizado pelo problema de encontrar um conjunto de sentenças R (explanação abdutiva de G) da seguinte forma:

T U R |= G

T U R é consistente

Esta forma de descrever abdução é independente de qual linguagem T, G e R são formuladas. O sinal de implicação lógica |= pode ser alternativamente substituído pelo operadorde dedução ->.

Abdução permite uma grande expressividade e uma capacidade de representação de conhecimento de alto nível. Sua aplicabilidade tem sido verificada em abordagens de IA para problemas de diagnóstico de faltas, processamento de linguagens naturais, database view update, planejamento e aprendizagem de máquina [6]. Normalmente, as soluções desses problemas com enfoque abdutivo é realizado através da técnica ACLP (Abductive Constraint Logic Programming)[12].

De acordo com [12] e [10] uma teoria ACLP é uma tupla ***{P, A, IC}*** aonde:

P é um programa de lógica de restrições

A é um conjunto de predicados abducíveis, diferentes dos predicados das restrições

IC é um conjunto de fórmulas de primeira ordem, chamado de Integrity Constraints.

Soluções para observações G são chamadas de explanações de G sob a teoria ACLP. Essas soluções tem a forma de um subconjunto dos predicados abducíveis somados a um conjunto de restrições oriundas do programa P.

3.2 Regras Disjuntivas de Resolução de Restrições (CHR^{v})

CHR, ou *Constraint Handling Rules*, foi originalmente definida para ser uma linguagem lógica para descrever *constraint solvers.* Tais programas seriam partes de outros programas escritos em outra linguagem de programação, como Java, Prolog ou Haskell[13].

Atualmente, CHR é uma linguagem de representação de conhecimento de propósito geral, além de ser também uma linguagem de programação Turing-compatível.

Em um programa CHR, a base de fatos pode possuir tanto conhecimento extensional quanto intencional, na forma de conjunções de restrições. CHR possui uma extensão chamada de CHRv, a qual permite o uso de busca por *backtracking*, descrita mais na frente.

3.3 CHR^{v} ilustrado por exemplo

A execução de um programa CHR^{v} se dá através da aplicação iterativa das regras de reescrita sobre um conjunto inicial de restrições chamado de observações (ou *goal)*. Como exemplo temos o seguinte programa CHR^{v}:

```
bird ⇔ albatross ∨ penguin.
penguin ∧ flies ⇒ false.
```

Levando em consideração as seguintes observações:

```
bird ∧ flies
```

Irá se obter a execução do programa através da reescrita das regras. A cada momento uma expressão lógica será processada, a qual é chamada de

ConstraintStore. Neste exemplo inicialmente a única regra aplicável é a primeira delas, logo a *ConstraintStore* inicial será reescrita para:

```
(albatross ∨ penguin) ∧ flies
```

Colocando a expressão na forma normal disjuntiva:

```
(albatross ∧ flies) ∨ (penguin ∧ flies)
```

Aplicando-se a segunda regra na segunda cláusula da disjunção, tem-se:

```
(albatross ∧ flies) ∨ (false)
```

o qual será o resultado final do processamento do programa CHR^{v}.

3.4 Sintaxe

A figura a seguir representa a sintaxe de CHR^{v} através de um metamodelo MOF (Meta-object Facility) [14]:

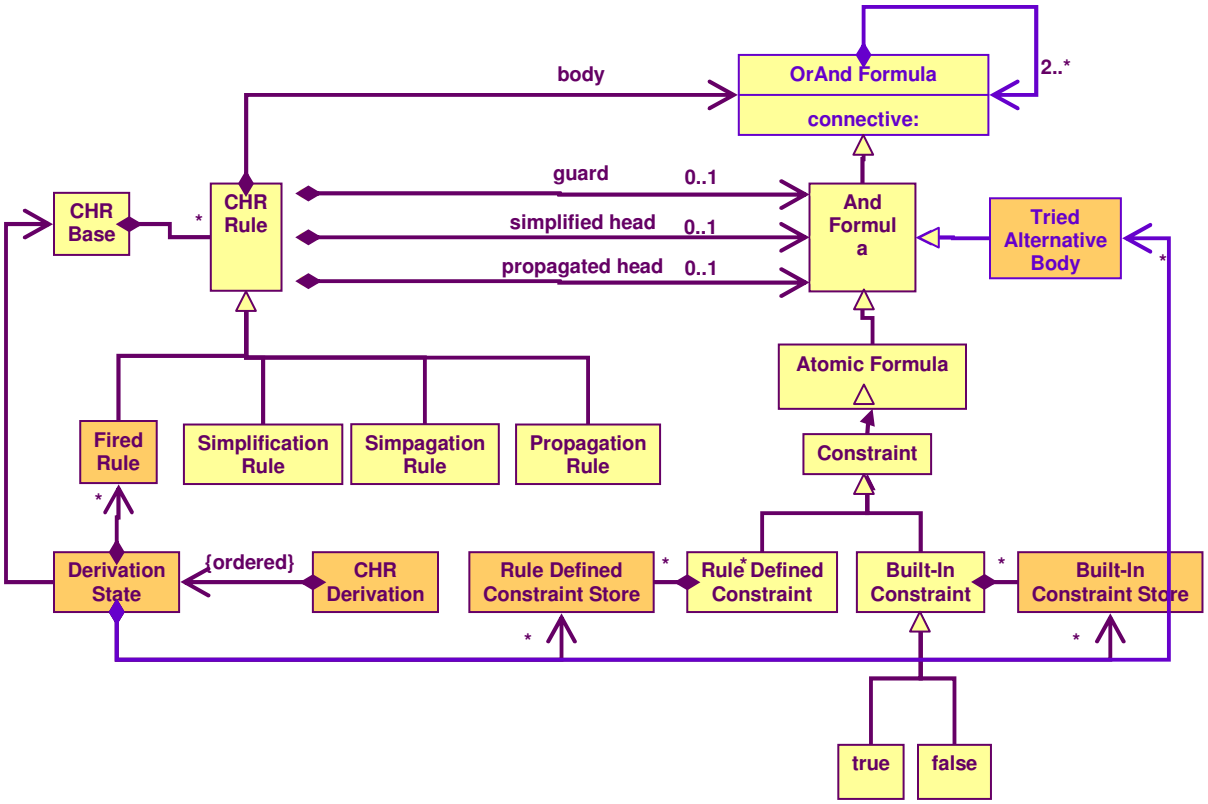

Figura 7: Metamodelo MOF para CHR^v

Ao definirmos um metamodelo para uma linguagem abstraímos as notações específicas adotadas pela linguagem e nos concentramos somente com os conceitos que ela representa e as relações entre eles.

De acordo com o modelo, existem três tipos de regras: as do tipo *SimpagationRule*, que representam as regras do tipo "*simpagation*"; as do tipo "*SimplificationRule*", que representam as regras de simplificação e as do tipo "*PropagationRule*", que representam as regras de propagação.

3.5 Semântica lógica

A semântica de CHR^v pode ser descrita através do comportamento lógico das suas regras. A seguir descreveremos o comportamento dessas regras através de lógica formal de primeira ordem [15]:

Simplification rule: **sh1, ... , shi <=> g1, ..., gj | b11, ..., bkp ; ... ; b11, ..., blq.**

aonde: **{X1, ..., Xn} = vars(sh1 ∪ ... ∪ shi ∪ g1 ∪ ... ∪ gj)**

e **{Y1, ... , Ym} = vars(b1 ∪ ... ∪ bk) \ {X1, ..., Xn}**

∀X1, ..., Xn g1 ∧ ... ∧ gj ⇒ (sh1 ∧ ... ∧ shi ⇔

∃Y1, ... , Ym ((b11 ∧ ... ∧ bkp) ∨ ... ∨ (b11 ∧ ... ∧ bkq))

Propagation rule: **ph1, ... , phi ==> g1, ..., gj | b11, ..., bkp ; ... ; b11, ..., blq.**

aonde: **{X1, ..., Xn} = vars(ph1 ∪ ... ∪ phi ∪ g1 ∪ ... ∪ gj)**

e **{Y1, ... , Ym} = vars(b1 ∪ ... ∪ bk) \ {X1, ..., Xn}**

∀X1, ..., Xn g1 ∧ ... ∧ gj ⇒ (ph1 ∧ ... ∧ phi ⇒

∃Y1, ... , Ym ((b11 ∧ ... ∧ bkp) ∨ ... ∨ (b11 ∧ ... ∧ bkq))

3.6 Semântica operacional

A semântica operacional da linguagem pode ser descrita em linguagem natural, como a seguir [15]:

- Quando uma regra disjuntiva R B1 ; ... ; Bk é disparada
  - Atualiza-se a constraint stores com B1
  - Inicia-se novo ciclo de busca e atualização

- Quando houver na Constraint Store uma Built-In com valor false ou houver uma regra que não combine com nenhuma regra definida no programa
  - Backtrack para a última disjunção Bi
  - Atualiza-se a constraint stores para seu estado anterior e depois acrescenta-se Bi
  - Atualiza-se a constraint store com Bi+1
  - Inicia-se novo ciclo de busca e atualização

- Executa-se exaustivamente todas as disjunções com todas as regras disparadas, através de backtracking.

3.7 A Versatilidade de CHR^{v}

A resolução de um problema CLP se dá através da interação entre uma máquina de inferência lógica e de componentes resolvedores de restrições (*Constraint Solvers*). Antes do desenvolvimento de CHR, um *Constraint Solvers* geralmente era um componente de software construído em linguagem de baixo nível e específico para um domínio e a resolução de um conjunto fixo de restrições. Estes componentes funcionam como "caixas-pretas" para o programador, que não é capaz de modificá-los nem extendê-lo para novos domínios.

Constraint Handling Rules (CHR) [11] é uma linguagem declarativa baseada em regras lógicas para escrita de *Constraint Solvers*. Com CHR o *Constraint Solver* é desenvolvido numa linguagem de mais alto nível, facilitando tarefas corriqueiras, como o suporte a novas classes de restrições, a novos domínios para as variáveis ou a especialização de um *Constraint Solver* para um domínio específico.

Em CHR um *Constraint Solver* é um conjunto de regras de reescrita para simplificação de restrições e regras de produção para propagação de restrições. Este *Constraint Solver* é interpretado por uma máquina de inferência CHR que deve reescrever as restrições passadas como entrada iterativamente até que se chegue num estado que seja inconsistente ou ao qual não se possam aplicar mais regras.

3.8 Abdução com CHR^{v}

O programa a seguir, escrito em CHRv, apresenta os predicados extensionais father, mother e person, e os predicados intensionais parent e

sibling. As integrity Contraints representam restrições naturais ao domínio da aplicação que devem ser satisfeitas pelos fatos conhecidos pelo programa.

```
%definition rules
parent(P, C) <=> father(P, C) ; mother(P, C).
sibling(C1, C2) <=> C1\==C2 , parent(P, C1), parent(P, C2).

%extensional introduction rule
facts ==>
father(john, mary) , father(john, peter) ,
mother(jane,mary) ,
person(john, male) , person(peter, male) ,
person(jane, female) , person(mary, female) ,
person(paul, male).

%closing rules
father(X, Y) ==> (X = john, Y = mary) ; (X = john, Y = peter).
mother(X,Y) ==> (X = jane , Y=mary).
person(X, Y) ==>
(X=john, Y=male) ; (X=peter, Y=male);
(X=jane, Y=female) ; (X=mary,Y=female) ;
(X=paul, Y=male).

%integrity constraints
father(F1, C) , father(F2,C)==>F11=F2.
mother(M1, C) , mother(M2,C)==>M1=M2.
person(P,G1) , person(P,G2)==>G1=G2.
father(F,C) ==> person(F,male),person(C,S).
mother(M,C)==>person(M, female) , person(C,G).
```

A consulta sibling(peter, mary) vai gerar multiplas tentativas de solução, as quais todas falham, com exceção de father (john, mary) father(john,peter):

```
?- sibling(peter, mary).
father(john, mary)
father(john, peter)
person(mary, female)
person(john, male)
person(peter, male)
person(john, male)
More? ;

No
```

Já a consulta sibling(paul,mary) vão exigir necessariamente que father(john,paul) ou mother(jane,paul) sejam aceitos. Entretanto, as closing rules causam a rejeição dessas tentativas, tornando toda a consulta sem sucesso:

```
?- sibling(paul,mary).

No
```

Abdução é comumente conhecida como o processo de encontrar possíveis explanações para determinado objetivo (goal) a partir de uma teoria que descreve o domínio do problema. Diferentemente do exemplo anterior, na abdução o objetivo não necessariamente é obtido a partir dos fatos conhecidos (database), mas sim de uma explanação que descreverá caminhos pelos quais o objetivo pode ser obtido a partir da database e de outras hipóteses tomadas como verdadeiras. O exemplo a seguir é idêntico ao exemplo anterior, com exceão que as closing rules dos predicados father e mother são removidos:

```
%definition rules
parent(P, C) <=> father(P, C) ; mother(P, C).
sibling(C1, C2) <=> C1\==C2 , parent(P, C1), parent(P, C2).


%extensional introduction rule
facts ==>
father(john, mary) , father(john, peter) ,
mother(jane,mary) ,
person(john, male) , person(peter, male) ,
person(jane, female) , person(mary, female) ,
person(paul, male).

%closing rules
person(X, Y) ==>
(X=john, Y=male) ; (X=peter, Y=male);
(X=jane, Y=female) ; (X=mary,Y=female) ;
(X=paul, Y=male).

%integrity constraints
father(F1, C) , father(F2,C)==>F11=F2.
mother(M1, C) , mother(M2,C)==>M1=M2.
person(P,G1) , person(P,G2)==>G1=G2.
father(F,C) ==> person(F,male),person(C,S).
mother(M,C)==>person(M, female) , person(C,G).
```

Esta nova versão do problema, abdutiva, permite encontrar várias soluções para o goal sibling(paul,mary):

```
?- sibling(paul,mary).
father(john, mary)
father(john, paul)
person(mary, female)
```

```
person(john, male)
person(paul, male)
person(john, male)
More? ;
father(peter, mary)
father(peter, paul)
person(mary, female)
person(peter, male)
person(paul, male)
person(peter, male)
More? ;
mother(jane, mary)
mother(jane, paul)
person(mary, female)
person(jane, female)
person(paul, male)
person(jane, female)
More? ;

No
```

O processo de remover as closing rules de um predicado torna o mesmo "abducível". Os demais predicados que mantém restrições de closing rules continuam gerando o mesmo comportamento de "mundo fechado" na aplicação. Por exemplo, a consulta sibling(goofty, mary) falha, visto que o predicado person não é abducível:

```
?- sibling(goofty, mary).

No
```

4 Um Sistema Especialista CHR^{v} para Diagnóstico de Falha da Rede de Telecomunicação da CHESF

Esta seção descreve o sistema especilista desenvolvido para verificar a viabilidade do uso de CHRv para diagnóstico abdutivo de falha.

4.1 Ferramentas

O desenvolvimento do sistema foi feito utilizando as seguintes ferramentas:

IDE Eclipse – Ambiente de desenvolvimento multiplataforma e multilinguagem utilizado para desenvolver tanto o programa CHR^{v} quanto o protótipo utilizado nas sessões de coleta de informação. Não confundir com EcliPse-prolog, que é um ambiente para desenvolvimento CHR/Prolog.

CHR Plugin – Para facilitar o desenvolvimento do programa CHR foi construído um plugin para o eclipse IDE que permite:

- Coloração para diferenciar variáveis, predicados, built-in e comentários
- Execução na IDE do programa SWI-prolog

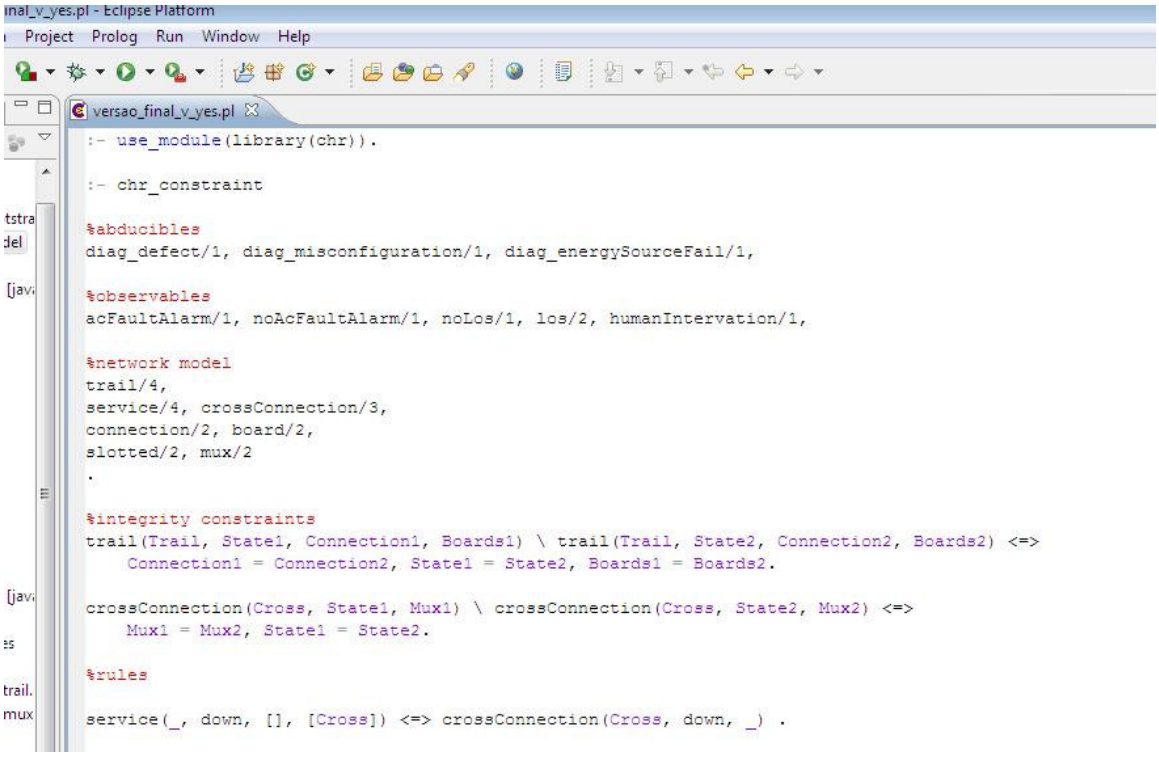

Figura 8: Ambiente de Desenvolvimento

SWI-prolog – Ambiente de execução de programas CHR/Prolog. SWI é um programa maduro (o início do seu desenvolvimento foi em 1987), Freeware sob a licença Lesser GNU, utilizado em aplicações reais tanto em pesquisas científicas e ensino quanto em aplicações comerciais.

Plataforma Java – foram desenvolvidos alguns programas Java utilizados para se prospectar informações dos especialistas. Alguns destes programas utilizaram interfaces de comunicação com o ambiente de execução CHR.

ECLiPSe-prolog – assim como SWI, o ECLiPSe foi utilizado como ambiente de execução CHR. Entretanto, devido a suas limitações, ECLiPSe foi substituído pelo SWI. As principais limitações identificadas no ECLiPSe foram:

- ECLiPSe não suporta regras com mais de duas conjunções na cabeça. Esta limitação foi considerada grave, pois limitava a expressividade do programa CHR para regras mais complexas

- ECLiPSe utiliza uma notação CHR antiga e não ANSI, de modo que todos os exemplos que eram encontrados na literatura precisavam ser traduzidos para a notação de ECLiPSe. Esta limitação foi considerada mediana.
- O mecanismo de tracing disponibilizado por ECLiPSe Prolog não é intuitivo e a documentação disponível é pouca e superficial. Esta limitação foi considerada grave.

4.2 Diagnose computacional

Diagnose computacional foi uma das primeiras aplicações desde que os computadores começaram a ficar disponíveis. Muitas áreas do conhecimento humano já utilizaram este tipo de ferramenta. Especialmente, na medicina e na engenharia elétrica o uso de sistemas especialistas para a verificação de sintomas e falhas já foi bem trabalhado.

De acordo com [16] existem os seguintes tipos de diagnose:

- Diagnose baseada em consistência
- Diagnose por abdução
- Diagnose por cobertura de conjuntos
- Diagnose hipotética-dedutiva

A estratégia utilizada neste trabalho foi a baseada em abdução.

4.3 Interface para coleta de requisitos

O desenvolvimento do sistema foi guiado pelos requisitos obtidos através da prospecção junto aos especialistas responsáveis pela operação de telecomunicação da Chesf. De modo a facilitar a coleta desses requisitos, foram desenvolvidas duas ferramentas:

Interface de configuração da rede baseada na Web – Através de formulários HTML/JavaScript e utilizando a plataforma J2EE como tecnologia no lado do servidor, a interface de configuração permite que usuário definam os equipamentos que compõe a rede de telecomunicação. Essa ferramenta permite construir mini-mundos aonde simulações podem ser realizadas, abstraindo toda a complexidade do mundo real.

A figura a seguir mostra a tela aonde é possível cadastrar uma placa, disposta (slotted) em um multiplexador (mux):

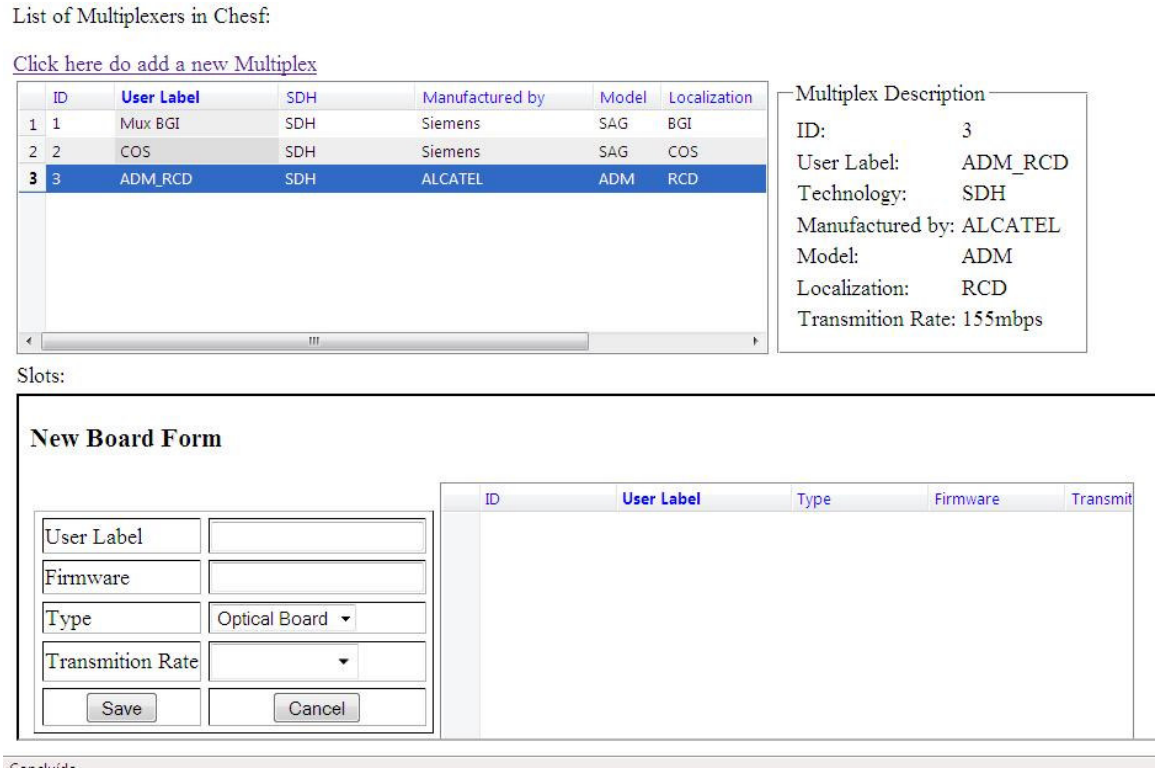

Figura 9: Interface de Configuração da Rede

Interface de simulação – Após a rede estar configurada, é possível utilizar um simulador para verificar as várias explanações possíveis para aquela configuração. Através do mouse, o usuário pode selecionar as placas que estejam sinalizando loss of signal – los. O passo seguinte é calcular as explanações possíveis para a configuração e a sinalização de los fornecida. A figura a seguir apresenta uma simulação aonde as placas 3IOL do Multiplex de

BGI (Bongi) e kkJhY do multiplex ADM_RCD (Recife II) estão com sinalizações de LOS:

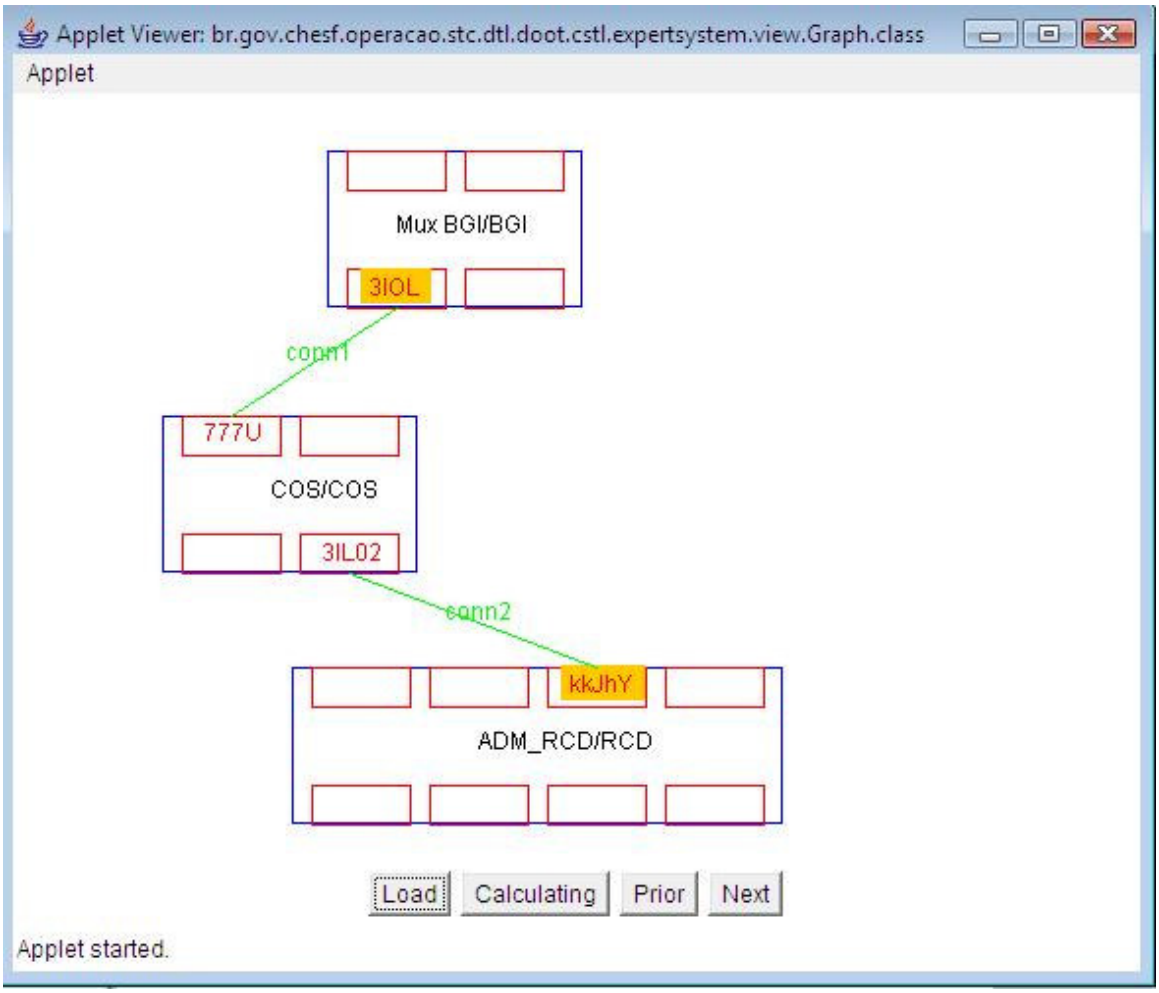

Figura 10: Simulação de Falha na rede com LOS

4.4 Regras CHR^{v} para entidade service, trail e crossconnection

As primeiras regras do programa reescrevem a ocorrência de um service down como duas possíveis causas: um dos trails associados está down ou uma das crossconnections associadas está down:

```
service(_, down, [], [Cross]) <=> crossConnection(Cross, down, _) .
```

```
service(S, down, Trails, [Cross]) <=> (
      crossConnection(Cross, down, _) ;
      service(S, down, Trails, [])
).

service(S, down, Trails, [Cross|CrossConnections]) <=>
[Cross|CrossConnections]\==[] |
(
      crossConnection(Cross, down, _) ;
      service(S, down, Trails, CrossConnections)
).

service(_, down, [Trail], []) <=> trail(Trail, down, _, _) .

service(S, down, [Trail|Trails], []) <=>
[Trail|Trails]\==[] |
(
      trail(Trail, down, _, _) ;
      service(S, down, Trails, [])
).
```

Deste modo, a primeira execução do programa, ao encontrar um serviço down, é abduzir causas parciais, como trail down ou connection down. Por exemplo:

```
      ?- service(s1, down, [trail1,trail2], [cross1, cross2]).
      crossConnection(cross1, down, _G22151)
      More? ;
      crossConnection(cross2, down, _G22276)
      More? ;
      trail(trail1, down, _G22401, _G22402)
      More? ;
      trail(trail2, down, _G22513, _G22514)
      More? ;
```

```
No
```

Semanticamente, um trail é um trecho de um circuito, composto por uma conexão física, duas placas. Um serviço é constituído de pelo menos um ou mais trails e uma ou duas crossconexões.

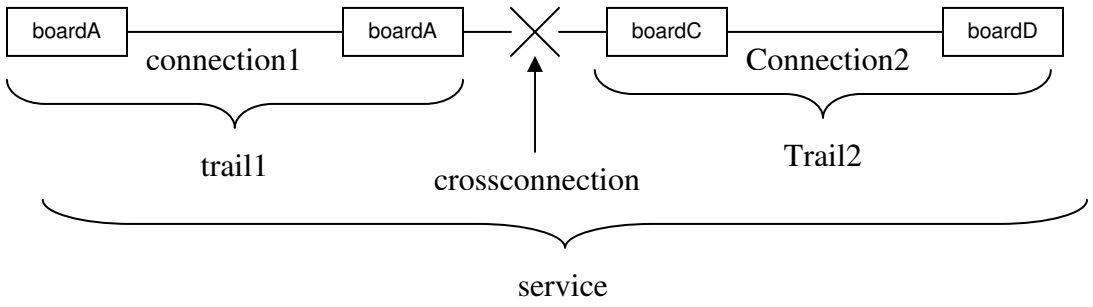

Figura 11: Configuração de um serviço

4.5 Identificando Placas defeituosas com LOS

Uma placa (board) possui dois circuitos, um que envia dados (TX) e outro que recebe dados (RX). Quando uma placa percebe que não está mais recebendo dados (através do circuito RX) ela envia uma sinalização para a gerência. Esta sinalização é chamada de loss of signal, ou simplismente, los. Somado a isso, quando uma placa percebe que não está mais recebendo dados, por segurança, ela pára de enviar dados também. No programa, o los foi modelado com um predicado los(Board, Time) que sinaliza que a placa Board enviou um sinal de los no instante Time.

Quando uma placa apresenta defeito, na maioria dos casos (os casos cobertos neste trabalho), ela não mais consegue enviar dados, mas continua recebendo, ou capaz de receber. Por exemplo, nas placas interligadas na figura a seguir:

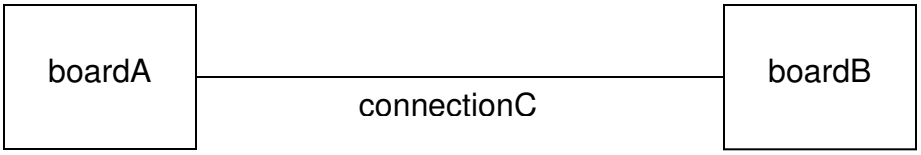

Figura 12: Placas boardA e boardB interconectadas

Caso a placa boardA apresente defeito, deixará de enviar dados. Neste caso, a placa boardB irá perceber que não está mais recebendo dados e sinalizará sinal de los(boardB, TimeB). Além disso, a placa boardB irá parar de enviar dados (por segurança) de modo que a placa boardA também não mais receberá dados. Assim, a placa boardA irá sinalizar um sinal de los(boardA, TimeA). Desde que a placa boardB sinalizou los antes da placa boardA, é seguro que TimeA < TimeB. Deste modo, a causa das sinalizações de los em ambos os terminais de uma connection poderá ser diagnosticada através da inspeção dos tempo TimeA e TimeB.

O diagnóstico de defeito em placa foi modelado no programa com as seguintes regras:

```
trail(_, down, _, [BoardA, BoardB]), los(BoardA, TimeA), los(BoardB,
TimeB) <=>
TimeA - TimeB > 1 |     diag_defect(BoardA) .

trail(_, down, _, [BoardA, BoardB]), los(BoardA, TimeA), los(BoardB,
TimeB) <=>
TimeB - TimeA > 1 |     diag_defect(BoardB) .
```

Na prática, os tempos de sinalização não são exatos. Por isso, foi considerado um tempo de 1 unidade, de forma que essas incertezas sejam desconsideradas.

Fazendo a seguinte consulta ao programa tem-se, entre os resultados:

```
?- service(s1, down, [trail1,trail2], [cross1,
cross2]),trail(trail1, _, _,[b1,b2]),los(b1,10), los(b2,14).
...
diag_defect(b2)
...
```

Observar que o tempo de los de b2 (14) foi maior que o tempo de b1 (10).

4.6 Regras para defeitos em connection

Observando mais uma vez a figura 12, pode-se perceber que, havendo um defeito neste elemento, haverá a perda de dados TX e RX em ambas as placas. Neste caso, as placas boardA e boardB irão sinalizar respectivamente los(BoardA, TimeA) e los(boardB, TimeB) quase que no mesmo instante. Esse diagnóstico foi modelado no programa com a seguinte regra:

```
trail(_, down, Connection, [BoardA, BoardB]), los(BoardA, TimeA),
los(BoardB, TimeB) <=>
      abs(TimeA - TimeB) =< 1 | diag_defect(Connection) .
```

Segue um exemplo da aplicação desta regra:

```
?- service(s1, down, [trail1], []),trail(trail1, _,
connectionC,[b1,b2]),los(b1,10), los(b2,11).
diag_defect(connectionC)
More? ;
```

```
No
```

4.7 Regras para diagnose de falha em Multiplexadores (Mux)

Multiplexadores, no contexto deste trabalho, são equipamentos que suportam as placas, física e logicamente. Um multiplex, ou Mux, pode suportar até dezenas de placas de comunicação, a depender de seu modelo. Uma falha em um mux causará a indisponibilidade de todas as placas suportadas por ele, levando a perda de todos os serviços associados a estas placas.

As causas de falhas contempladas neste estudo são:

Defeito interno no mux: queima ou falha de um ou mais circuitos internos

Falta de AC: perda de alimentação AC na instalação, de modo que o mux não possa mais suportar os circuitos das placas.

O comportamento diagnóstico do programa, em relação às falhas do multiplex, foi mapeado nas seguintes regras:

```
trail(_, down, _, [BoardA, BoardB]), slotted(BoardA, Mux),
noLos(BoardA), los(BoardB, _) <=> mux(Mux, down) .

trail(_, down, _, [BoardA, BoardB]), slotted(BoardB, Mux),
noLos(BoardB), los(BoardA, _) <=> mux(Mux, down) .

mux(Mux, down), noAcFaultAlarm(Mux) <=>   diag_defect(Mux) .

mux(Mux, down), acFaultAlarm(Mux) <=> diag_energySourceFail(Mux) .
```

O predicado slotted(BoardA, Mux) representa o fato de uma placa Board está sendo suportada por um mux Mux.

O predicado noLos(Board) representa o fato de uma placa não está sinalizando los. Caso uma placa boardA esteja sinalizando los e a placa boardB,

interconectada a boardA, não estiver sinalizando los, indica que a placa boardB está inativa. Neste trabalho, estas observações foram mapeadas em uma falha no multiplex que mantém a placa boardB (representado pelo predicado mux(Mux, down).

As razões mapeadas para um mux está fora do ar foram mapeadas nos seguintes predicados: diag_defect(Mux) e diag_energySourceFail(Mux). Um defeito em um mux, representado por diag_defect(Mux), será caracterizado sempre que não houver falha de alimentação AC, representado por noAcFaultAlarm(Mux). A falha de alimentação AC serácaracterizada por um multiplex e na presença de sinalização de acFaultAlarm(Mux).

A seguir são apresentadas duas consultas que exercitam as regras mencionadas neste item:

```
1 ?- service(s1, down, [trail1], []),trail(trail1, _,
connectionC,[b1,b2]),slotted(b1,m1), noLos(b1), los(b2,_),
noAcFaultAlarm(m1).
diag_defect(m1)
More? ;

No
2 ?- service(s1, down, [trail1], []),trail(trail1, _,
connectionC,[b1,b2]),slotted(b1,m1), noLos(b1), los(b2,_),
acFaultAlarm(m1).
diag_energySourceFail(m1)
More? ;

No
```

4.8 Regras específicas para cross-conections

Dentro do multiplex, as cross-conection tem a responsabilidade de interligar uma placa a outra. Cross-connections são componentes lógicos, responsáveis por rotear os dados de uma placa a outra.

Falhas em cross-conections não são rotineiras. Quando há uma falha no hardware que suporta a cross-connection, não é considerada a falha da cross-connection, e sim a falha do multiplex. Entretanto, a ocorrência de defeitos por conta de cross-connections são associados a falhas de configuração por ação humana. Neste sentido, uma falha de cross-connection poderá ser considerada caso haja a presença do agente humano intervindo no equipamento ou gerência. Estes requisitos foram mapeados nas seguintes regras:

```
crossConnection(CrossConnection, down, Mux), humanIntervation(Mux) <=>
diag_misconfiguration(CrossConnection) .
```

A aplicação dessas regras pode ser verificada através da seguinte consulta:

```
?- service(s1, down, [t1, t2], [c1]), humanIntervation(mux),
crossConnection(c1, _, mux).
diag_misconfiguration(c1)
More? ;
...
More? ;

No
```

O predicado diag_misconfiguration representa o diagnóstico de falha na configuração de uma cross-connection pela ação de um agente humano.

4.9 Programa completo

O programa completo é composto pelas regras previamente apresentadas, adicionadas às regras de fechamento (closing rules). Foram definidas closing rules para todos os predicados, com exceção dos predicados abducíveis (diag_defect, diag_energySourceFail). Este é um requisito para o desdobramento do raciocínio abdutivo, conforme citado por [10] e [6].

Além das regras de fechamento, foram adicionadas também as regras de integridades (integrity contraints), responsáveis por garantir a unicidade das entidades service, trail e cross-connection.

```
:- use_module(library(chr)).

:- chr_constraint

% Sistema Especialista para diagnóstico de falhas em redes de
telecomunicações da Chesf
% Autor: Luiz Carlos d´Oleron
% email: lcadb@cin.ufpe.br
% Data: 24/11/2008

%abducibles
diag_defect/1, diag_energySourceFail/1, diag_misconfiguration/1,

%net
trail/4,
service/4, crossConnection/3,
mux/2, slotted/2,

%observables
los/2, noLos/1, noAcFaultAlarm/1, acFaultAlarm/1, humanIntervation/1
.

%integrity constraints

service(Service, State1, Trails1, CrossConnections1) \ service
(Service, State2, Trails2, CrossConnections2) <=>
      State1 = State2, Trails1 = Trails2, CrossConnections1 =
CrossConnections2.

trail(Trail, State1, Connection1, Boards1) \ trail(Trail, State2,
Connection2, Boards2) <=>
      Connection1 = Connection2, State1 = State2, Boards1 = Boards2.

crossConnection(Cross, State1, Mux1) \ crossConnection(Cross, State2,
Mux2) <=>
      Mux1 = Mux2, State1 = State2.
```

```
%Constraint Theory

service(_, down, [], [Cross]) <=> crossConnection(Cross, down, _) .

service(S, down, Trails, [Cross]) <=> (
      crossConnection(Cross, down, _) ;
      service(S, down, Trails, [])
).

service(S, down, Trails, [Cross|CrossConnections]) <=>
[Cross|CrossConnections]\==[] |
(
      crossConnection(Cross, down, _) ;
      service(S, down, Trails, CrossConnections)
).

service(_, down, [Trail], []) <=> trail(Trail, down, _, _) .

service(S, down, [Trail|Trails], []) <=>
[Trail|Trails]\==[] |
(
      trail(Trail, down, _, _) ;
      service(S, down, Trails, [])
).

trail(_, down, _, [BoardA, BoardB]), los(BoardA, TimeA), los(BoardB,
TimeB) <=>
TimeA - TimeB > 1 |     diag_defect(BoardA) .

trail(_, down, _, [BoardA, BoardB]), los(BoardA, TimeA), los(BoardB,
TimeB) <=>
TimeB - TimeA > 1 |     diag_defect(BoardB) .

trail(_, down, Connection, [BoardA, BoardB]), los(BoardA, TimeA),
los(BoardB, TimeB) <=>
abs(TimeA - TimeB) =< 1 | diag_defect(Connection) .

trail(_, down, _, [BoardA, BoardB]), slotted(BoardA, Mux),
noLos(BoardA), los(BoardB, _) <=> mux(Mux, down) .

trail(_, down, _, [BoardA, BoardB]), slotted(BoardB, Mux),
noLos(BoardB), los(BoardA, _) <=> mux(Mux, down) .

mux(Mux, down), noAcFaultAlarm(Mux) <=>   diag_defect(Mux) .

mux(Mux, down), acFaultAlarm(Mux) <=> diag_energySourceFail(Mux) .

crossConnection(CrossConnection, down, Mux), humanIntervation(Mux) <=>
diag_misconfiguration(CrossConnection) .


%closing rules
service(Name, _, Trails, CrossConnections) ==> (Name = service1, Trails
= [trail1, trail2], CrossConnections = [cross1]) ;

(Name = service2, Trails = [trail3], CrossConnections = []) .
service(_, State, _, _) ==> (State = up) ; (State = down) .
```

```
trail(Name, _, _, _) ==> (Name = trail1) ; (Name = trail2) ; (Name =
trail3) .
trail(_, State, _, _) ==> (State = up) ; (State = down) .


crossConnection(Name, _, Mux) ==> (Name = cross1, Mux = mux2) .
crossConnection(_, State, _) ==> (State = up) ; (State = down) .

mux(Name, _) ==> (Name = mux1) ; (Name = mux2) ; (Name = mux3).
mux(_, State) ==> (State = up) ; (State = down) .

slotted(Board, Mux) ==> (Board = boardA, Mux = mux1) ; (Board = boardB,
Mux = mux1) ;
                                    (Board = boardC, Mux = mux2) ;
(Board = boardD, Mux = mux2) ;
                                    (Board = boardE, Mux = mux3) ;
(Board = boardF, Mux = mux3) .

los(Board, _) ==> (Board = boardA) ; (Board = boardB) ; (Board =
boardC) ; (Board = boardD) ; (Board = boardE) ; (Board = boardF) .

noLos(Board) ==> (Board = boardA) ; (Board = boardB) ; (Board = boardC)
; (Board = boardD) ; (Board = boardE) ; (Board = boardF) .

noAcFaultAlarm(Mux) ==> (Mux = mux1) ; (Mux = mux2) ; (Mux = mux3).
acFaultAlarm(Mux) ==> (Mux = mux1) ; (Mux = mux2) ; (Mux = mux3).
humanIntervation(Mux) ==> (Mux = mux1) ; (Mux = mux2) ; (Mux = mux3).
```

5 Conclusão

A seguir serão apresentadas as conclusões finais mostrando as principais contribuições obtidas a partir desse trabalho, dificuldades encontradas para sua realização e trabalhos futuros.

5.1 Principais contribuições

As principais contribuições desse trabalho foram:

Verificação prática de CHRv em projeto real – O grupo de pesquisa do CIn no qual este projeto foi desenvolvido possuía uma forte base de

conhecimento teórico no uso de CHRv, mas faltava ainda a verificação de uma solução real para um problema real, escrita em CHRv. Este trabalho permitiu que essa experiência fosse vivenciada, servindo como base para outros futuros trabalhos de pesquisa aplicada junto à industria.

Verificação prática de técnica de abdução em problema real – O mesmo se aplica à técnica de abdução no que foi evidenciado sobre o uso de CHRv. Entretanto, o uso da abdução era ainda mais desconhecido. Por exemplo, o problema da complexidade e intratabilidade de soluções abdutivas não era ainda levado em conta, nas discussões do grupo de pesquisa.

Maturidade no uso de ferramentas CHR/Prolog – Durante a execução do trabalho foram usadas duas implementações distintas de máquinas CHR/Prolog. Neste sentido, foram identificados fraquezas e pontos fortes nas ferramentas de forma a subsidiar o uso de uma determinada implementação. Antes desse trabalho, o grupo de pesquisa ORCAS adotava a implementação EcliPse-prolog. Após a execução desse trabalho, os novos projetos do grupo são implementados utilizando a implementação SWI-prolog.

Enriquecimento e refinamento do conhecimento implícito e explícito por parte dos especialistas – Durante o processo de aquisição de conhecimento dos especialistas, estes precisaram contestar fatos considerados antes irrefutáveis. Este processo serviu para o amadurecimento dos conhecimentos da própria equipe técnica de suporte em tempo real de telecomunicações da Chesf, sendo identificadas algumas melhorias e nivelamento entre a equipe.

Subsídio para especificação técnica de aquisição da Chesf – A Chesf especificou a aquisição de um sistema de integração de gerências e serviços o qual deverá permitir, entre outras coisas, a correlação de alarmes e identificação de causas, baseado na sinalização das gerências. O presente trabalho e a

pesquisa ao redor dele serviram de suporte para a definição dessa especificação.

5.2 Limitações

Nas duas implementações que foram utilizadas neste trabalho, foram avaliadas suas interfaces com outras linguagens de programação que permitissem o desenvolvimento de uma solução amigável, baseada em janelas e botões, de forma a melhorar a experiência da implementação por parte dos especialistas.

SWI-prolog – Permite a troca de mensagens com programas desenvolvidos tanto em C/C++ quanto em Java. A interface JPL do SWI possui tanto a versatilidade de se invocar consultas CHR/Prolog á máquina SWI quanto se invocar objetos Java e seus métodos a partir de predicados built-in em um programa Prolog/CHR.

Eclipse-prolog – Assim como o SWI, o EcliPse Prolog possui uma interface Java-Prolog que permite efetuar consultas à máquina EcliPse. Entretanto, diferentemente de SWI, no EcliPse não existe uma interface que permita invocar objetos Java através de um programa EcliPse.

Nas duas implementações testadas, não foi encontrado uma forma transparente de se extrair regras que permanecem na Constraint Store, após a execução do programa CHR. Essa dependência fracassou a tarefa de se construir uma GUI Java capaz de apresentar os predicados abduzidos após a execução do programa CHR.

5.3 Trabalhos Futuros

Geralmente, programas que utilizam raciocínio abdutivo tendem a propor muitas explanações, algumas delas pouco plausíveis. Uma abordagem que possa unir abdução com probabilidade, a fim de identificar explanações mais factíveis dentre todas as explanações possíveis (ou ignorar explanações pouco prováveis) é de grande importância na construção de programas reais de diagnóstico de falhas.

Neste sentido, o trabalho de [17] oferece uma importante contribuição para o desenvolvimento de aplicações abdutivas que podem se beneficiar da probabilidade. Este trabalho fornece duas interessantes contribuições:

- Escolher qual regra ativar dependendo da usa probabilidade, permitindo a priorização das explanações, de modo que soluções mais factíveis possam ser obtidas mais cedo,
- Permitir o retorno do controle para outra seqüência de execução do backtraking, cancelando ramos de solução considerados heuristicamente pouco prováveis, de modo que menos computação seja realizada.

O tratamento probabilístico é realizado através de uma função de prioridade definida pelo desenvolvedor, no próprio programa CHR.

Em particular, o presente trabalho de construção de um sistema especialista baseado em abdução poderá ser enriquecido com o tratamento de incertezas e priorização probabilísticas.

Referências

1. Chesf, Companhia Hidro Elétrica do São Francisco. Março, 2008. Disponível em: http://www.chesf.gov.br

2. Sistema Aberto de Supervisão e Controle. Março, 2008. Disponível em: http://www.sage.cepel.br

3. Feigenbaum, E. A., Buchanan, B.G., e Lederberg, J., 1971. On generality problem solving: A case study using the DENDRAL program. Edinburg University Press, Edinburgh, Scotland.

4. Russel, S e Novig, P. Artificial Inteligence: A Modern Approach. 2a. edição, Prentice Hall, 2003.

5. Charles Sanders Peirce, Abril, 2008. Disponível em: http://plato.stanford.edu/entries/peirce

6. Kakas, A. C., Kowalski, R. A., Toni, F., The Role of Abduction in Logic Programming. In Gabbay, D. M., Hogger, C. J., & Robinson, J. A. (Eds.), Handbook of logic in Artificial Intelligence and Logic Programming, Vol. 5, pp. 235--324. Oxford University Press.

7. Peirce, C.S., Collected papers of Charles Sandres Peirce. Vol.2, 1931-1958, Hartshorn et al. eds.; Harvard University Press

8. Bylander, T., Dean Allemang , Michael C. Tanner , John R. Josephson, The computational complexity of abduction, Artificial Intelligence, v.49 n.1-3, p.25-60, Maio 1991.

9. PC AI Glossary. Disponível em:
http://www.pcai.com/web/glossary/pcai_d_f_glossary.html

10. Abdennadher, S., Rule-based Constraint Programming: Theory and Practice , Habilitation, Institut für Informatik, Ludwig-Maximilians-Universität München, 2001.

11. Fruhwirth, T. 1998. Theory and Practice of Constraint Handling Rules, Special Issue on Constraint Logic Programming. Journal of Logic Programming, 95-138.

12. A.C.Kakas, A.Michael, C.Mourlas. Abductive Constraint Logic Programming. Technical Report, University of Cyprus, 1998

13. Robin, J. P. L. ; Vitorino, J. . ORCAS: Towards a CHR-Based Model-Driven Framework of Reusable Reasoning Components. In: 20th Workshop on Logic Programming (WLP'2006), 2006, Vienna. Proceedings of the 20th Workshop on Logic Programming (WLP'2006), 2006.

14. Meta-Object Facility Specification, Object Management Group Inc. Março, 2003. Disponível em: http://www.omg.org.

15. Robin, J. P. L. Constraint Handling Rules (CHR):
Rule-Based Constraint Solving and Deduction. Disponível em:
www.cin.ufpe.br/~mact/mestrado/rarcs/CHR.ppt

16. Peter Lucas, Symbolic diagnosis and its formalisation, The Knowledge Engineering Review, v.12 n.2, p.109-146, Junho, 1997 .

17. Henning Christiansen. Prioritized Abduction with CHR. In T. Schrijvers, F. Raiser, and T. Frühwirth, editors, CHR '08: Proc. 5th Workshop on

Constraint Handling Rules, Hagenberg, Austria, pages 159-173, July 2008. RISC Report Series 08-10, University of Linz, Austria.