THIAGO CARVALHO PEDRAZZI

UM AMBIENTE DE DESENVOLVIMENTO BASEADO EM TABELAS DE DECISÃO ADAPTATIVAS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia

São Paulo
2007

THIAGO CARVALHO PEDRAZZI

UM AMBIENTE DE DESENVOLVIMENTO BASEADO EM TABELAS DE DECISÃO ADAPTATIVAS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia

Área de Concentração:
Sistemas Digitais

Orientador: Prof. Dr. Ricardo Luis de Azevedo da Rocha

São Paulo
2007

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de maio de 2007.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Pedrazzi, Thiago Carvalho
Um ambiente de desenvolvimento baseado em tabelas de decisão adaptativas / T.C. Pedrazzi. -- São Paulo, 2007.
112 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Aprendizagem computacional 2.Linguagem de programação 3.Tomada de decisão I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Aos meus pais.

AGRADECIMENTOS

Principal e especialmente ao Prof. Dr. Ricardo Luis de Azevedo da Rocha, meu orientador, que sempre com suas palavras de incentivo e compreensão, além de paciência e temperança, soube compreender as minhas dificuldades e orientar-me através dos desafios desta nova busca. Agradeço também à pessoa Ricardo, um exemplo dentro e fora do ambiente acadêmico.

Aos meus pais, Luiz e Sonia, que sempre me incentivaram a me esforçar e tentar sempre me superar.

Aos familiares e amigos, que sempre me apoiaram e incentivaram. Mesmo aquelas pessoas que não compreenderam esta fase da caminhada...

Ao Prof. Dr. João José Neto, uma referência em diversos aspectos, não somente restrita ao campo científico.

Ao Prof. Dr. Paulo Sergio Muniz Silva pela participação na banca de exame de qualificação, pelos comentários pertinentes e valiosos.

E a todas as pessoas que, direta ou indiretamente, contribuíram para que este trabalho fosse terminado em tempo hábil. Especialmente a amiga Profª. MSc. Ivone Penque Matsuno, companheira de disciplinas e artigos.

RESUMO

Este trabalho apresenta uma aplicação da tecnologia adaptativa, de forma viável e prática, ao processo de tomada de decisão encontrado nas organizações e nos sistemas de apoio à decisão. Para alcançar esse fim, é proposto um ambiente para desenvolvimento de aplicações baseado em tabelas de decisão adaptativas. Desta forma um usuário leigo nos conceitos de computação, mas que tenha conhecimento do processo de decisão e do uso de tabelas de decisão, possa compreender, validar, modificar e mesmo desenvolver programas nesta ferramenta. Os principais resultados obtidos neste estudo foram a definição e construção de dispositivos adaptativos, em especial, as tabelas de decisão adaptativas, em linguagem funcional; e a definição de uma linguagem de entrada para a ferramenta proposta, classificada como uma linguagem de domínio específico.

Palavras-chave: Aprendizagem computacional. Tecnologia Adaptativa. Linguagem de programação. Tomada de decisão.

ABSTRACT

This work presents an application of adaptive technology to the decision making process applied in the business management and Decision Support Systems. To accomplish this goal, a development environment based on adaptive decision tables is proposed. This way, a user who is not familiar to the computing concepts, but that does know the decision making process and the use of decision tables, can understand, change and even develop applications in this environment. The main results obtained from this study were the implementation of adaptive devices, specifically the adaptive decision tables, in functional languages, and the definition of an input language to the development environment proposed, that can be classified as a domain specific language.

Keywords: Computer learning. Adaptive technology. Programming languages. Decision making.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A humanidade está em crescente evolução tecnológica. A cada vez mais elevada quantidade de dispositivos controlados eletronicamente e a sua aplicação em áreas de alta complexidade fez surgir a necessidade de desenvolvimento de técnicas de projetos de sistemas que contemplem este aumento de complexidade.

O aumento da complexidade dos sistemas ocorre, basicamente, em função de duas características: a automatização de tarefas intrinsecamente complexas, que inviabiliza a utilização da análise de todas as possibilidades; e a necessidade de definir a solução ótima, no caso de sistemas que sejam diretamente responsáveis ao fator humano (RUSSEL; NORVIG, 1995).

A utilização do método clássico de projeto de sistemas em determinadas áreas é inviabilizada pelo número de variáveis e seus possíveis valores. Esta característica causaria uma explosão em escala exponencial das possibilidades a serem analisadas, além disso, o dinamismo do ambiente e a velocidade com que ele se altera, tornam muito difícil e complexa a tomada de decisão (MITCHELL, 1997).

Quando a lacuna entre as necessidades práticas de controle necessárias ao mundo real e as práticas clássicas de projeto de sistemas se estendeu, uma área de pesquisa interdisciplinar denominada aprendizagem de máquina expandiu-se (MITCHELL, 1997).

A área de aprendizagem de máquina preocupa-se com a questão de como desenvolver sistemas computacionais que melhorem automaticamente com a experiência. Durante os últimos anos foram desenvolvidas muitas aplicações bem-sucedidas destas técnicas. Estas aplicações abrangem desde sistemas de mineração de dados que aprendem a detectar transações fraudulentas de cartões de créditos, quanto aos sistemas de filtragem de informações que ‘aprendem’ as preferências de leitura do usuário, e até mesmo sistemas para o controle de veículos automotores que conseguem dirigir autonomamente em rodovias (MITCHELL, 1997).

Ao longo do tempo, esta linha de pesquisa cada vez mais importante nos dias atuais apresentou diversas tecnologias para a solução de problemas específicos. Como exemplo, pode-se citar as redes neurais artificiais, em suas mais diversas

variedades, os algoritmos genéticos, os quais se baseiam na característica de ‘sobrevivência do mais apto’, encontrada na natureza, e as denominadas técnicas bayesianas, as quais são baseadas, total ou parcialmente, no teorema de Bayes (RUSSEL; NORVIG, 1995).

1.1 MOTIVAÇÃO

O acesso a cada vez mais informações e a facilidade de armazená-las e manipulá-las estimula a utilização de sistemas que ajudem o processo de tomada de decisão nas organizações. Tais sistemas, denominados Sistemas de Apoio à Decisão, fazem uso de diversas técnicas na manipulação da informação armazenada, incluindo algumas técnicas de aprendizagem de máquina. Estas técnicas são utilizadas na busca por soluções quando o ambiente é caótico, ou quando não há a disponibilidade a toda informação necessária ao processo clássico de tomada de decisão.

Estes sistemas estão se tornando cada vez mais importantes nas organizações modernas, devido justamente ao aumento da quantidade de informação disponível para a tomada de decisão, e adicionalmente, a outros fatores, como o aumento da competitividade do ambiente empresarial (SHIMIZU; CARVALHO; LAURINDO, 2006).

A qualidade das decisões tomadas, especialmente as estratégicas, afeta diretamente o futuro da organização, sendo naturalmente responsável pela situação de lucro ou prejuízo, podendo até mesmo causar a falência da mesma. Conclui-se que, quão melhor for a qualidade da decisão tomada, maior a possibilidade de sucesso da organização (SHIMIZU, 2001).

O aumento da possibilidade de lucro da organização, e por conseqüência do seu sucesso, justifica o emprego cada vez maior de recursos na pesquisa da área de tomada de decisão nas empresas nos últimos anos. Desta forma, a utilização de técnicas computacionais cada vez mais sofisticadas para o tratamento dos problemas torna-se uma vantagem competitiva interessante.

O problema central, o objeto de estudo nesta pesquisa, é encontrar alternativas viáveis e práticas para a questão do apoio à tomada de decisão em organizações, ou seja, pretende-se obter um dispositivo cuja interface com os indivíduos que tomam as decisões seja simples e intuitiva, e que apresente comportamento similar aos clássicos utilizados em inteligência de máquina.

O uso da tecnologia adaptativa na solução de problemas em diversas áreas do conhecimento humano (PISTORI, 2003) apresenta resultados que podem ser considerados extremamente satisfatórios. Tais resultados são, em sua maioria, compatíveis com o resultado de técnicas clássicas de aprendizagem de máquina, apresentando, além disso, características muito desejáveis para o desenvolvimento de sistemas comerciais: a relativa simplicidade de manipulação do dispositivo, em comparação a outras técnicas, e o baixo custo no projeto e desenvolvimento de tais sistemas.

1.2 OBJETIVOS

Sistema de Apoio à Decisão é um conjunto de procedimentos baseados em modelos para o processamento de dados e a análise de problemas, tendo como finalidade assistir aos administradores no processo de tomada de decisão (SHIMIZU, 2001).

Um Sistema de Apoio à Decisão é composto, resumidamente, por três componentes principais: *um sistema de linguagem*, que permita a comunicação entre os usuários do Sistema de Apoio à Decisão e seus subsistemas; *um sistema de conhecimento*, que é responsável pelo armazenamento dos dados, dos procedimentos e das informações inteligentes disponíveis; e *um sistema para o processamento de problemas*, que estrutura e executa os modelos na busca de soluções para os problemas apresentados (SHIMIZU, 2001), (TURBAN; ARONSON, 2001).

O objetivo geral do presente trabalho é apresentar uma aplicação da tecnologia adaptativa no processo de tomada de decisão encontrado nas organizações e nos Sistemas de Apoio à Decisão. Este objetivo será atingido com

uma proposta de projeto de um módulo para o processamento de problemas, baseado na tecnologia adaptativa, bem como a sua implementação.

Os objetivos específicos são:

- Definir uma linguagem de acesso e programação a uma tabela de decisão adaptativa, simples, didática e próxima da realidade empresarial;
- Construir o compilador da linguagem de entrada definida no item anterior;
- Projetar, construir e testar uma tabela de decisão adaptativa, que será o modelo subjacente de apoio à decisão;

Como a utilização da tabela de decisão isoladamente torna-se pouco viável, algumas estruturas adicionais foram desenvolvidas com o intuito de facilitar a sua operação. Uma linguagem de entrada simplificada para o módulo foi projetada, bem como uma estrutura utilizada como repositório de dados, visando à completude do projeto como uma ferramenta independente. Ressalta-se, no entanto, que estas estruturas são simplificadas ao máximo, de forma que o módulo, parte principal do trabalho, possa ser utilizado sem a construção completa de um Sistema de Apoio à Decisão, o que foge do escopo do trabalho.

1.3 ORGANIZAÇÃO DO TRABALHO

Além do presente capítulo, que trata da introdução do trabalho, outros seis capítulos, divididos em duas partes, são utilizados na distribuição do mesmo.

A primeira parte, responsável pela apresentação dos fundamentos utilizados no projeto é composta de três capítulos. Um capítulo responsável pela introdução do conceito de tomada de decisão, um capítulo responsável pela tecnologia adotada na proposta, no caso a tecnologia adaptativa, e um capítulo responsável pela forma utilizada na implementação.

A segunda parte trata da apresentação da solução proposta. A especificação do projeto, a descrição de sua estrutura, bem como as justificativas das técnicas e

tecnologias utilizadas são apresentadas no quinto capítulo do presente trabalho. O sexto capítulo cuida dos aspectos de implementação da solução proposta no capítulo anterior, juntamente com a parte experimental do trabalho, mostrando a solução de alguns problemas interessantes.

Finalmente, no sétimo e último capítulo, tem-se a conclusão do trabalho, a apresentação crítica dos benefícios gerados pelo processo de produção deste trabalho, considerações futuras e possíveis desdobramentos e novas idéias de pesquisa.

2 TOMADA DE DECISÃO

A tomada de decisão sempre esteve presente na vida do ser humano. Desde os primórdios, o homem deveria tomar uma decisão a respeito do que se deveria caçar, pescar ou colher alimentos, embora as decisões neste nível fossem muito mais baseadas nos instintos do que na ponderação a respeito do que poderia trazer o máximo de benefícios ao longo do tempo.

Conforme a humanidade evoluiu, ambientes mais complexos foram criados pelo próprio homem, o que acarretou diretamente no aumento da necessidade de tomada de decisão por parte das pessoas. Na vida contemporânea diversas situações de tomada de decisão surgem perante as pessoas, muitas vezes no mesmo dia, deviso justamente a esse aumento de complexidade do ambiente. Uma decisão pode envolver simplesmente a escolha entre estudar, nadar ou jogar golfe como forma de passar um dia. Não importa qual alternativa é escolhida, somente que uma decisão foi tomada (PIDD, 2001).

Portanto, decisão é uma escolha feita entre uma ou mais alternativas disponíveis. Escolher a melhor alternativa para alcançar os objetivos é, em seu sentido mais restrito, tomar uma decisão (CERTO, 1994).

No mundo moderno, por força da distribuição geográfica e principalmente por fatores socioeconômicos, a humanidade é distribuída em organizações, sejam elas com fins lucrativos, governamentais ou de quaisquer outros tipos. Para alcançar os objetivos de uma organização de maneira otimizada, as decisões tomadas ao longo da existência da empresa devem ser as mais acertadas possíveis. Assim sendo, nada mais interessante do que utilizar as diversas técnicas para a tomada de decisão no processo decisório de uma empresa (CERTO, 1994).

2.1 DECISÃO NAS ORGANIZAÇÕES

Uma organização freqüentemente se encontra diante de sérios problemas de decisão. Devido à magnitude e à complexidade destes problemas, que quase

sempre apresentam riscos e incerteza, o processo de tomada de decisão baseado em um único indivíduo é praticamente inviável. Desta forma há invariavelmente a necessidade da opinião e participação de diversas pessoas, das mais variadas áreas e níveis funcionais. Além disso, o processo de decisão em uma empresa ou organização deve ser estruturado e resolvido de modo formal, detalhado, consistente e transparente, de modo a permitir a sua contínua melhoria (SHIMIZU, 2001).

Diariamente, os decisores[1] são responsáveis por tomadas de decisão e pela comunicação destas aos outros membros da organização. No entanto, nem todas as decisões têm a mesma importância para a organização. Algumas afetam um grande número de membros da organização, custam muito dinheiro para serem colocadas em prática, ou têm um efeito a longo prazo na organização (CERTO, 1994), (SHIMIZU, 2001).

Para a empresa, talvez não seja interessante ter apenas a melhor decisão no momento, mas também conhecer as outras possíveis alternativas da decisão. A empresa poderia contentar-se com uma boa decisão dentro de suas possibilidades, ou com a segunda melhor decisão. Essa análise da variação das possibilidades em torno de uma decisão escolhida é possível por meio de uma análise de sensibilidade das estratégias ou alternativas de decisão (SHIMIZU, 2001).

Como pode ser observado, o processo de tomada de decisão em uma organização normalmente não é trivial. Em empresas de médio e grande porte, a tomada de decisão de qualidade sem uma ferramenta de auxílio torna-se extremamente complexa, devido a diversos fatores, como os apresentados acima. Portanto, a utilização de sistemas computacionais na simplificação, automatização e análise de informações justifica-se pelo ganho de velocidade e qualidade na execução de tais tarefas (TURBAN; ARONSON, 2001), (SHIMIZU; CARVALHO; LAURINDO, 2006).

[1] No caso mais genérico, todos somos decisores, visto que tomamos diversas decisões ao longo da nossa vida. No sentido mais restrito apresentado aqui, decisor é uma pessoa (ou grupo de pessoas) responsável por escolher uma dentre muitas alternativas disponíveis para a solução de um problema, utilizando o seu conhecimento e informações, além de quaisquer outros métodos que julgue necessário (CERTO, 1994).

2.2 O PROCESSO DE TOMADA DE DECISÃO

Uma decisão é a escolha de uma alternativa dentre um conjunto de alternativas disponíveis. O processo de tomada de decisão é composto pelos passos que o decisor deve seguir na escolha de uma alternativa. Obviamente, o processo empregado pelo decisor na tomada de decisão tem impacto significante na qualidade da decisão tomada. Os decisores fazendo uso de um processo organizado e sistemático, aumentam a probabilidade de que suas decisões sejam melhores, se comparada ao caso do processo ser desorganizado e não-sistemático. A organização e sistematização do processo de tomada de decisão permitem o aprendizado com o histórico dos eventos anteriores e facilita a percepção de possíveis otimizações, sejam elas locais a um passo ou ao processo como um todo (CERTO, 1994), (PIDD, 2001).

O modelo de processo de tomada de decisão recomendado para o uso dos decisores é mostrado na figura 1. Em ordem de ocorrência, os passos para a tomada de decisão são (1) identificar a existência de um problema, (2) listar as possíveis alternativas para a solução do problema, (3) selecionar dentre estas alternativas a que produz os melhores resultados, (4) colocar a alternativa selecionada em ação, e (5) coletar o '*feedback*' para avaliar se a alternativa implementada está solucionando o problema identificado (CERTO, 1994).

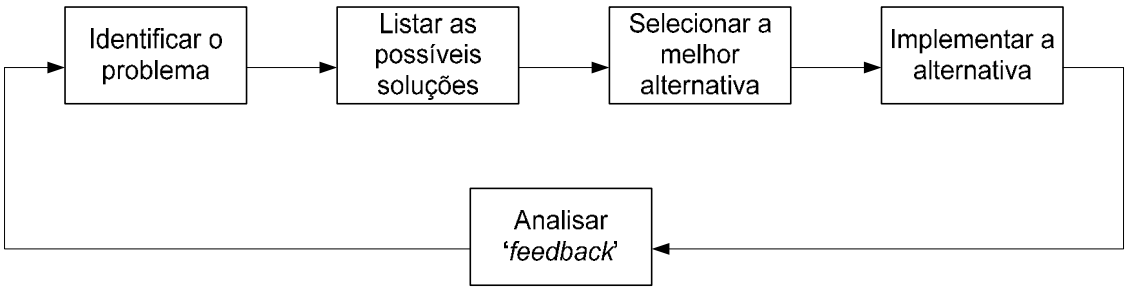

Figura 1 - Processo de tomada de decisão

O modelo de processo de tomada de decisão apresentado é baseado em três premissas básicas. Primeiro, o modelo assume que os seres humanos são seres econômicos com o objetivo de maximizar a satisfação ou o retorno. Segundo, ele assume que dentro da situação de tomada de decisão todas as alternativas e suas possíveis conseqüências são conhecidas. A última premissa é que os decisores tenham algum sistema de prioridade claro, bem definido, que os permita classificar a utilidade de cada alternativa. No caso de todas estas premissas serem atendidas na situação de tomada de decisão, os decisores provavelmente tomarão a melhor decisão possível para a organização. Nas situações do mundo real, contudo, nem sempre todas estas premissas são atendidas, e decisões relativas são tomadas, sendo normalmente menos eficientes, ou mesmo eficazes, que a melhor decisão possível para a organização (CERTO, 1994), (SHIMIZU, 2001).

2.3 CONDIÇÕES DE TOMADA DE DECISÃO

Na maioria dos casos no mundo real, é praticamente impossível aos decisores preverem exatamente quais as conseqüências da implementação de uma alternativa. A palavra futuro é a chave na discussão das condições de tomada de decisão. Para propósitos práticos, devido à constante mudança das organizações e do ambiente em que elas se encontram, as futuras conseqüências das decisões implementadas não são perfeitamente previsíveis (CERTO, 1994), (PIDD 2001).

Em geral, pode-se classificar as condições sob as quais a decisão deve ser tomada em três categorias. A divisão destas categorias é baseada no grau de previsibilidade das conseqüências das alternativas da decisão. Estas condições são a certeza completa, a incerteza completa, e o risco (CERTO, 1994).

2.3.1 Condição de certeza completa

A situação de certeza completa existe quando os decisores sabem exatamente quais serão os resultados da implementação de determinada alternativa. Nesta condição, eles têm conhecimento completo a respeito da decisão. Tudo o que os decisores têm a fazer é listar os resultados das alternativas e então escolher a alternativa cujo resultado maximiza os ganhos para a organização. Por exemplo, o resultado de uma alternativa de investimento baseado na aquisição de títulos do governo é, para todos os propósitos práticos previsível, visto que as taxas de retorno são pré-estabelecidas pelo governo. Decidir implementar esta alternativa seria essencialmente tomar uma decisão em uma situação de certeza completa. A maioria das decisões organizacionais, no entanto, é tomada fora de uma condição de certeza completa (CERTO, 1994).

2.3.2 Condição de incerteza completa

A situação de incerteza completa existe quando decisores não têm absolutamente nenhuma idéia de quais serão os resultados da implementação de determinada alternativa. A situação de incerteza completa existe, por exemplo, se não houver nenhum histórico sobre o qual a decisão possa ser baseada. Não conhecer o que aconteceu no passado torna muito difícil a previsão do que acontecerá no futuro. Nestas situações, os decisores normalmente assumem que decisões corretas são meramente uma questão de sorte. Um exemplo de tomada de decisão em uma situação de incerteza completa seria a escolha de puxar a alavanca de uma máquina de doces rotulada “surpresa do dia” ao invés de puxar a alavanca que disponibilizaria uma barra de chocolate de uma marca conhecida. Felizmente muito poucas decisões organizacionais precisam ser tomadas em uma ambiente de incerteza completa (CERTO, 1994).

2.3.3 Condição de risco

A principal característica de uma condição de risco é que os decisores têm informação apenas o suficiente para estimar a probabilidade da ocorrência do resultado se determinada alternativa for implementada. A condição de risco está classificada entre a certeza completa e a incerteza completa. Como exemplo, suponha um gerente que decide contratar mais dois vendedores com o intuito de aumentar as vendas da organização está decidindo em uma situação de risco. Ele pode acreditar que é alta a probabilidade que estes dois vendedores adicionais irão aumentar o total de vendas, mas não há garantias que isso ocorra. Algum risco é associado a esta decisão (CERTO, 1994).

No mundo real, graus de risco podem ser associados às decisões tomadas em situações de risco. Quanto menor for a qualidade da informação[2] disponível aos possíveis resultados da escolha de uma alternativa, mais perto da incerteza completa será a situação, e maior será o risco se aquela alternativa for escolhida. A maioria das decisões tomadas nas organizações tem algum grau de risco associado (CERTO, 1994).

2.4 NECESSIDADE DO ESTUDO DE TOMADA DE DECISÃO

Com a evolução tecnológica experimentada pela humanidade, evoluíram também as necessidades de tomada de decisão. O crescente avanço nas áreas de pesquisa proporciona a disponibilidade de diversas alternativas nas soluções de problemas. Adicionando a este fator o aumento da complexidade do ambiente em que a humanidade se encontra, de tal forma que o caos é tópico de diversas linhas de pesquisa, as decisões em condição de risco são cada vez mais freqüentes (RUSSEL; NORVIG, 1995).

O mundo globalizado com todas as suas conseqüências, como o aumento da competitividade, também influencia o processo de decisão, no tocante ao aumento

[2] Neste contexto, qualidade da informação pode ser interpretada como a pertinência, a classificação e a estrutura da informação disponibilizada ao decisor.

da qualidade necessária ao processo de decisão. Uma tomada de decisão executada não tão bem quanto poderia ser pode significar a ruína de uma organização (CERTO, 1994), (PIDD 2001).

Além destes fatores, ainda há outros dois fatores importantes a serem considerados na emergência do estudo de tomada de decisão: a disponibilidade de computação eletrônica de alto desempenho e a crescente necessidade de formalismo no processo decisório (HERNANDEZ, 1992).

Apesar do estado de elaboração de seus fundamentos lógicos, os estudos de tomada de decisão seriam meramente curiosidade intelectual e não uma poderosa ferramenta, se os meios para construir modelos e manipulá-los de maneira economicamente viável não estivessem disponíveis. O rápido desenvolvimento de computadores eletrônicos associado ao desenvolvimento de sistemas de software nas últimas décadas tornou possível o que seria impossível há apenas um quarto de século atrás. A disponibilidade da computação eletrônica é uma condição essencial para o crescimento do campo de tomada de decisão (HERNANDEZ, 1992).

Uma forte tendência no corrente desenvolvimento dos estudos da tomada de decisão é a propensão em direção à administração profissional nas organizações atuais. O '*one-show-man*' está dando passagem a comitês e a conselhos, e o empreendedor individual está se tornando relativamente menos importante. Um fato concomitante a esta mudança é a necessidade por parte dos novos administradores profissionais de apresentar evidências de decisões mais cuidadosamente refletidas e documentadas. Até mesmo o bom agente decisor intuitivo terá de convencer os outros quanto à lógica de suas decisões. Cada vez mais, bons resultados provenientes de decisões intuitivas serão vistos da mesma forma que ganhos em corridas – isto é, como resultados de sorte, e não como uma prática administrativa prudente (HERNANDEZ, 1992).

2.5 UTILIZAÇÃO DE MODELOS PARA SIMULAÇÃO

Em razão do alto custo, da inviabilidade de implantação devido a restrições de tempo ou quaisquer outros fatores que não permitam a livre manipulação, a

utilização de modelos para a simulação do comportamento da situação real mostra-se interessante. Além destes fatores, há a possibilidade de alteração das variáveis pertinentes ao sistema a fim de observar as suas possíveis conseqüências no futuro. Aliada ao poder da computação eletrônica de alto desempenho (e as facilidades a ela associadas) a utilização de modelos apresenta-se como uma maneira muito interessante para a execução dos estudos necessários para a tomada de decisão de alta qualidade (SHIMIZU, 2001), (LAUDON, K; LAUDON, J, 2004).

Por definição, um modelo descreve, representa ou imita o procedimento que ocorre no mundo real, estabelecendo o relacionamento das variáveis com os objetivos, da melhor maneira possível, obedecendo a limitação de tempo e de custo. Segundo (SHIMIZU, 2001), os modelos podem ser de vários tipos, entre os quais:

- Verbais: quando descritos e representados por palavras e sentenças (exemplos: questionários, sistemas especialistas, etc.);
- Físicos: quando representados por algum tipo de material ou hardware, alternando-se suas dimensões, formato e custo (exemplos: maquete, protótipos);
- Esquemáticos: quando representados por meio de gráficos, tabelas, diagramas ou árvores de decisão;
- Matemáticos: quando representados por equações e valores numéricos ou valores da lógica simbólica (exemplos: programação linear, rede neural, etc.).

No presente trabalho serão utilizados, especificamente, modelos que podem ser classificados como matemáticos. Um modelo matemático recebe as entradas, representadas pelos parâmetros, variáveis de entrada e decisões, e as processa para produzir as saídas, neste caso as variáveis de saída ou resultados da decisão. Havendo a necessidade de adição de outro fator ou característica ao modelo, novas variáveis nas equações existentes, ou novas equações podem ser incorporadas.

A escolha do modelo para a representação da situação de decisão depende da finalidade da decisão, da limitação do tempo e custo, e da complexidade do problema. Um problema pode ser considerado como complexo quando:

- O número de variáveis e/ou dos objetivos aumenta: são os problemas multidimensionais com múltiplos objetivos;
- A ocorrência dos valores das variáveis e/ou dos objetivos está sujeita a riscos ou incertezas; e
- Os valores das variáveis e os objetivos são definidos de modo impreciso, nebuloso ou difuso (*fuzzy*).

Durante os vários anos de pesquisa na área de tomada de decisão, muitos modelos e métodos foram desenvolvidos, cada um dos quais utilizado com maior eficiência em determinada área do conhecimento humano. Alguns dos mais utilizados são a árvore de decisão, os Sistemas de Apoio à Decisão, as tabelas de decisão, a média ponderada, utilidade Min/Max, teoria das filas, cadeias de Markov, programação linear com múltiplos objetivos, heurísticas, *simulated annealing*, algoritmo genético, teoria dos jogos, e a programação quadrática, entre diversos outros (SHIMIZU, 2001), (LAUDON, K; LAUDON, J, 2004).

Recentemente, com o avanço das pesquisas em inteligência artificial, muitas das técnicas utilizadas nesta área de pesquisa vêm sendo empregadas com sucesso na modelagem de modelos e métodos de decisão. Entre as principais técnicas utilizadas, pode-se citar os sistemas especialistas, as redes neurais artificiais, em suas diversas variantes, a lógica difusa (*fuzzy*), os algoritmos genéticos, e os agentes inteligentes (SHIMIZU, 2001), (LAUDON, K; LAUDON, J, 2004).

2.6 O USO DE INTELIGÊNCIA ARTIFICIAL NA TOMADA DE DECISÃO

O conceito de Inteligência Artificial pode ser definido de diversas formas, dependendo da área de conhecimento e do ponto de vista adotado. Em uma definição genérica e simplificada, é uma área de pesquisa multidisciplinar que investiga diferentes formas de habilitar o computador a realizar tarefas que, até o momento, o ser humano apresenta desempenho superior (MITCHELL, 1997), (RUSSEL; NORVIG, 1995).

Devido à natureza desta área de pesquisa, especialmente em querer superar o desempenho humano em determinadas áreas, ela emergiu com um forte apelo multidisciplinar. Muitos conceitos utilizados e aprimorados na Inteligência Artificial tiveram origem em outras áreas do conhecimento humano, tais como a biologia, a psicologia, a filosofia e a lingüística, além da engenharia, ciência da computação e da lógica matemática (RUSSEL; NORVIG, 1995).

Com o aumento da complexidade do ambiente empresarial e da velocidade com que as tomadas de decisão têm de ser executadas, é natural empregar quaisquer classes de técnicas, incluindo a inteligência artificial, visando a melhoria do processo decisório. As empresas mostram-se cada vez mais interessadas na aplicação das técnicas de inteligência artificial por diversos motivos. Alguns dos mais interessantes são certamente (LAUDON, K; LAUDON, J, 2004):

- No armazenamento de informações de maneira ativa sob a forma de memória organizacional, criando desta forma uma base de conhecimento organizacional que muitos funcionários possam examinar e preservar a experiência técnica que pode ser perdida quando os peritos das áreas de conhecimento deixam a empresa.
- Na criação de um mecanismo que não esteja sujeito a características humanas, como a fadiga e a preocupação. Isso é especialmente útil quando as tarefas são ambientais, físicas ou mentalmente perigosas aos seres humanos. Alguns destes sistemas também podem ser utilizados como ‘conselheiros’, muito úteis em tempos de crise.
- No processo de eliminação de tarefas rotineiras executadas por pessoas.
- No aprimoramento da base de conhecimento da organização, gerando soluções para problemas específicos que são muito complexos para análise humana em curto espaço de tempo. Este tipo de problemas normalmente envolve um número muito elevado de variáveis e necessita do cruzamento de informações aparentemente não-relacionadas. Algumas técnicas de inteligência artificial auxiliam na execução deste tipo de tarefa.

2.7 SISTEMAS ESPECIALISTAS

Uma classe de sistemas de informação[3] que contenha dados ou sentenças descritivas sobre determinado ramo da atividade humana, como diagnóstico médico, previsão de tempo, previsão financeira, projeto de máquinas, consultas sobre geografia, arqueologia, turismo, língua, etc., representa os denominados sistemas especialistas.

As informações existentes no sistema especialista foram coletadas e organizadas sob a supervisão de um grupo de especialistas do assunto, e o sistema responde de modo natural a qualquer dúvida do usuário, como se ele estivesse consultando um profissional do ramo (LAUDON, K; LAUDON, J, 2004), (RUSSEL; NORVIG, 1995).

Normalmente, as informações são armazenadas em forma de sentenças denominadas regras de produção, do tipo:

IF (fatos forem verdadeiros) THEN (executar algoritmo a)
ELSE (executar algoritmo b)

Formando uma base de dados de conhecimento. Essas sentenças são selecionadas para resolver um problema por meio de algoritmos que formam um mecanismo de inferência. Sistemas especialistas conseguem também incorporar novas informações e, por meio do cruzamento e das combinações das informações existentes, responder a perguntas ou analisar situações cada vez mais complexas. Essa capacidade do sistema especialista é denominada aprendizagem e auto-organização e serve como indicador da existência de certo tipo de inteligência (SHIMIZU, 2001).

Desta forma conclui-se o presente capítulo, que teve por objetivo a introdução do conceito da tomada de decisão, especificamente nas organizações modernas. Ao longo do capítulo foram brevemente apresentadas as condições das tomadas de decisão, a utilização de modelos como auxílio no processo de tomada de decisão, e

[3] Um sistema de informação pode ser representado por arquivos ou banco de dados administrados por um sistema de software, que seja responsável pela manutenção das informações no mesmo.

a utilização de técnicas de inteligência artificial como um dos possíveis modelos matemáticos utilizados na simulação, além de outros tópicos pertinentes.

Dando seqüência à primeira parte do presente trabalho, responsável pela introdução dos conceitos teóricos utilizados na proposta apresentada, o próximo capítulo será responsável pela apresentação do modelo matemático utilizado, algumas de suas características e outras informações pertinentes ao entendimento e desenvolvimento do mesmo.

3 TECNOLOGIA ADAPTATIVA

Como ilustrado ao final do capítulo anterior, a utilização de técnicas de Inteligência Artificial no processo decisório das empresas vem sendo cada vez mais disseminada. Diversos sistemas de informação comerciais modernos incorporam, além das funcionalidades de tomada de decisão ditas ‘clássicas’ também técnicas de Inteligência Artificial. Isso ilustra a necessidade do estudo desta categoria de técnicas de tomada de decisão no presente trabalho.

Este capítulo apresenta a tecnologia para o auxílio de tomada de decisão utilizada na proposta do presente trabalho, a denominada tecnologia adaptativa, sua origem, seus conceitos básicos e algumas aplicações de seu uso em problemas computacionais.

A organização deste capítulo se dá da seguinte forma: uma seção para apresentar um breve histórico da tecnologia adaptativa, com a sua origem e uma breve ilustração da sua evolução, outra seção para ilustrar a generalização da tecnologia adaptativa, e ainda outra seção para apresentar o dispositivo específico utilizado na proposta do presente trabalho.

Após o breve histórico apresentado na primeira seção, segue a generalização do conceito de dispositivos dirigidos por regras, a generalização da tecnologia adaptativa a partir dos dispositivos dirigidos por regras e pela apresentação do modelo específico utilizado na proposta do presente trabalho.

Ao longo do capítulo são apresentados exemplos da utilização da tecnologia adaptativa de forma a ilustrar melhor os conceitos técnicos apresentados no trabalho.

3.1 ORIGEM

A crescente disseminação de novos e diferentes sistemas acarretou o surgimento de ambientes heterogêneos, gerando a necessidade de integração entre eles, portanto da tradução da linguagem de um sistema para outro. Com o também

aumento da qualidade das interfaces homem-máquina, com linguagens cada vez mais próximas da linguagem natural, e as cada vez mais necessárias qualidade e confiabilidade de software, a necessidade de dispositivos que atendessem essas premissas surgiu.

A intensificação do trabalho e pesquisa sobre dispositivos formais, especificamente para o tratamento de linguagens de programação, desenvolvimento de compiladores e modelos de computação ocasionou o desenvolvimento de uma nova classe de dispositivos que atendesse a essas recém-criadas necessidades.

O primeiro dispositivo formal com algumas características que atendem estas novas necessidades foi proposto em Neto e Magalhães (1981), e denomina-se Autômato de Pilha Estruturado. Este dispositivo apresenta uma opção mais didática e de manipulação mais intuitiva em comparação ao autômato de pilha convencional.

Nesta formalização, o Autômato de Pilha Estruturado é uma máquina de estados composta por um conjunto de submáquinas com transições que consomem símbolos de entrada ou fazem chamada ou retorno de outra submáquina. Estas submáquinas são na verdade autômatos finitos, diferenciando-se dos mesmos apenas pelas transições de chamada e retorno de submáquinas. Uma chamada de submáquina implica em guardar o próximo estado na pilha e continuar o reconhecimento da cadeia a partir do estado inicial da submáquina em questão.

Ao término do processamento desta cadeia e sendo o estado corrente um estado final, a análise da cadeia continuará, então, a partir do estado armazenado no topo da pilha, o qual será desempilhado. Este procedimento é chamado de transição de retorno de submáquina. No Autômato de Pilha Estruturado a pilha é utilizada somente para o controle do aninhamento sintático, que é a característica que diferencia uma construção livre de contexto de uma construção regular (NETO; MAGALHÃES, 1981). Desta forma, a cada nova instância de aninhamento, uma chamada à submáquina será efetuada.

Portanto, aumenta-se o poder computacional do dispositivo sem alterar de forma significativa o seu funcionamento, visto que se pode considerar o Autômato de Pilha Estruturado como um conjunto de autômatos finitos coordenados através do uso de uma pilha implícita.

Foi apresentado em Neto e Magalhães (1981) um estudo comparativo que indica a equivalência entre o Autômato de Pilha Estruturado e o Autômato de Pilha convencional. No entanto, a vantagem didática apresentada pelo Autômato de Pilha

Estruturado devido ao principio de progressão suave de recursos oferecidos proporciona uma excelente opção de ensino, além de apresentar-se como uma técnica muito interessante, que permite a manipulação da pilha de maneira sub-ótima, exibindo, entretanto, excelente desempenho. A formalização completa do Autômato de Pilha Estruturado pode ser encontrada em Neto (1993).

O dispositivo que iniciou o desenvolvimento de uma série de dispositivos adaptativos, e posteriormente o conceito generalizado da tecnologia adaptativa, foi o Autômato Adaptativo. O seu funcionamento parte da premissa de aumentar o poder computacional de um dispositivo sem alterar de forma significativa o seu funcionamento, provendo, desta maneira, uma curva de aprendizado muito mais suave. A proposta do Autômato Adaptativo é prover uma extensão ao Autômato de Pilha Estruturado, de tal forma que ele possa executar transições que modifiquem a topologia do dispositivo, alterando o seu comportamento, dotando-o de comportamento adaptativo (que se adapta) em relação aos estímulos de entrada.

Estas características lhe conferem a possibilidade de, a cada nova transição, poder alterar a sua própria estrutura, ou seja, adicionar ou remover estados e transições, por meio de chamadas às funções adaptativas. Isso lhe permite memorizar em sua própria estrutura as construções necessárias para o tratamento das dependências de contexto.

As funções adaptativas são declaradas basicamente em termos de ações adaptativas elementares, que são a ação de pesquisa (representada pelo símbolo ?), a ação de inclusão (representada pelo símbolo +) e a ação de exclusão (representada pelo símbolo -), além de uma estrutura de suporte, como declaração de variáveis, geradores e parâmetros, entre outros recursos. Maiores informações, incluindo a completa formalização do conceito do autômato adaptativo podem ser encontradas em Neto (1994) e Neto (1993).

Com sua capacidade de auto-modificação, o Autômato Adaptativo tem poder computacional mais elevado do que o Autômato de Pilha Estruturado. Tendo este último a capacidade de reconhecimento de linguagens livres de contexto, o Autômato Adaptativo é capaz de aceitar linguagens recursivamente enumeráveis. Esta categoria de linguagens é tratada normalmente pelo dispositivo conhecido como Máquina de Turing com fita de entrada infinita (LEWIS; PAPADIMITRIOU, 1998). A demonstração da equivalência de expressividade computacional entre a

Máquina de Turing e o Autômato Adaptativo foi apresentada em Rocha e Neto (2000).

As vantagens didáticas da utilização dos dispositivos Autômato de Pilha Estruturado e Autômato Adaptativo foram apresentadas em Pedrazzi; Matsuno e Rocha (2004). Este trabalho apresenta uma proposta de ensino da disciplina de linguagens formais e autômatos utilizando o conceito de adição de recursos aos dispositivos de forma a incrementar o seu poder computacional. Partindo da utilização do Autômato Finito para o reconhecimento de linguagens regulares; do Autômato de Pilha Estruturado para o reconhecimento de linguagens livres de contexto e da utilização do Autômato Adaptativo para o reconhecimento de linguagens sensíveis a contexto e tratamento das linguagens recursivamente enumeráveis, obteve-se uma curva de aprendizado mais suave em comparação ao método de ensino comumente empregado.

Do ponto de vista tecnológico, a utilização do Autômato Adaptativo na solução de problemas computacionais obteve muito sucesso, como pode ser observado em Pistori (2003), onde é apresentado um compêndio dos dispositivos adaptativos propostos, juntamente com uma linha do tempo situando o desenvolvimento da tecnologia adaptativa como um todo. Especial atenção deve ser prestada às soluções propostas a alguns problemas relacionados a linguagens formais, fundamentos de linguagens de programação e teoria de compiladores. Em Neto (1993) foram apresentadas algumas aplicações interessantes do Autômato Adaptativo nestas áreas específicas.

A partir do desenvolvimento do Autômato Adaptativo, seguiu-se a construção de outros dispositivos adaptativos. O estudo do comportamento, projeto e aplicações desta classe de dispositivos culminou na criação de uma nova linha de pesquisa, a tecnologia adaptativa (PISTORI, 2003).

3.2 DISPOSITIVOS DIRIGIDOS POR REGRAS

A presente e a próxima seção do presente trabalho, em tempo seções 3.2 e 3.3, foram baseadas em Neto (2001).

Qualquer dispositivo cujo comportamento dependa exclusivamente de um conjunto fixo e finito de regras é um dispositivo dirigido por regras. Este conjunto de regras deve mapear todas as possíveis configurações juntamente com as possíveis entradas do dispositivo em uma próxima configuração.

A definição apresentada é extremamente genérica, levando à conclusão que praticamente todos os dispositivos formais baseados em eventos discretos e não-dinâmicos, ou seja, que não tenham o seu comportamento alterado durante o processamento, seja classificado como um dispositivo dirigido por regras.

O funcionamento destes dispositivos é relativamente simples. Inicia-se o seu funcionamento em uma configuração pré-determinada e executa-se a transição de uma configuração à outra de acordo com a sua cadeia de estímulos de entrada, até o final da mesma. O processo de transição de configurações avalia apenas a configuração atual do dispositivo e o seu estímulo de entrada, não havendo a avaliação do histórico de operações executadas.

Ao final da cadeia de estímulos de entrada, de acordo com a configuração final do dispositivo, determina-se se a cadeia foi aceita ou rejeitada. Ainda, como resultado do processamento da cadeia de estímulos de entrada, uma cadeia de estímulos de saída pode ser gerada.

Este tipo de dispositivo assume uma determinada configuração pelo acionamento de uma regra de seu conjunto de regras. Cada regra mapeia o dispositivo em uma configuração.

Pode-se formalizar o conceito de dispositivo dirigido por regras através de uma héxupla DR, da forma DR = (C, R, EE, c_0, CA, ES), os quais:

- DR é um dispositivo dirigido por regras;
- C é o conjunto de todas as possíveis configurações do dispositivo DR;
- c_0 é a configuração inicial do dispositivo DR;
- CA é o conjunto de todas as configurações de aceite do dispositivo DR;
- EE é o conjunto de todos os estímulos de entrada possíveis ao dispositivo DR, incluindo o possível estímulo vazio (ε);
- ES é o conjunto de todos os estímulos de saída possíveis ao dispositivo DR, incluindo o possível estímulo vazio (ε);

- R é o conjunto de regras que determinam o comportamento do dispositivo DR, e é definido pela relação R $\subseteq$ C×EE×C×ES.

Cada uma das regras do conjunto R segue a forma r_k = (c_i,e,c_j,s), especificando que, em resposta a qualquer estímulo de entrada $e \in$ EE, r_k altera a configuração atual do dispositivo de c_i para c_j, consome o estímulo de entrada e, e gera como saída o estímulo de saída $s \in$ ES.

Uma determinada regra r_k, da forma acima exposta, é dita compatível com a configuração atual c do dispositivo se e somente se $c_i \equiv c$, com e representando o estímulo vazio ou o próximo estímulo de entrada a ser processado. Neste caso, a aplicação de uma única regra compatível transiciona o dispositivo para configuração c_j, consome o estimulo de entrada e adiciona s à cadeia de estímulos de saída do dispositivo, recordando que e, s ou ambos podem ser vazios.

Portanto, uma cadeia de estímulos de entrada é dita aceita quando, após o completo processamento da mesma, o dispositivo dirigido por regras encontrar-se em uma configuração de aceite c_m, com $c_m \in$ CA. De forma análoga, a cadeia de estímulos de entrada é dita rejeitada se $c_m \notin$ CA, ou ainda, $c_m \in$ C-CA. Uma formalização com maior grau de detalhamento dos dispositivos dirigidos por regra pode ser encontrada em Neto (2001).

Como exemplos de dispositivos dirigidos por regras, pode-se citar o Autômato Finito, os Autômatos de Pilha clássico e Estruturado, a Máquina de Turing, as Gramáticas em suas diversas formas; sendo estes todos exemplos de formalismos utilizados nos estudos de linguagens formais e autômatos. Porém, diversos outros dispositivos podem ser classificados como dispositivos dirigidos por regras, como os statecharts (HAREL, 1987a) e (HAREL, 1987b), as tabelas de decisão (MONTALBANO, 1974) e as árvores de decisão (MITCHELL, 1997), dispositivos estes amplamente utilizados em áreas diversas do conhecimento humano, como a administração, a economia e a matemática, entre outras.

3.3 PROPOSTA GENERALIZADA DA ADAPTATIVIDADE

A formalização do conceito do dispositivo dirigido por regras foi muito importante para o desenvolvimento da tecnologia adaptativa, pois possibilitou a generalização do processo de construção dos dispositivos adaptativos.

Anteriormente a esta formalização, o processo de definição de um dispositivo adaptativo era originado por um extensivo estudo do dispositivo subjacente que seria estendido por um dispositivo adaptativo. Observa-se que este processo era praticamente artesanal.

Com o embasamento da formalização dos dispositivos dirigidos por regras, a separação do dispositivo subjacente da sua extensão adaptativa, bem como os meios de integração destas partes para a composição do dispositivo adaptativo, tornou-se mais simples facilitando o processo de evolução da tecnologia adaptativa do estado de produção artesanal, executada caso a caso, para propriamente uma linha de pesquisa da área da engenharia da computação.

Dispositivos adaptativos são dispositivos cujo comportamento é alterado dinamicamente, em resposta a estímulos de entrada, sem interferências externas além destes estímulos. Para adquirir tal capacidade, um dispositivo adaptativo deve ser dotado de algum recurso capaz de alterar a sua estrutura, de modo a alterar o seu comportamento.

Partindo do conceito dos dispositivos dirigidos por regras, definido na seção 3.2 deste capítulo, um dispositivo adaptativo é qualquer dispositivo que, através de uma extensão, pode ter o seu conjunto de regras alterado dinamicamente durante o seu funcionamento.

Para se obter este comportamento dinâmico, a extensão adaptativa de um dispositivo dirigido por regras deve ser projetada de forma a prover duas funcionalidades primordiais:

- Um conjunto de funções adaptativas que alterem as regras do dispositivo dirigido por regras, dito subjacente;

- A alteração das regras do dispositivo subjacente, de modo a incluir chamadas as funções adaptativas definidas, as denominadas ações adaptativas.

As funções adaptativas são declaradas basicamente em termos de ações adaptativas elementares, que são a ação de pesquisa (representada pelo símbolo ?), a ação de inclusão (representada pelo símbolo +) e a ação de exclusão (representada pelo símbolo -), além de uma estrutura de suporte, como declaração de variáveis, geradores e parâmetros, dentre outros recursos. Esta definição é praticamente a mesma da definição das funções adaptativas do Autômato Adaptativo. A diferença básica é que, ao invés das funções adaptativas elementares tratarem de pesquisa, inclusão e exclusão de estados e transações, elas operam sobre as regras do conjunto de regras do dispositivo subjacente.

O conceito de um dispositivo adaptativo DA pode ser formalizado por uma tripla da forma DA = (DR_0, EA, FA), os quais:

- DA é um dispositivo adaptativo;
- DR_0 é um dispositivo dirigido por regras inicial, dito subjacente ao dispositivo adaptativo;
- FA é um conjunto de funções adaptativas, incluindo a função vazia (ε);
- EA é a extensão adaptativa, definida pela relação EA $\subseteq$ FA$\times$R$\times$FA, com R sendo o conjunto de regras do dispositivo subjacente.

Observa-se que a extensão adaptativa associa a cada regra do dispositivo subjacente duas ações adaptativas. Uma ação é denominada ação anterior e a outra ação posterior. Isto se deve ao momento da execução da ação em relação à aplicação da regra a qual ela foi associada. A ação anterior é executada antes da mudança de configuração do dispositivo pela regra, e a ação posterior após.

Determinada regra A_k estendida pela extensão adaptativa, deve seguir a forma A_k = (AA, C_i, S, C_j, Z, AP), com AA e AP pertencendo a FA. C_i indica a configuração atual do dispositivo, isto é, a situação corrente. C_j indica a configuração futura, a situação que o dispositivo assumirá após a aplicação da regra, com S sendo o estímulo de entrada do dispositivo e Z o estímulo de saída.

Sempre que uma ação adaptativa não-nula for executada, o dispositivo subjacente DR_n será alterado para DR_{n+1}, representando uma linha de alteração das regras do dispositivo por meio da execução de funções adaptativas. Portanto, sendo o dispositivo dirigido por regras inicial DR_0, após a execução de uma ação adaptativa não-nula, o dispositivo subjacente seria alterado para DR_1, representando que o conjunto de regras do mesmo foi alterado, e assim sucessivamente até o fim do processamento do dispositivo. Esta proposta generalizada do conceito de dispositivos adaptativos, com maiores detalhes da formalização, é apresentada com maiores detalhes em Neto (2001).

A extensão de qualquer dispositivo pelo conceito da adaptatividade aumenta o seu poder computacional, dentro dos limites conhecidos da computabilidade. A Tese de Church propõe que o limite superior de poder computacional é o poder da Máquina de Turing com fita ilimitada aos dois lados (HOPCROFT; MOTWANI; ULLMAN, 2001). No entanto, pode-se simular uma Máquina de Turing com fita ilimitada em uma Máquina de Turing com fita limitada à direita e ilimitada a esquerda[4].

Exemplificando esta extensão, pode-se comparar o poder computacional de um Autômato Finito clássico e de um Autômato Finito Adaptativo.

Sabe-se que um Autômato Finito clássico é capaz de reconhecer apenas linguagens classificadas como linguagens regulares, ou linguagens do tipo 3 de acordo com a hierarquia de Chomsky (LEWIS; PAPADIMITRIOU, 1998). A extensão do Autômato Finito pela tecnologia adaptativa, fato que permite ao dispositivo a adição, consulta e remoção de estados e transições, capacita-o a reconhecer linguagens recursivamente enumeráveis, ou linguagens do tipo 0, ainda de acordo com a hierarquia de Chomsky. Somente um dispositivo com o poder computacional da Máquina de Turing com fita de entrada ilimitada seria capaz de reconhecer tal classe de linguagens (HOPCROFT; MOTWANI; ULLMAN, 2001).

Como formalizado em Rocha e Neto (2000), o autômato adaptativo é Turing-compatível, ou seja, possui o mesmo poder computacional da Máquina de Turing. Vale lembrar que a Máquina de Turing é considerado o modelo formal subjacente da

[4] A fita infinita à esquerda e à direita pode ser facilmente simulada por uma fita tradicional dividindo-a ao meio com um marcador, com as células pares representando a parte direita da fita e as ímpares a parte esquerda.

arquitetura de Von Neumann, e conseqüentemente, o modelo de computação mais amplamente utilizado nos computadores digitais modernos.

Em Pistori, (2003) pode-se encontrar um estudo interessante sobre os dispositivos adaptativos já desenvolvidos e as suas características básicas. Ainda em Pistori, (2003), uma linha do tempo apresentando os avanços científicos e os avanços tecnológicos resume os principais eventos ocorridos durante o desenvolvimento da tecnologia adaptativa, com as contribuições teóricas e a apresentação de algumas ferramentas, aplicações e protótipos.

3.4 MODELO UTILIZADO – TABELAS DE DECISÃO ADAPTATIVAS

Esta seção tem por objetivo apresentar o dispositivo adaptativo utilizado para a implementação da proposta do presente trabalho. Será apresentado o conceito do dispositivo, o seu funcionamento básico e a sua estrutura, um algoritmo em alto nível do funcionamento detalhado do dispositivo, juntamente com um exemplo didático. Ao final da seção, um breve comparativo entre as Tabelas de Decisão Adaptativas e alguns outros dispositivos adaptativos é apresentado.

Tabelas de decisão são ferramentas muito populares entre programadores de sistemas de informação e engenheiros de software, além de apresentar um razoável nível de aceitação em diversas outras áreas do conhecimento humano, principalmente nas áreas de administração de empresas e engenharia de produção. As Tabelas de Decisão Adaptativas (NETO, 2001) constituem uma aplicação interessante dos conceitos de dispositivos adaptativos à área de sistemas de informação. Esta classe de dispositivos pode ser concebida como a classe de dispositivos adaptativos que utiliza a tabela de decisão convencional como dispositivo subjacente.

Uma tabela de decisão típica é constituída por um conjunto de regras. Cada uma destas regras possui duas partes: uma parte composta pelas condições a serem testadas e uma parte composta pelas ações a serem executadas, caso o teste de todas as condições seja verdadeiro (MONTALBANO, 1974).

Os campos das regras podem conter três valores distintos: Verdadeiro (V), Falso (F) ou Nulo (-). No processo de teste das condições das regras, as condições cujo campo estiver preenchido com o valor Nulo não serão testadas, e, portanto, tornam-se irrelevantes na avaliação da regra. Já nos campos das ações a serem executadas da regra, apenas os valores Verdadeiro (V) e Falso (F) são possíveis.

Pode-se representar uma tabela de decisão convencional de tal forma que as regras sejam representadas por colunas, onde cada coluna corresponde a uma regra independente; e as condições a serem testadas e as ações a serem executadas sejam representadas pelas linhas da tabela. A seguir tem-se um exemplo de uma tabela de decisão típica.

material=	128	128	128	903	903	32
obra=	15	18	18	15	18	33
fornecedor=	589	589	101	589	101	101
custo<limite_custo=	S	S	N	S	S	N
proximidade=	S	N	N	S	S	S
imprimir (regra)	✓			✓	✓	

Figura 2 - Exemplo de tabela de decisão

A operação desta classe de dispositivo é extremamente intuitiva:

- Primeiramente, executa-se uma busca nas condições de todas as regras da tabela de decisão, verificando se há alguma regra compatível com a situação atual do ambiente.
- Não havendo nenhuma regra compatível, nenhuma ação é executada.
- Havendo apenas uma regra compatível, esta regra é selecionada para a aplicação.
- Havendo mais de uma regra compatível, obtém-se o que se denomina de configuração não-determinística. Neste caso, selecionam-se todas as regras compatíveis para a aplicação.
- A regra selecionada para aplicação é acionada, pela execução de todas as ações cujo campo da regra foi solicitado. Havendo mais de

uma regra selecionada para aplicação, todas as regras são executadas em paralelo.

- Uma vez executadas todas as ações solicitadas correspondentes à regra selecionada, o processo de operação da tabela de decisão é reiniciado.

Como pode ser observado em Nagayama (1990), existem algumas propriedades que classificam as tabelas de decisão em categorias. Alguns exemplos destas propriedades são a existência ou não de regra do tipo '*else*', que seria executada caso nenhuma das regras ditas 'convencionais' da tabelas de decisão seja plausível de aplicação; ou então a possibilidade de uso de valores ou ainda de variáveis nos campos de condição das regras. Extrapolando um pouco mais o modelo, algumas tabelas de decisão permitem até mesmo o uso de conjuntos valorados nos campos de condição das regras.

A disponibilidade deste tipo de recursos torna ainda mais expressiva a tabela de decisão. Diversos outros recursos possíveis de utilização em uma tabela de decisão podem ser encontrados em Nagayama (1990).

A Tabela de Decisão Adaptativa é o dispositivo adaptativo que utiliza como dispositivo subjacente a tabela de decisão. Desta forma, a estrutura de uma tabela de decisão adaptativa é similar à estrutura de uma tabela de decisão convencional. Associando-se ações adaptativas às regras da tabela de decisão e havendo a incorporação de um conjunto de funções adaptativas à mesma, obtêm-se o dispositivo desejado (NETO, 2001).

Pode-se observar um possível modelo de tabela de decisão adaptativa na figura a seguir:

		Tag →	H	-	-	-	+	S	R	R	R	R	R	R	R	E
Tabela de Decisão Subjacente	Condições	material=		p1	p1	p1	p1		"µ"	128	128	128	903	903	32	
		obra=		p2	p2	p2	p2		"o"	15	18	18	15	18	33	
		fornecedor=					g1			589	589	101	589	101	101	
		custo<limite_custo=		N	N	S	S			S	S	N	S	S	N	
		proximidade=		S	N	N	S			S	N	N	S	S	S	
	Ações	leitura (dados)						✓								
		imprimir (regra)					✓			✓			✓	✓		
Funções Adapt.	Fun.	*F*	B						✓							
	Obj.	*p1*	P						"µ"							
		p2	P						"o"							
		g1	G													

Figura 3 - Exemplo de tabela de decisão adaptativa

Um algoritmo genérico do funcionamento de qualquer dispositivo dirigido por regras foi apresentado em Neto (2001). Apresenta-se a seguir uma versão deste algoritmo adaptado especificamente para as Tabelas de Decisão Adaptativas.

- Primeiramente, executa-se uma busca nas condições de todas as regras da tabela de decisão, verificando se há alguma regra compatível com a situação atual do ambiente.
- Não havendo nenhuma regra compatível, nenhuma ação é executada.
- Havendo apenas uma regra compatível, esta regra é selecionada para a aplicação.
- Havendo mais de uma regra compatível, obtém-se o que se denomina de configuração não-determinística. Neste caso, selecionam-se todas as regras compatíveis para a aplicação.
- A regra selecionada para aplicação é executada. Esta execução é feita em 3 partes:
- Executa-se a chamada de função adaptativa anterior. Sendo esta chamada diferente de vazia, o conjunto de regras do dispositivo sofrerá alteração; Neste caso há a possibilidade de a execução da ação adaptativa excluir a própria regra chamadora. Desta forma a execução do processamento desta regra é abortada após a eliminação da regra, a Tabela de Decisão Adaptativa procura outra regra para execução.

- Executam-se todas as ações cujo campo da regra fora solicitado, da mesma maneira que as ações são executadas em uma tabela de decisão convencional;
- Executa-se a chamada de função adaptativa posterior. Sendo esta chamada diferente de vazia, o conjunto de regras do dispositivo sofrerá alteração; Neste caso há também a possibilidade da execução da ação adaptativa excluir a própria regra chamadora.
- Havendo mais de uma regra selecionada para aplicação, todas as regras são executadas em paralelo.
- Uma vez executadas todas as chamadas a ações adaptativas e todas as ações solicitadas correspondentes à regra selecionada, o processo de operação da tabela de decisão é reiniciado.

Para o projeto da solução proposta pelo presente trabalho, diversos formalismos adaptativos atualmente disponíveis foram analisados. Seguindo alguns critérios de generalidade, facilidade de manipulação do dispositivo e aplicabilidade em outras áreas do conhecimento, concluiu-se que o dispositivo denominado Tabela de Decisão Adaptativa é um dos dispositivos mais recomendados para a representação computacional do dispositivo adaptativo.

A facilidade e a disponibilidade de recursos para a manipulação de tabelas nas linguagens de programação modernas, o seu funcionamento quase que intuitivo e a sua já citada aplicabilidade em diversas áreas, em contraste a outros dispositivos adaptativos, foram alguns dos fatores que decidiram pela escolha da tabela de decisão adaptativa. Adicionalmente a esses fatores, a proposta do presente trabalho de disponibilizar um protótipo de ferramenta para o auxilio no processo de tomada de decisão ao publico leigo em computação reforçou a decisão na escolha final do dispositivo.

Conclui-se com isso o presente capítulo. Seguindo a apresentação do modelo formal a ser utilizado como base da solução proposta no presente trabalho, conclui-se a primeira parte, responsável pela apresentação de conceitos utilizados no projeto da solução, com a escolha da forma de implementação utilizada. O próximo

capítulo trata especificamente da apresentação da tecnologia utilizada para implementação da solução em um ambiente computacional.

4 LINGUAGENS FUNCIONAIS

A tecnologia utilizada no auxílio de tomada de decisão utilizada no projeto da solução apresentada no presente trabalho foi apresentada no último capítulo, juntamente com um breve histórico e as justificativas para a sua utilização.

Este capítulo é responsável pela apresentação da forma de implementação utilizada no projeto da solução do trabalho, no caso a utilização de linguagem funcional, de forma a concluir a primeira parte do mesmo, responsável pela introdução dos fundamentos utilizados na proposta.

O capítulo está estruturado da seguinte forma: uma seção para uma breve apresentação da origem das linguagens funcionais, outra seção para uma introdução dos seus fundamentos, uma seção sobre a primeira linguagem de programação funcional, uma outra seção sobre algumas características e finalmente uma seção responsável pela apresentação da escolha específica utilizada na presente proposta.

4.1 ORIGEM

Diversos fatores influenciam direta ou indiretamente o projeto de uma linguagem de programação. Um dos mais importantes é a arquitetura computacional utilizada (SEBESTA, 1996).

A maioria das linguagens de programação disponíveis nos últimos 50 anos vem sendo projetada sobre a arquitetura computacional prevalente, a denominada arquitetura de Von Neumann. Praticamente todos os computadores digitais construídos desde a década de 1940 até os dias atuais foram baseados nesta arquitetura As linguagens de programação baseadas na arquitetura de Von Neumann são denominadas linguagens imperativas (SEBESTA, 1996), (MITCHELL, 1996).

Devido a esse embasamento, os principais recursos disponíveis nas linguagens imperativas são as variáveis, que são a representação das áreas de

memória de um computador digital; os comandos de atribuição, que são baseados no processo de transmissão dos dados da CPU para a memória, e vice-versa; e os comandos de iteração, que representam a maneira mais eficiente de implementação do processo de repetição na arquitetura de Von Neumann (SEBESTA, 1996).

As linguagens de programação imperativas baseiam o seu processo de computação, basicamente, na atribuição de valores às variáveis. Praticamente todos os programas escritos em linguagens de imperativas podem conter funções ou procedimentos que alteram variáveis globais em relação ao seu corpo. Esta característica é denominada pelas referências clássicas como efeito colateral (SEBESTA, 1996), (MARCOTTY; LEDGARD, 1986).

Com isso, pode-se obter o absurdo de uma operação simples de soma perder a propriedade de comutatividade. Um exemplo muito interessante deste tipo de funções e de suas conseqüências pode ser encontrado em Marcotty e Ledgard (1986), p. 426.

Este tipo de possibilidade dificulta o entendimento do programa, e torna muito mais custoso o processo de manutenção do mesmo, pois pode ser necessária a revisão de grande parte do programa (MARCOTTY; LEDGARD, 1986).

Outras bases para o projeto de linguagens de programação foram desenvolvidas ao longo do tempo, algumas delas particularmente orientadas a paradigmas e metodologias, em contraste a uma solução de execução eficiente em uma arquitetura computacional particular. O paradigma funcional de programação é a base do projeto de um dos mais importantes estilos de programação que não o imperativo. Este estilo de programação tem no mundo prático a representação pelas linguagens de programação denominadas funcionais ou aplicativas (SEBESTA, 1996).

Uma linguagem funcional, ou aplicativa, é aquela cujo meio primário de computação é a aplicação de funções a determinados parâmetros. A programação de computadores digitais pode ser efetuada em uma linguagem funcional sem a utilização do tipo de variável disponível nas linguagens imperativas, sem os comandos de atribuição e sem comandos de iteração (MITCHELL, 1996).

Um dos fatores que mais influencia as diferenças entre as linguagens funcionais e as linguagens de programação imperativas é o modelo formal computacional adotado como fundamento. Enquanto no paradigma imperativo, o modelo computacional é o dispositivo conhecido como Máquina de Turing, no

paradigma funcional o modelo computacional utilizado é o Cálculo Lambda (MITCHELL, 1996).

Em uma linguagem puramente funcional, todas as suas construções sintáticas podem ser reduzidas ou transformadas em expressões do Cálculo Lambda. Pode-se então, através de alguns teoremas, provar que determinada função produz exatamente o resultado que se pretende, em todos os casos, sem exceções. Isso é interessante no tocante à qualidade e à confiabilidade do software produzido, questão relevante e extremamente necessária atualmente (MITCHELL, 1996), (GORDON, 1988).

4.2 FUNDAMENTOS

Como dito anteriormente, as linguagens funcionais são baseadas no modelo formal conhecido como Cálculo Lambda. O Cálculo Lambda foi proposto por Alonzo Church em 1932 inicialmente como parte de um programa para a construção de um sistema lógico na área de fundamentos da matemática. Em 1935 Stephen Kleene e John Rosser, alunos de Church à época, demonstraram, usando os argumentos e o Teorema de Gödel, que o sistema lógico formal proposto por Church resultava ser inconsistente. Entretanto o núcleo do sistema proposto por Church foi demonstrado por Kleene como sendo consistente e completo, ou seja, não-diagonalizável. Este núcleo consistente é o Cálculo Lambda. O fato de as linguagens de programação funcionais serem baseadas no Cálculo Lambda fornece a elas uma base matemática e teórica extremamente forte, conferindo-lhes ainda algumas características peculiares (BARENDREGT, 1984).

O conceito da Definibilidade Lambda[5], formalizada por Church, é baseado em sua teoria do Cálculo Lambda e é suficientemente robusto para abranger a notação e a demonstração do que é chamado de procedimento ‘efetivamente computável’, conforme a definição vigente à época, oriunda do século XIX por Hilbert. Posteriormente, colocando-o em equivalência com outros conceitos como Máquinas

[5] Determinado procedimento é dito Lambda-Definível se houver uma função $F \in \Lambda$, com Λ sendo o conjunto de todas as Expressões Lambda, de forma que F consiga reproduzir o seu efeito sobre os mesmos determinados parâmetros.

de Turing (Turing em 1936/1937) e Recursividade (Gödel – Herbrand, Kleene 1936); e pela sua simplicidade de sintaxe, torna-se a maneira ideal para ilustrar tal procedimento (BARENDREGT, 1984).

4.2.1 Cálculo Lambda

A presente subseção e todas as suas partes componentes foram fortemente embasadas em Barendregt (1984), Gordon (1979), Gordon (1988) e Mitchell (1996).

O Cálculo Lambda é uma teoria para a definição de funções. As expressões descritas nesta notação são chamadas Expressões Lambda, e podem ser classificadas, basicamente, em três tipos: variáveis, aplicações e abstrações.

As variáveis representam o valor simbólico que lhes foi atribuído em determinado ambiente, similarmente a atribuição de valores a variáveis em um ambiente de programação. Por exemplo: x, y, z.

As aplicações denotam a aplicação de uma função a um conjunto de expressões como parâmetros, argumentos. Por exemplo: (F1 F2) significa aplicar a função F1 utilizando F2 como argumento.

As abstrações, como o nome indica, são construções para a abstração de algum conceito; e definem propriamente uma função. Pode-se interpretá-las como a declaração de uma função ou procedimento em ambiente de programação. Por exemplo: λV.E denota uma função cujo corpo é E, com a variável de ligação é V. Novamente, em comparação a um ambiente de programação, o parâmetro da função é V, o corpo é E e, no caso de uma aplicação sobre esta construção, todas as instâncias de V em E serão substituídas pelo argumento referenciado na chamada (GORDON, 1988).

O conceito básico para a manipulação de Expressões Lambda é a conversibilidade de expressões pela aplicação de operações chamadas conversões.

O conceito de conversão origina-se na propriedade de conversibilidade de uma teoria. Significa que, por meio da manipulação de determinadas características das expressões, podemos converter uma expressão em outra sem a perda da expressividade da mesma. Originalmente, dois tipos de conversões foram

disponibilizadas no Cálculo Lambda: α e β-conversão. A terceira conversão disponível, a η-conversão, não foi originalmente apresentada como uma conversão do Cálculo Lambda, mas sim como uma operação de extensão do Cálculo Lambda original e, por isso, será apresentada mais adiante (BARENDREGT, 1984).

4.2.1.1 β-conversão

A β-conversão consiste na substituição de uma variável em uma abstração por um argumento em todas as suas ocorrências no corpo da abstração.

Por exemplo: $(\lambda x.xx)y = yy$

Onde x foi substituído por y em todas as suas ocorrências. Isso leva diretamente a introdução do conceito de variáveis livres e variáveis ligadas.

Variáveis ligadas são aquelas ‘declaradas’ no ‘cabeçalho’ da abstração. No exemplo acima, x é uma variável ligada. Na substituição de uma variável por uma expressão, todas as ocorrências da variável ligada no corpo da abstração serão substituídas pela expressão.

Por exemplo: $(\lambda xy.xyy)z = \lambda y.zyy$

A retirada da variável x do cabeçalho da abstração acima indica que a variável não mais existe no corpo da mesma. Se a variável ocorrer no corpo da abstração, mas não em seu cabeçalho, podemos dizer que esta variável é livre. No exemplo abaixo, x é uma variável ligada e w é uma variável livre:

$(\lambda x.xxwx)$

Pois a substituição de x por uma expressão qualquer y implicaria na expressão yywy, com todas as ocorrências de x no corpo da abstração sendo substituídas por y.

4.2.1.2 α-conversão

O princípio básico no conceito de α-conversão é o de que, não surgindo nenhum conflito de nomes, as variáveis de uma expressão podem ser renomeadas. Por exemplo:

$(\lambda x.xx) = (\lambda y.yy)$

utilizando o conceito de α-conversão e havendo a substituição de x por y. Note que isso somente é possível se as variáveis forem ligadas, pois se

$(\lambda x.xy) = (\lambda y.yy)$

com o mesmo conceito utilizado no exemplo anterior, y que não era uma variável ligada acaba por tornar-se, o que invalida o conceito de conversibilidade.

4.2.1.3 Teorema do ponto fixo

Para qualquer Expressão Lambda F, existe uma Expressão Lambda X tal que:

FX = X

Ou seja, aplicando X a F, obtêm-se novamente a expressão X.

Exemplo (BARENDREGT, 1984): suponha $W=\lambda x.F(xx)$ e X=WW

$X = WW \Rightarrow (\lambda x.F(xx))W \Rightarrow$
$(\lambda x.F(xx))W \Rightarrow F(WW) \Rightarrow$
$F(WW) \Rightarrow F(X)$

4.2.1.4 Conjunto de variáveis livres

O conjunto de variáveis livres de uma Expressão Lambda M (denotado por $V_L(M)$), como o próprio nome indica, é o conjunto de todas as variáveis não ligadas na expressão M.

Por exemplo: L = λx.xy. Tem-se que $V_L(L) = \{y\}$.

Seguindo o exemplo acima: N = λxy.xy; $V_L(N) = \emptyset$. Ainda, sendo O = λx.xzxw; $V_L(O) = \{z,w\}$.

Uma Expressão Lambda M é considerada fechada se $V_L(M) = \emptyset$. Representa-se por Λ^0 o conjunto de todas as Expressões Lambda fechadas, e por $\Lambda^0(x)$ o conjunto de todas as Expressões Lambda onde somente a variável x é livre. Representa-se por Λ o conjunto de todas as Expressões Lambda.

Portanto se $V_L(M) = \{\emptyset\}$, então $M \in \Lambda^0$ e; se $V_L(N) = \{z\}$, então $N \in \Lambda^0(z)$.

Define-se também que:

- O conjunto de variáveis livres de uma variável é ela mesma ($V_L(x) = \{x\}$).
- O conjunto de variáveis livres de uma aplicação é a união dos conjuntos de variáveis livres de cada Expressão Lambda ($V_L(MN) = V_L(M) \cup V_L(N)$).

4.2.1.5 Subtermos

Subtermo é uma Expressão Lambda contida em outra, de forma similar ao conceito de subconjuntos. A expressão λxy.y(λz.z)x tem como subtermo a expressão (λz.z). Uma Expressão Lambda pode conter mais de um subtermo,

inclusive subtermos repetidos. Como exemplo, tem-se (λx.(λy.yy)x(λy.yy)), com duas ocorrências do subtermo (λy.yy).

4.2.1.6 Extensibilidade

Para a obtenção desta extensão do Cálculo Lambda, partiu-se do princípio que se duas Expressões Lambda produzem os mesmos resultados sob os mesmos argumentos, elas são iguais. Esta operação não foi prevista na teoria desenvolvida originalmente, por isso a sua denominação como extensão. Por exemplo, λx.Mx e M produzem o mesmo resultado quando submetidos ao mesmo parâmetro (N, por exemplo). Então a regra

$$Mx = Nx \Rightarrow M = N \quad , \text{com } x \notin V_L(MN)$$

é derivada da regra de extensibilidade e dá forma a mais um tipo de conversão, a η-conversão, a qual indica que

$$\lambda x.M = M$$

A teoria estendida por esta Conversão Lambda é denominada de $\lambda\eta$-calculus ou Cálculo Lambda Estendido. Uma de suas importantes características baseia-se na possibilidade de um mapeamento direto de Expressões Lambda em estruturas compostas por combinadores (BARENDREGT, 1984).

4.2.1.7 Consistência

Para a definição de consistência, definir-se-á primeiramente o conceito de equação. Uma equação é uma expressão da forma M = N, onde M, N $\in \Lambda$, com Λ

representando o conjunto de todas as Expressões Lambda. Uma equação é dita fechada se M, N $\in \Lambda^0$, ou seja, $V_L(MN) = \varnothing$).

Uma teoria formada por um conjunto de equações é denominada consistente se não for possível provar que todas as suas equações são fechadas. Do contrário, a teoria é dita inconsistente.

Para exemplificar, suponha uma teoria U, composta por um conjunto de equações. Se *Lambda + U* é uma teoria obtida pela adição das equações de U à teoria Lambda como axiomas, então U é consistente (usa-se a notação Con(U)), se Con(*Lambda + U*).

4.2.1.8 Compatibilidade

Duas Expressões Lambda, M e N, são ditas incompatíveis (M # N) se a formula M = N for inconsistente (¬Con(M = N)).

Esta propriedade será mostrada, através do exemplo abaixo:

Para K = λxy.x e S = λxyz.xz(yz), K # S.

Lambda + K = S tem de ser compatível.

Para todo e qualquer X,Y,Z Expressões Lambda:

KXYZ = SXYZ

XZ = XZ(YZ)

Supondo X ≡ Z ≡ I. Então, para todo e qualquer Y

I = YI

Supondo Y ≡ KM, M sendo qualquer Expressão Lambda, tem-se que:

I = M

4.2.1.9 Forma Normal

Uma Expressão Lambda qualquer é dita β-forma normal (β-nf), ou simplesmente forma normal (nf), se não houver a possibilidade de nenhuma β-conversão nesta expressão. Por exemplo:

(λx.xy)z tem como forma normal zy. Obtem-se a Expressão Lambda zy pela aplicação de z à variável ligada x na Expressão Lambda supracitada. Não havendo, portanto, mais nenhuma β-conversão aplicável, observa-se que ela é uma β-forma normal (β-nf).

Uma Expressão Lambda é chamada de forma normal se não houver nenhuma conversão possível a ser aplicada. Por exemplo, a Expressão Lambda x é uma forma normal (diz-se que a Expressão Lambda está em forma normal), e pode sê-la de uma outra Expressão Lambda, (λy.y)x por exemplo, pois executando uma β-conversão na expressão (λy.y)x, obtem-se como resultado x, e em x não se pode executar mais nenhuma β-conversão.

Isso indica uma das principais propriedades das formas normais: uma Expressão Lambda M tem uma forma normal N, se $M =_\beta N$, ou seja, M é β-conversível para N, e N é uma forma normal.

N estará em forma normal se não houver nenhum subtermo elegível para uma β-conversão, ou seja, nenhum subtermo na forma (λx.M)N.

Similarmente, uma expressão R estará em λη forma normal, uma expressão no λη-calculus ou Cálculo Lambda Estendido, se não houver nenhum subtermo elegível para β- ou η-conversões, ou seja, nenhum subtermo nas formas (λx.M)N ou (λx.Mx).

Desta forma, afirma-se que se duas expressões são conversíveis, então elas possuem a mesma forma normal. Este é o princípio básico do Teorema Church-Rosser, o qual indica:

$M = N \rightarrow \exists S\ (M = S \wedge N = S),$

com M, N e $S \in \Lambda$.

4.2.1.10 Reduções

Quando indicado que uma Expressão Lambda $(\lambda x.x)y = y$, o símbolo de igualdade (=) indica que $(\lambda x.x)y$ é conversível para y. Porém isso não significa que $y = (\lambda x.x)y$, o que indicaria reprocicidade na operação de conversão. Portanto, utilizar-se-á a notação $(\lambda x.x)y \rightarrow y$, indicando que y é resultado da computação de $(\lambda x.x)y$.

A relação de conversibilidade em Lambda não é a única relação de igualdade possível a ser analisada por reduções. Suponha M, N um par de Expressões Lambda, pertencente à relação R. Então, se esta relação for compatível, (ZM, ZN), (MZ, NZ) e ($\lambda x.M$, $\lambda x.N$) também pertencem a esta relação, ou seja ela é fechada sob outros aspectos, em tempo, aplicação e abstração.

Uma relação de igualdade, ou congruência, em Lambda é uma relação compatível de equivalência, e uma relação de redução em Lambda é uma relação compatível, reflexiva e transitiva.

Sendo R uma noção de redução, a qual é basicamente uma relação binária em Lambda, as seguintes relações binárias são induzidas:

$\rightarrow_R$ R-redução em um passo, indica o fechamento compatível da relação;

$\rightarrow\rightarrow_R$ R-redução, indica o fechamento reflexivo e transitivo da relação;

$=_R$ R-conversibilidade, indica a equivalência da relação.

Mostrar-se-á um exemplo, supondo que Lambda seja uma noção de redução:

$(\lambda x.xx)(\lambda y.y)z \rightarrow_\beta (\lambda y.y)(\lambda y.y)z$

$\rightarrow_\beta (\lambda y.y)z$

$\rightarrow_\beta z$

Então, pela notação indicada:

$(\lambda x.xx)(\lambda y.y)z \rightarrow\rightarrow_\beta z$

e, portanto:

$(\lambda x.xx)(\lambda y.y)z =_\beta z$

4.2.1.11 Conceitos de computação

Conforme pode ser observado, a similaridade dos conceitos de redução e computação é evidente. Traçando um paralelo de conceitos, pode-se dizer que uma redução em um passo ($\rightarrow_R$) seja um passo de computação, uma redução ($\rightarrow\rightarrow_R$) seja o resultado do fechamento reflexivo e transitivo do passo de computação e uma equivalência ($=_R$) seja a noção de computabilidade.

Ainda na comparação entre conceitos, a forma normal de uma Expressão Lambda indicará o fim da computação da mesma e, portanto, a existência de uma forma normal indicaria a possibilidade real de computação.

Outro conceito que vale a pena ser analisado é a existência de subtermos em Expressões Lambda. Havendo subtermos independentes em uma mesma expressão, os mesmos podem ser reduzidos paralelamente, visto que de acordo com o teorema de Church-Rosser, não importa a ordem em que as reduções sejam aplicadas, pois se a Lambda Expressão apresentar a mesma forma normal, elas são equivalentes. Desta forma, um interpretador ou compilador de uma linguagem funcional pode por si só ser capaz de paralelizar um programa, independentemente da interação do programador. Ao contrário das linguagens de programação imperativas, que exigem um esforço razoável por parte do programador em obter as

‘unidades’ do programa que possam ser paralelizáveis, como pode ser visto em um exemplo em Sebesta (1996), p.624.

Pode-se observar a inexistência de efeitos colaterais nas Expressões Lambda. Como em uma teoria de equações matemáticas, a expressividade de uma expressão é tão somente dependente dela mesma. Havendo a possibilidade de redução, a operação é aplicada até que exista uma forma normal, ou que se conclua que não há a possibilidade de reduzi-la a uma forma normal. Seguindo as propriedades da álgebra do Cálculo Lambda, pode-se provar que determinada expressão e, portanto em uma linguagem puramente funcional, um programa, executa exatamente aquilo que a se propõe, em todos os casos, como dito anteriormente.

4.3 LISP

A primeira linguagem de programação funcional foi projetada com o intuito de prover recursos para o processamento de listas, uma necessidade que surgiu a partir das primeiras aplicações vislumbradas no campo da inteligência artificial. Esta linguagem foi denominada LISP, que é uma contração para LISt Processing, processamento de listas. A primeira versão do LISP é algumas vezes referenciada por LISP puro, por ela ser uma linguagem puramente funcional (SEBESTA, 1996).

Em McCarthy (1960) é apresentada a primeira versão da linguagem LISP, partindo inicialmente de algumas definições a respeito de funções, como o conceito de funções parciais, as expressões e predicados proposicionais, o conceito de expressões condicionais e de expressões recursivas. A partir deste arcabouço, foi mostrada uma notação de definição de funções baseadas no Cálculo Lambda, de forma a trabalhar totalmente sobre o domínio das funções e formas matemáticas. Portanto, partindo apenas de definições matemáticas, um método de programação de computadores é apresentado. Desta forma, o LISP em sua primeira versão é classificado como uma linguagem de programação puramente funcional, pois apresenta todas as suas construções derivadas de expressões matemáticas e lógicas.

Definindo logo após o conceito de expressões simbólicas, as denominadas *S-Expressions*, e já havendo definido o conceito de átomos e listas, é definido um conjunto de funções básicas para a manipulação destas formas, como por exemplo, as funções *equal*, *among* e *assoc*, dentre várias outras, muitas ainda disponíveis nas versões mais recentes do LISP. Especialmente, a função universal *apply* é apresentada, juntamente com o seu propósito e um 'passo-a-passo' na aplicação de uma função a uma lista de argumentos. Esta função exerce a função de modelo computacional, equivalente à Máquina de Turing Universal, e que no sistema de programação exerce a função de interpretador das expressões do LISP (McCARTHY, 1960).

Outra característica importante apresentada é a função *eval*, que serve tanto como uma definição formal da linguagem LISP, como um interpretador da mesma. Juntamente com o conceito de gerenciamento e limpeza da área de armazenamento *heap*, o denominado processo de *Garbage Collection*, mostra como o LISP contribuiu para o desenvolvimento das linguagens de programação modernas ao apresentar diversos recursos inovadores.

A publicação de McCarthy (1960) é extremamente importante para esta pesquisa, pois demonstra de maneira simples a definição de uma linguagem e um sistema de programação funcional, baseados em expressões simbólicas – um dos grandes desafios computacionais à época de sua publicação -, além da desejada tarefa de definir um método de programação de computadores totalmente baseado no conceito de funções matemáticas.

4.4 CARACTERÍSTICAS

O objetivo do projeto de uma linguagem de programação funcional é imitar o comportamento das funções matemáticas o mais fielmente possível. Isto resulta em uma abordagem para a solução de problemas que é fundamentalmente diferente do método utilizado com as linguagens de programação imperativas (SEBESTA, 1996), (GORDON, 1988).

Em uma linguagem imperativa, uma expressão é avaliada e o seu resultado é armazenado em uma área de memória, representada por uma variável do programa. Esta atenção em relação ás áreas de memória da máquina resulta em uma metodologia de programação de nível relativamente baixo, no tocante à dependência da arquitetura computacional utilizada (GORDON, 1988).

Uma linguagem de programação funcional não faz uso do recurso de instruções de atribuição ou de variáveis, no sentido imperativo, para o armazenamento do estado do programa executado. Este fato deixa o programador livre das preocupações a respeito das áreas de memória e suas respectivas variáveis. Sem a utilização de variáveis, surgem algumas características muito interessantes do paradigma funcional (GORDON, 1988).

Primeiramente, a execução de um programa puramente funcional não apresenta o que se denomina estado, no conceito da semântica operacional. A execução de uma função produz sempre o mesmo resultado quando aplicada aos mesmos parâmetros. Esta característica é denominada transparência referencial, e torna a semântica das linguagens puramente funcionais muito mais simples em relação à das linguagens imperativas. Embora algumas linguagens funcionais forneçam aos desenvolvedores recursos de programação imperativos, a semântica destas linguagens ainda é mais simples que a de qualquer linguagem imperativa (SEBESTA, 1996), (GORDON, 1988), (MITCHELL, 1996).

Ainda como característica da ausência de variáveis, não há a possibilidade da definição de construções de repetição iterativas. Nas linguagens de programação funcionais as construções de repetição são definidas pelo conceito de função recursiva (GORDON, 1988).

Curiosamente, este foi um dos fatos que mais contribuiu para a não utilização de linguagens de programação funcionais nos primórdios da computação. Para a execução de construções recursivas nos computadores com arquitetura de Von Neumann, uma área de memória de capacidade considerável era necessária, além de diversos acessos CPU - memória principal, que são executados através do gargalo de Von Neumann, ponto crítico de desempenho desta arquitetura computacional (SEBESTA, 1996).

As linguagens de paradigma imperativo caracterizam-se pela execução eficiente de programas sobre a arquitetura de Von Neumann. Com a escassez de recursos computacionais (poder de processamento bruto e capacidade de memória)

característica do início da utilização dos computadores digitais, a disseminação da linguagem FORTRAN, que foi a primeira linguagem imperativa de alto nível, em detrimento a utilização da linguagem LISP como linguagem de programação padrão para o desenvolvimento de aplicações de cunho genérico foi decisiva para a história dos projetos de linguagens de programação (SEBESTA, 1996).

4.5 ESCOLHA DA LINGUAGEM DE DESENVOLVIMENTO

Embora a utilização de linguagens de programação funcionais tenha sido desestimulada comercialmente, devido aos problemas iniciais de desempenho, diversas linguagens funcionais foram desenvolvidas ao longo dos anos. As linguagens LISP (em seus vários dialetos), ML e Haskell, entre outras, são alguns dos exemplos de linguagens deste paradigma (SEBESTA, 1996).

A linguagem Scheme, que é um dialeto do LISP, surgiu no MIT durante os anos 70, e é caracterizada pelo seu tamanho reduzido, o uso de escopo estático e o tratamento de funções como entidades de primeira classe (SEBESTA, 1996), (ABELSON; SUSSMAN, G; SUSSMAN, J, 1996).

Common Lisp foi criada com o intuito de combinar os recursos de diversos dialetos do LISP disponíveis no início dos anos 80. Devido a esta origem, ela é uma linguagem grande e complexa, porém como a sua base é o LISP original, a sua sintaxe e o seu conjunto de funções básicas é relativamente simples. Interessantemente, Common Lisp permite tanto o escopo estático como dinâmico (SEBESTA, 1996), (WINSTON; HORN, 1989).

A linguagem de programação ML é um pouco diferente das linguagens LISP. A sua sintaxe é similar a das linguagens de programação imperativas, como C e Pascal. Além disso, ela faz uso da declaração de tipos, e o tipo de qualquer variável e expressão pode ser determinado em tempo de compilação (SEBESTA, 1996).

Haskell é similar à linguagem ML em vários aspectos, como a sintaxe, o uso de escopo estático e o método de inferência de tipos. Ela difere da ML em ser uma linguagem puramente funcional, ou seja, não inclui nenhum tipo de construção

imperativa. Alguns das suas características foram originadas na linguagem de programação Miranda[6] (SEBESTA, 1996).

A escolha da linguagem de desenvolvimento utilizada nesta pesquisa recaiu sobre a linguagem Scheme, devido a sua relativamente ampla divulgação no ambiente acadêmico; a relativa facilidade de aprendizado; o fato de ser um dialeto LISP, que permite a tradução de seus programas para outras linguagens LISP com muito pouco esforço; e a disponibilidade de ambientes de desenvolvimento que apresentam ferramentas interessantes ao processo de programação.

Desta forma conclui-se o presente capítulo e a primeira parte do trabalho, responsável pela apresentação dos fundamentos teóricos utilizados na proposta apresentada.

Primeiramente, o processo de decisão, especificamente o processo de decisão em uma empresa foi apresentado, juntamente a algumas técnicas utilizadas para o auxilio deste processo. Em seguida, a tecnologia utilizada na proposta do presente trabalho foi apresentada, juntamente a um breve histórico da tecnologia e as características básicas responsáveis pela escolha efetuada. Finalmente, a forma de implementação desta tecnologia em ambiente computacional foi apresentada, mostrando o paradigma funcional de desenvolvimento de sistemas, alguns exemplos de linguagens funcionais e a escolha adotada no presente trabalho.

No próximo capítulo, inicia-se a segunda parte do trabalho, responsável pela apresentação da proposta em si, especificação do projeto, aspectos de implementação, escolhas de projeto e justificativas das mesmas, que surgiram ao longo do desenvolvimento do presente trabalho.

[6] Miranda é uma marca registrada de Research Software Ltd. of England.

5 ESPECIFICAÇÃO DO PROJETO

Com a finalização da primeira parte do presente trabalho, concluiu-se a apresentação dos fundamentos utilizados para o projeto da proposta. Inicia-se agora a segunda parte, responsável pela apresentação da proposta propriamente dita.

Este capítulo é responsável pela apresentação da especificação do projeto da proposta, a descrição da sua arquitetura e as justificativas das técnicas e tecnologias utilizadas. Ele está estruturado em três seções, de forma a uma seção ser responsável pela apresentação da arquitetura da solução proposta, uma seção responsável pela descrição dos componentes da solução e uma terceira seção responsável pela apresentação das soluções adotadas em cada um dos módulos, bem como as suas justificativas.

5.1 ARQUITETURA PROPOSTA

Como apresentado no primeiro capítulo, o objetivo do presente trabalho é apresentar uma aplicação da tecnologia adaptativa ao processo de tomada de decisão nas organizações, especificamente, uma proposta de ambiente para o desenvolvimento de aplicações cujo núcleo seja um dispositivo adaptativo.

Para atingir este objetivo, foi especificada uma arquitetura que inclui: uma linguagem de programação para a especificação dos programas nesse ambiente; o desenvolvimento de um compilador ou interpretador que transforme o código-fonte escrito nesta linguagem de entrada nas estruturas básicas aceitas por um módulo ambiente de execução; e a construção do ambiente de execução em si, de forma a atender aos requisitos da tecnologia adaptativa. Estes são os elementos principais na obtenção do ambiente desejado.

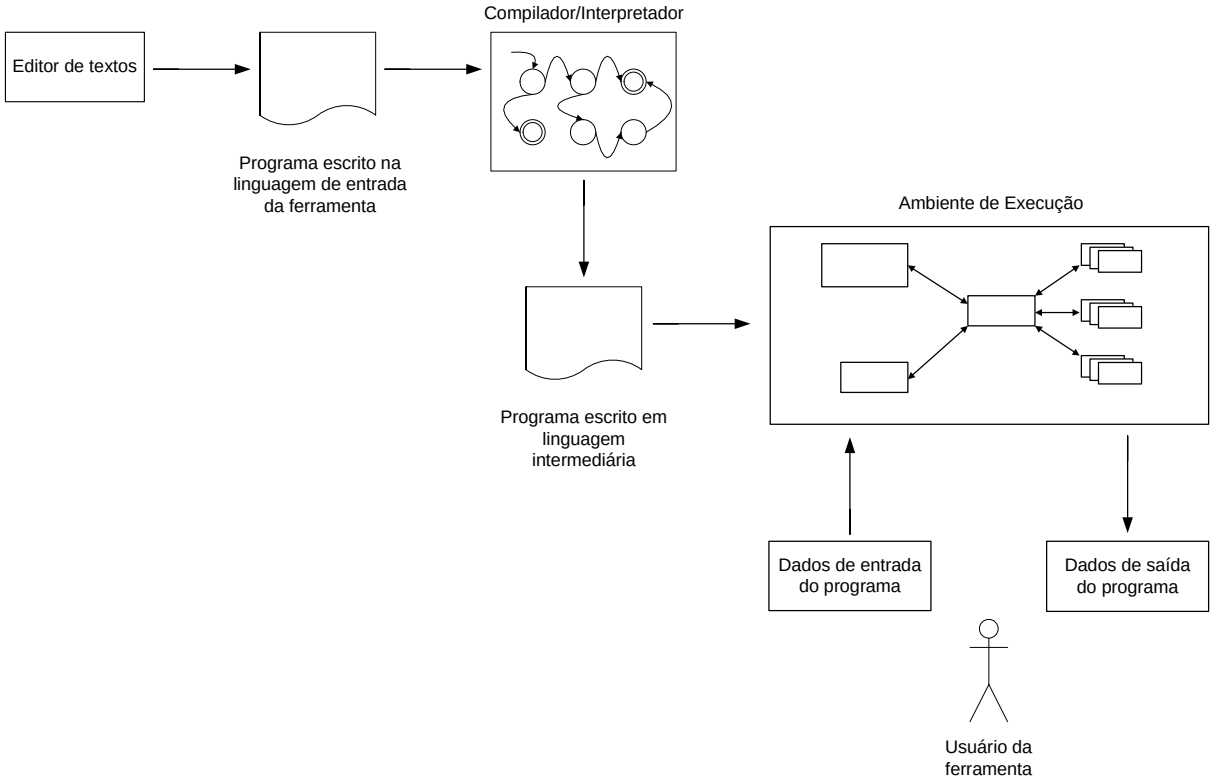

Figura 4 - Arquitetura proposta

A arquitetura proposta apresenta o mínimo de componentes necessários para a obtenção de um ambiente de desenvolvimento. Em qualquer ambiente, faz-se necessário um meio de comunicação entre o desenvolvedor e o ambiente propriamente dito, tarefa esta que é executada pelo processador da linguagem de entrada; um dispositivo capaz de transformar as instruções desta linguagem de entrada em instruções que possam ser compreendidas pelo ambiente de execução, neste caso um compilador ou um interpretador; e o ambiente de execução propriamente dito (LOUDEN, 2004). Os demais módulos apresentados na proposta são módulos incorporados à solução principal de forma facilitar a utilização da ferramenta e permitir a melhor integração com outras ferramentas.

A utilização da ferramenta proposta segue, em linhas gerais, os passos descritos abaixo. Observa-se que este procedimento, tanto em relação à ordem das tarefas quanto às tarefas em si contidas, pode ser aplicado à utilização de qualquer ferramenta no auxílio da tomada de decisão:

1-) O decisor executa o levantamento das variáveis do problema a ser analisado, suas características e possíveis ações a serem executadas no decorrer do processo de solução;

2-) Utilizando a linguagem de entrada da ferramenta proposta, o usuário desenvolve um modelo que atenda as premissas resultantes do processo de analise do problema-alvo;

3-) Um dos módulos da ferramenta efetua a transformação das instruções entradas pelo usuário em instruções que possam ser executadas diretamente no módulo ambiente de execução;

4-) O módulo ambiente de execução, com o modelo devidamente construído de acordo com as instruções do usuário da ferramenta, exercita-o em relação aos dados de entrada e saída, executando assim as regras de decisão codificadas;

5-) Uma lista de possíveis dados de saída pode ser gerado, decorrente das ações executadas durante a execução do modelo, dependendo exclusivamente da definição do modelo construído.

O resultado final do processamento é a escolha da regra, ou regras no caso de uma solução não-determinística, de acordo com os parâmetros modelados pelo usuário e os conceitos da tecnologia adaptativa, ou seja, a tomada de decisão propriamente dita, e a execução desta(s) regra(s).

5.2 LINGUAGEM DE ENTRADA

A arquitetura proposta apresenta uma linguagem de entrada para o desenvolvimento de soluções neste ambiente. Como mencionado anteriormente, a aplicação inicial do ambiente proposto é a utilização da ferramenta por parte de usuários leigos aos conceitos de computação, especificamente aos administradores e decisores em geral.

Com esse objetivo, a linguagem foi projetada tentando manter um alto nível de simplicidade, sem a utilização de construções muito rebuscadas, ou que precisassem de muito treino para a sua utilização. Tentou-se também fornecer uma linguagem de entrada amigável, que fizesse uso de conceitos conhecidos por parte

do público-alvo, visando à amenização da curva de aprendizado (PRESSMAN, 2005). Portanto, embora outros grupos de usuários possam fazer uso do ambiente proposto, o seu público-alvo principal é composto por pessoas leigas na área da computação.

Uma linguagem de programação deve permitir ao seu usuário a construção da solução correta para um problema de maneira fácil e natural. O propósito principal de uma linguagem de programação é auxiliar a tarefa do desenvolvedor, especialmente nas áreas mais problemáticas, como o projeto do programa, o seu correto entendimento e a sua verificação (MARCOTTY; LEDGARD, 1986).

O conceito da tabela de decisão é amplamente divulgado dentre diversas linhas do conhecimento, conforme as observações feitas no terceiro capítulo do presente trabalho. Desta forma, a idéia de um editor de tabelas de decisão surgiu inicialmente de forma natural, como uma solução óbvia para a implementação da linguagem de entrada da ferramenta.

Embora qualquer procedimento possa ser representado por uma tabela de decisão (NAGAYAMA, 1990), a codificação de alguns conceitos neste formato apresenta um nível de dificuldade relativamente alto. Optou-se então por um projeto de linguagem de entrada similar às linguagens de programação de paradigma imperativo disponíveis.

Poder-se-ia fazer uso da facilidade da definição e implementação de uma linguagem de entrada de paradigma funcional, já que será utilizada a linguagem Scheme no desenvolvimento da proposta. Esta facilidade surge justamente da manipulação de definições de programas como dados, conforme observado no quarto capítulo do presente trabalho.

Porém, devido à baixa divulgação dos conceitos da programação funcional, a escolha recaiu sobre o projeto de linguagem de entrada segundo os preceitos do paradigma imperativo de programação.

Esta linguagem de entrada oferecerá um conjunto de instruções extremamente reduzido em comparação às linguagens de programação comerciais disponíveis. Desta forma procura-se centrar o foco especificamente no processo de tomada de decisão, sem se preocupar com a manipulação de periféricos ou a possibilidade de construção e manipulação de estruturas de dados rebuscadas, características das linguagens comerciais de múltiplo propósito.

Um aspecto muito importante a ser considerado é o fato de uma tabela de decisão, seja ela adaptativa ou não, ser um dispositivo orientado a eventos. Portanto, embora a linguagem de entrada defina um código de forma linear, no momento da interpretação da tabela de decisão resultante, as instruções codificadas não serão executadas necessariamente na ordem em que elas foram codificadas.

Este fato não é extremamente relevante para o público-alvo da ferramenta proposta, visto que decisores estão habituados ao conceito de tabela de decisão e de seu funcionamento. Porém, um usuário que tem contato mais intenso com o conceito de programação de computadores pode não compreender, de início, o porquê de determinadas características.

Basicamente, a linguagem de entrada segue a forma de formulações de sentenças do tipo SE *fatos* ENTÃO *ações*, que podem ser encontradas na quase totalidade das linguagens de programação de alto nível. Este tipo de construção, levemente alterada para a inclusão das ações adaptativas (apresentadas no terceiro capítulo do trabalho) é a base para a definição dos eventos da tabela de decisão adaptativa.

A estrutura destas instruções é similar aos ‘comandos guardados’ apresentados em Dijkstra (1975). De maneira simplificada, um comando guardado é uma instrução normal de uma linguagem de programação, com a diferença que ela somente será executada quando o seu ‘guarda’ for verdadeiro. Este guarda é uma expressão que retorna valor booleano.

Pode-se associar o conceito do ‘guarda’ dos comandos guardados ao conceito de fatos apresentado acima, na formulação de sentenças. Se todos os fatos codificados na regra forem verdadeiros, as ações associadas aos fatos serão executadas (SEBESTA, 1996).

Encapsulando os comandos guardados correspondentes às regras da tabela de decisão em uma construção iterativa especial (DIJKSTRA, 1975), o processo de execução de uma tabela de decisão pode ser obtido. Esta construção iterativa executa a avaliação dos guardas de todos os comandos guardados codificados encapsulados. Havendo algum comando cujo guarda retorne valor *verdadeiro*, as respectivas instruções associadas a ele são executadas.

Esse processo é repetido até que não haja nenhum comando cujo guarda retorne valor *verdadeiro*. Quando não houver nenhuma instrução cujo guarda retorne valor *verdadeiro*, a construção iterativa é terminada.

O funcionamento básico de uma tabela de decisão é basicamente o mesmo: havendo alguma regra cujas todas as condições sejam verdadeiras, esta regra é selecionada para execução, e todas as ações associadas a esta regra são executadas. Este processo é repetido até que não haja nenhuma regra cujas condições propiciem a continuação do processamento da tabela de decisão.

Um conjunto de instruções responsável pelas operações de entrada e saída, de forma a possibilitar a solicitação de entrada de dados ao programa, bem como tornar possível a saída dos resultados do processamento faz-se também necessário. Adicionalmente, faz-se necessário um conjunto de instruções responsável pela declaração de variáveis e sua manipulação.

Com isso, definem-se quais são as funcionalidades básicas necessárias ao projeto específico da linguagem de entrada, a ser apresentado a seguir.

5.2.1 Definição da linguagem de entrada

A definição formal de uma linguagem pode se dar de diversas formas. Pode-se, por exemplo, fazer uso de um formalismo reconhecedor, de forma a este reconhecer somente as sentenças da linguagem desejada. Uma outra forma, muito interessante e amplamente utilizada são os dispositivos geradores, como, por exemplo, as gramáticas. As gramáticas são classificadas como dispositivos geradores porque são compostas por regras que permitem a geração de todas as sentenças da linguagem descrita.

Para ambos os formalismos, existem inúmeras formas pelas quais a representação pode ser descrita. Estas notações são denominadas meta-linguagens, visto que elas são linguagens através das quais as linguagens são especificadas (NETO, 1987), (AHO; SETHI; ULLMAN, 1986).

A linguagem de entrada do ambiente proposto foi formalizada na notação BNF (acrônimo para Backus-Naur Form) (AHO; SETHI; ULLMAN, 1986).

Certamente uma das mais populares formas de expressão da sintaxe de linguagens de programação, a BNF é uma metalinguagem que tem sido utilizada com sucesso para este objetivo desde que foi publicada pela primeira vez, na especificação da linguagem Algol 60. Maiores detalhes a respeito da meta-

linguagem BNF, o seu histórico e o formato das regras das regras de produção podem ser encontrados em Neto (1987), Aho; Sethi e Ullman (1986) e Sebesta (1996).

Devido à sua extensão, a formalização completa da linguagem de entrada na notação BNF foi disponibilizada no Anexo I do presente trabalho. A seguir a sintaxe da linguagem de entrada é descrita informalmente, seguida pela apresentação de um exemplo simples de tomada de decisão e sua codificação na linguagem, de forma a exemplificar e facilitar a compreensão da mesma.

Um programa escrito nesta linguagem de entrada é iniciado pela palavra reservada[7] **TDA**. Este programa pode ser dividido, basicamente, em três partes principais: a declaração de variáveis, que segue logo após a palavra reservada TDA; a declaração e codificação das funções adaptativas, e a codificação das regras da tabela de decisão adaptativa propriamente dita, que é introduzida pela palavra reservada **INICIO**. O final do programa é marcado pela palavra reservada **FIM**.

A seção responsável pela declaração e codificação das funções adaptativas é opcional; sendo a mesma omitida, o modelo comportar-se-á como uma tabela de decisão convencional.

Na linguagem projetada, a seção de declaração de variáveis é formada por várias declarações; cada uma consistindo da palavra reservada que corresponde ao seu tipo (Numero, Moeda ou Caract), seguida do símbolo de dois-pontos (:), seguida por uma lista de identificadores das variáveis, separados pelo símbolo de virgula (,). Cada declaração é terminada pelo símbolo de ponto-e-vírgula (;). Por exemplo:

Numero: A, B, C;
Caract: S1, S2;

A declaração das regras na linguagem de entrada segue a forma

SE condições ENTÃO ações;

[7] Uma palavra reservada é uma palavra que não pode ser utilizada como um identificador por ser reservada para uso da gramática da linguagem (MARCOTTY; LEDGARD, 1986).

Cada condição da lista de condições é uma expressão de comparação, composta de dois operandos separados por um dentre os seguintes operadores de comparação básicos (>, <, = ou !=), para as operações de maior que, menor que, igual e diferente, respectivamente. Cada um dos operandos de uma comparação pode ser um identificador de variável ou um valor absoluto, de forma a tornar a comparação válida. Por exemplo:

A = 3
5 > C
S1 = ‘Teste’

de acordo com as declarações exemplificadas anteriormente. A comparação

S1 > 5

é considerada uma comparação inválida, de forma que os operadores de comparação básicos > e < podem ser aplicados somente a variáveis e valores numéricos e de moeda.

Cada ação da lista de ações deve ser uma chamada a uma ação válida, seguida da correspondente lista de parâmetros necessários. Inicialmente foi disponibilizado um grupo reduzido de ações, responsável pela operação de atribuição de valores às variáveis declaradas, operações de entrada e saída de dados. Por exemplo:

C := 33
Imprimir C

A primeira ação atribui o valor numérico 33 (trinta e três) a variável C, declarada como tipo Numero. A segunda ação imprime o valor da variável C no dispositivo periférico padrão de saída. Normalmente, este dispositivo é a console gráfica utilizada.

Este conjunto básico de ações (atribuição, leitura e saída de dados) é suficiente para a modelagem de uma grande variedade de problemas. Isto se deve à estrutura naturalmente condicional das regras de uma tabela de decisão e a possibilidade natural de se controlar a chamada recursiva das regras, tem-se um conjunto de construções capaz de modelar praticamente qualquer problema[8].

O exemplo abaixo mostra o processo de decidir se deve-se ou não jogar tênis, em relação às condições climáticas. As variáveis que influenciam a tomada de decisão, ou as condições, são o tempo, a temperatura e a umidade do ar. A ação única correspondente à decisão é jogar tênis.

O domínio das variáveis são os seguintes:

- Tempo: Ensolarado, nublado ou chuvoso;
- Temperatura: Alta, média ou baixa;
- Umidade: Alta ou normal.

A figura 5 mostra as condições de decisão observadas, e a respectiva ação tomada em decorrência das condições, de acordo com a seguinte abreviação, com o intuito de facilitar o dimensionamento físico da tabela de decisão:

Ensolarado	→	Ens
Nublado	→	Nubl
Chuvoso	→	Chuv

[8] O teorema de Böhm-Jacopini estabelece que toda e qualquer função computável pode ser implementada em uma linguagem de programação que contenha três tipos possíveis de combinações de instruções: seqüencial, condicional e de iteração.

	Tempo	Temperatura	Umidade	Jogar Tenis?
1	Chuv	Baixa	Normal	Não
2	Chuv	Media	Alta	Não
3	Chuv	Media	Normal	Sim
4	Ens	Baixa	Normal	Sim
5	Ens	Media	Alta	Não
6	Ens	Media	Normal	Sim
7	Nubl	Baixa	Normal	Sim
8	Nubl	Alta	Alta	Sim
9	Nubl	Alta	Normal	Sim

Figura 5 - Tabela de decisão – Jogar tênis?

A seguir, a mesma tabela observada na figura 6 pode ser vista em um formato mais familiar, seguindo as representações utilizadas no terceiro capitulo do presente trabalho:

Tempo	Chuv	Chuv	Chuv	Ens	Ens	Ens	Nubl	Nubl	Nubl
Temperatura	Baixa	Média	Média	Baixa	Média	Média	Baixa	Alta	Alta
Umidade	Normal	Alta	Normal	Normal	Alta	Normal	Normal	Alta	Normal
Jogar Tenis?			✓	✓		✓	✓	✓	✓

Figura 6 - Tabela de decisão formatada

Com a tabela de decisão em formato conhecido, o programa fonte responsável pela codificação da tabela apresentada na figura 6 é mostrado na figura 7.

```
Tda
Caract: Tempo, Temperatura, Umidade;
Inicio
SE (Tempo=’Chuv’, Temperatura=’Baixa’, Umidade=’Normal’)
ENTÃO ();
SE (Tempo=’Chuv’, Temperatura=’Média’, Umidade=’Alta’)
ENTÃO ();
SE (Tempo=’Chuv’, Temperatura=’Média’, Umidade=’Normal’)
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo=’Ens’, Temperatura=’Baixa’, Umidade=’Normal’)
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo=’Ens’, Temperatura=’Média’, Umidade=’Alta’)
ENTÃO ();
SE (Tempo=’Ens’, Temperatura=’Média’, Umidade=’Normal’)
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo=’Nubl’, Temperatura=’Baixa’, Umidade=’Normal’)
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo=’Nubl’, Temperatura=’Alta’, Umidade=’Alta’)
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo=’Nubl’, Temperatura=’Alta’, Umidade=’Normal’)
ENTÃO (Imprimir Jogar_Tenis);
Fim;
```

Figura 7 - Código do exemplo na linguagem de entrada

Neste exemplo, pode-se observar as duas partes distintas no programa objeto: a parte de declaração das condições e a parte da codificação das regras da tabela de decisão. A primeira parte é representada pela porção de código mostrada na figura 8 e é compreendida entre as palavras-chave TDA e INICIO.

```
Tda
Caract: Tempo, Temperatura, Umidade;
Inicio
```

Figura 8 - Declaração das condições do programa exemplo

A segunda parte é representada pela porção de código mostrada na figura 9, compreendida entre as palavras-chave INICIO e FIM, é a responsável pela codificação das regras da tabela de decisão.

```
Inicio
SE (Tempo='Chuv', Temperatura='Baixa', Umidade='Normal')
ENTÃO ();
SE (Tempo='Chuv', Temperatura='Média', Umidade='Alta')
ENTÃO ();
SE (Tempo='Chuv', Temperatura='Média', Umidade='Normal')
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo='Ens', Temperatura='Baixa', Umidade='Normal')
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo='Ens', Temperatura='Média', Umidade='Alta')
ENTÃO ();
SE (Tempo='Ens', Temperatura='Média', Umidade='Normal')
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo='Nubl', Temperatura='Baixa', Umidade='Normal')
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo='Nubl', Temperatura='Alta', Umidade='Alta')
ENTÃO (Imprimir Jogar_Tenis);
SE (Tempo='Nubl', Temperatura='Alta', Umidade='Normal')
ENTÃO (Imprimir Jogar_Tenis);
Fim;
```

Figura 9 - Declaração das regras do programa exemplo

Esta introdução foi somente uma forma de familiarizar o uso da linguagem de entrada da ferramenta proposta. Ela de forma alguma substitui a necessidade do estudo da definição formal da linguagem, principalmente pelo fato do exemplo mostrado ser extremamente simples e não fazer uso de todos os recursos da ferramenta.

5.3 COMPILADOR/INTERPRETADOR

A materialização de uma linguagem de programação em um computador é denominada implementação. Linguagens de programação podem ser implementadas, basicamente, de duas formas distintas: por compilação e por interpretação. Embora soluções híbridas tenham sido projetadas, como por exemplo,

algumas implementações da linguagem de programação Java[9], a decisão de projeto resume-se na escolha de uma das duas opções principais (MARCOTTY; LEDGARD, 1986).

Na implementação por compilação, o programa escrito na linguagem de programação, denominado programa fonte, é traduzido em um programa equivalente, denominado programa objeto, escrito na linguagem de máquina do computador no qual ele será executado ou interpretado, no caso a denominada máquina alvo (MARCOTTY; LEDGARD, 1986).

O uso de um compilador, portanto, faz com que a execução de um programa escrito em uma determinada linguagem de programação seja um processo de duas fases. Pode-se observar na figura 10 a relação entre estas duas fases.

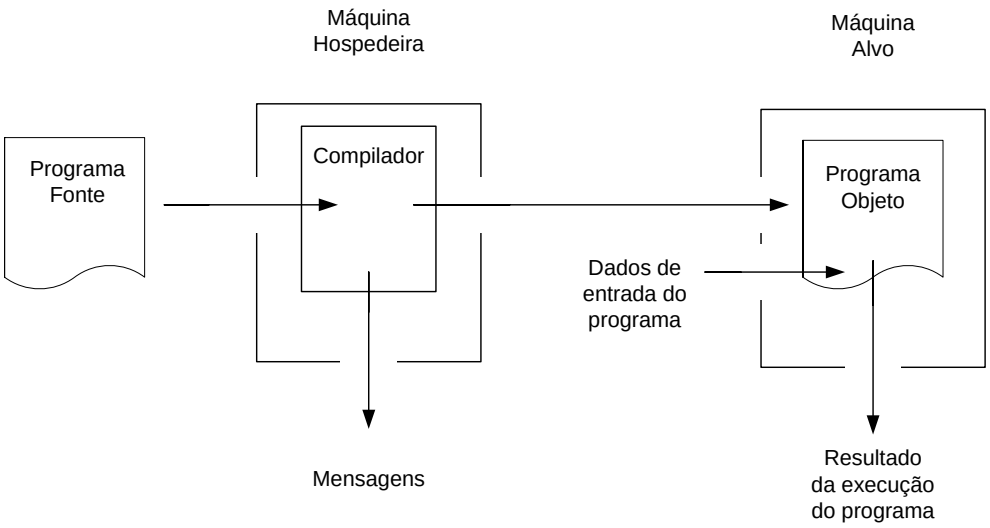

Figura 10 - Processo de compilação

Implementando uma linguagem de programação por interpretação, torna desnecessária a tradução do programa fonte. Neste tipo de implementação, as instruções são executadas diretamente através de um programa que é executado na máquina alvo. Conforme cada instrução do programa fonte é analisada, o interpretador executa as operações específicas associadas a ela, normalmente através da chamada de sub-rotinas (MARCOTTY; LEDGARD, 1986).

[9] Java é uma marca registrada de Sun Microsystems, Inc.

Desta forma, diferentemente do processo de compilação, a interpretação é um processo de uma única fase, como pode ser observado na figura 11.

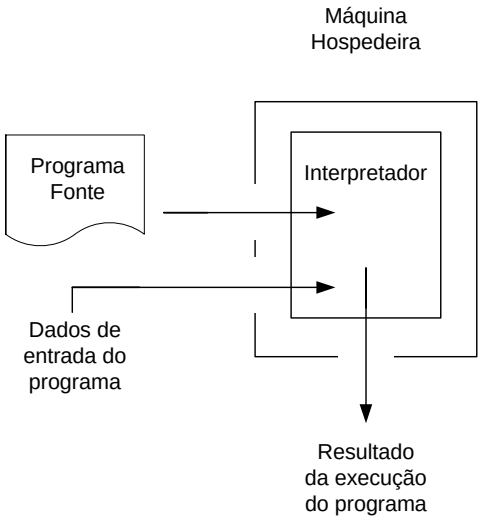

Figura 11 - Processo de Interpretação

O uso de um interpretador permite uma maior flexibilidade em relação à execução direta do programa. No entanto, a penalidade é o desempenho: a interpretação é geralmente muito mais lenta em comparação a execução direta (MARCOTTY; LEDGARD, 1986).

Na implementação da linguagem de entrada da ferramenta, conforme explicado na seção anterior, de nível mais elevado e paradigma imperativo, escolheu-se pelo projeto de um compilador, pois, devido à simplicidade das construções definidas na linguagem de programação, não alteraria o foco da pesquisa para outras áreas diferentes da tomada de decisão.

O projeto de um compilador integra um conjunto de funções, que pode ser simplificado em três partes primordiais: um analisador léxico, um analisador sintático e um gerador de código (NETO, 1987).

Uma biblioteca contendo as funções responsáveis pela construção das estruturas da tabela de decisão adaptativa gerada para a representação interna fez-se necessária. Esta biblioteca contém basicamente funções para a definição de condições, ações, regras e funções adaptativas, entre outras estruturas necessárias para a correta execução do programa objeto no ambiente de execução projetado.

A figura 12 exibe o relacionamento entre as partes do compilador e das estruturas geradas.

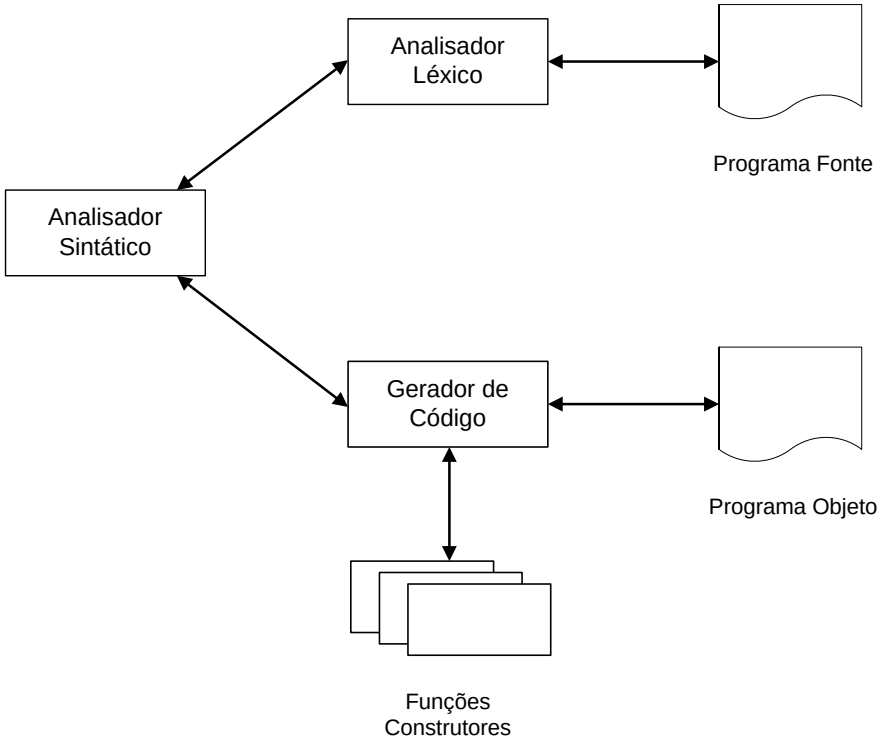

Figura 12 - Arquitetura do compilador

5.4 AMBIENTE DE EXECUÇÃO

Para o projeto do ambiente de execução, que representará o núcleo do ambiente de desenvolvimento proposto, optou-se pelo uso do dispositivo denominado tabela de decisão adaptativa, conforme explicitado no terceiro capítulo do presente trabalho.

Tabelas de decisão podem ser consideradas como descrições estruturadas de um procedimento. Duas outras formas comuns de descrição são a descrição narrativa e o fluxograma.

A descrição narrativa apresenta a vantagem de, se o escritor for habilidoso, fornecer maior detalhamento. Porém, se quisermos analisar o procedimento como

um todo, o seu fluxo ou a sua estrutura, ela se torna extremamente ineficiente (MONTALBANO, 1974).

O fluxograma é uma descrição que usa linhas e figuras geométricas para ilustrar o fluxo do processo, como o próprio nome diz. Embora o fluxograma facilite muito o processo de rastreamento das seqüências, condições e ações do procedimento, ele é ineficiente quando se quer obter uma visão do procedimento como um todo (MONTALBANO, 1974), (NAGAYAMA, 1990).

A tabela de decisão mostra resultados satisfatórios em ambos os casos, fornecendo uma imagem completa do processo, obtida pelo conjunto das regras propriamente ditas e ainda assim servindo como uma forma interessante de detalhamento do processo (MONTALBANO, 1974). Com a extensão da tabela de decisão pela tecnologia adaptativa, já explicitada no terceiro capítulo do presente trabalho, aproveita-se toda a capacidade ilustrada do dispositivo subjacente com a flexibilidade proporcionada pela tecnologia adaptativa na representação interna do programa na ferramenta proposta.

Com a escolha da tabela de decisão adaptativa como dispositivo formal na representação do programa internamente, o projeto do ambiente de execução encarregou-se do projeto de funções responsáveis pela manipulação da tabela de decisão e suas estruturas internas, como regras, condições e ações, a execução do algoritmo de funcionamento, de acordo com as suas especificações (NETO, 2001), a execução de algumas rotinas básicas de consistência; além do conjunto de funções responsável pela definição e execução das funções e ações adaptativas que concernem ao dispositivo.

Pode-se observar na figura 13 a arquitetura interna, em uma visão macroscópica, do ambiente de execução proposto para a ferramenta.

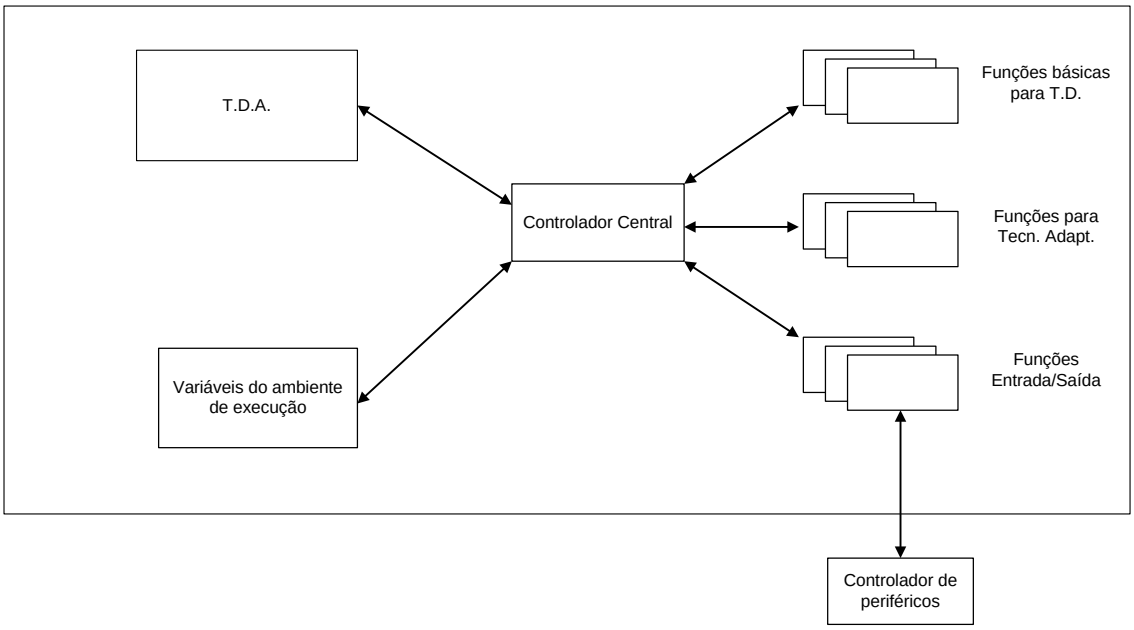

Figura 13 - Arquitetura do ambiente de execução

Com a exposição do projeto do ambiente de execução, finaliza-se o presente capítulo, cujo objetivo foi a apresentação da especificação dos principais componentes do ambiente proposto.

Dando seqüência à segunda parte do presente trabalho, o próximo capitulo se encarregará da apresentação dos aspectos de implementação mais relevantes do presente trabalho, juntamente com a parte experimental da pesquisa.

6. ASPECTOS DE IMPLEMENTAÇÃO

O presente capítulo é responsável pela finalização da segunda parte deste trabalho, sendo responsável pela apresentação dos aspectos de implementação da ferramenta proposta, de acordo com as especificações construídas no quinto capítulo.

Este capítulo também é responsável pela apresentação da parte prática da pesquisa desenvolvida, e está dividido em três seções, com as seguintes atribuições: uma seção é responsável pela apresentação da escolha da linguagem de programação e do ambiente de programação utilizados na implementação da ferramenta; uma segunda seção é responsável pela apresentação dos aspectos da construção de cada um dos principais componentes da ferramenta proposta; uma terceira seção fica encarregada da apresentação dos experimentos realizados e também é responsável pelos comentários a respeito da parte experimental da pesquisa.

Um aspecto a ser levado em consideração é o fato da ferramenta desenvolvida não possuir necessariamente um desempenho elevado, visto que se trata de um protótipo com o intuito de demonstrar a viabilidade do uso da tecnologia adaptativa em mais um campo do conhecimento humano. Apesar disso, faz-se necessário que se mostre a viabilidade da sua construção, de acordo com a arquitetura proposta, e que possa ser utilizada como referência na produção de uma ferramenta de escala comercial.

6.1 LINGUAGEM E AMBIENTE DE PROGRAMAÇÃO

Embora o protótipo da ferramenta proposta pudesse, em teoria, ser implementada utilizando qualquer linguagem de programação disponível, como

Java[10], C++ e Pascal, a escolha final foi utilizar uma linguagem de programação bastante difundida na comunidade científica da área de Inteligência Artificial, que é a linguagem LISP.

No entanto, a utilização de uma linguagem de paradigma funcional acabou por tornar-se uma premissa da pesquisa desenvolvida, de forma a fornecer um novo objetivo, por assim dizer, que seria a análise da implementação de dispositivos adaptativos em linguagem funcional. Premissa esta que é reforçada pela característica das linguagens funcionais de simplicidade na paralelização (SEBESTA, 1996), já que o objetivo desta pesquisa é apoiar a tomada de decisão quando há condição de risco e alta complexidade, fato facilitado por esta característica na análise de possíveis soluções não-determinísticas.

Esta premissa surgiu após a avaliação da proposta apresentada em Rocha e Neto (2003), sobre a proximidade do processo de avaliação de estruturas em linguagens de programação funcionais e o processo de avaliação de estruturas em dispositivos adaptativos.

Após uma breve reflexão sobre ambas as tecnologias, pode-se aproximar, o processo de avaliação das estruturas adaptativas como um processo de avaliação simbólica, no qual os estímulos de entrada do dispositivo seriam os símbolos a serem processados e no qual os passos de computação são as aplicações das funções (adaptativas ou não) do dispositivo à sua atual configuração.

Como explicado no quarto capítulo do presente trabalho, optou-se especificamente pela utilização do dialeto LISP denominado Scheme.

6.1.1 Ambiente para implementação

Após a escolha da linguagem de programação, seguiu-se a escolha do ambiente adequado, que integrasse as ferramentas de codificação, teste e documentação do projeto. Existem diversas opções, comerciais ou livres, para cada tipo de sistema e equipamento que se pretenda utilizar.

Como a opção para a plataforma de hardware deve seguir a linha dos computadores pessoais, pois é a tecnologia mais difundida nos ambientes

[10] Java é uma marca registrada de Sun Microsystems, Inc.

corporativos, a escolha do sistema operacional seguiu a mesma tendência, optando-se pelo sistema gráfico mais difundido atualmente, que é a plataforma Microsoft Windows[11]. Embora o ambiente para desenvolvimento faça uso da plataforma Microsoft Windows, a ferramenta originada pode ser utilizada em outras plataformas como, por exemplo, Linux.

Uma outra possibilidade seria a escolha de ambientes distintos, como por exemplo, uma ferramenta de análise e projeto, fazendo uso de uma outra ferramenta para a codificação. Porém, como o objetivo da implementação é desenvolver um protótipo, optou-se por um único ambiente de programação, que permita gerar programas de forma rápida e que possua capacidade gráfica.

A linguagem de programação Scheme foi projetada e implementada, em sua primeira versão, no laboratório de inteligência artificial do MIT (MIT AI Lab). Diversas universidades ao redor do mundo utilizam-na no ensino de conceitos de programação, e mesmo de conceitos relativos a projetos de linguagem de programação e engenharia de software (FINDLER et al, 2002).

A crescente disseminação da linguagem Scheme ocasionou posteriormente a publicação de uma norma IEEE (IEEE 1990) que especifica as construções, a maneira que a avaliação das instruções deve ser executada e a semântica das instruções.

Esta disseminação da linguagem Scheme também criou a necessidade do desenvolvimento de um ambiente mais amigável que o disponibilizado inicialmente pelo MIT, o qual oferecia alguns recursos básicos, mas não era interessante para o ensino de alunos ingressantes nas universidades. Além disso, também não supria as necessidades de ferramentas que facilitassem o processo de desenvolvimento de sistemas mais complexos (FINDLER et al, 2002).

O ambiente de programação DrScheme foi desenvolvido por um grupo de pesquisadores da Rice University (FINDLER et al, 2002) como uma tentativa de suprir a lacuna existente pela falta de um ambiente de desenvolvimento mais robusto. Este ambiente de programação apresenta uma interface gráfica altamente intuitiva e amigável, oferece a integração dos ambientes de desenvolvimento e execução dos programas e uma funcionalidade que aponta os erros de sintaxe com precisão de palavras. Devido à natureza da sintaxe da linguagem de programação

[11] Microsoft Windows e Windows são marcas registradas de Microsoft Corporation.

Scheme ser altamente permissiva[12], estes recursos tornam-se quase que essenciais.

Além de alguns recursos encontrados em ambientes de desenvolvimento comerciais mais sofisticados, como os algoritmos de tempo real para a classificação da sintaxe do programa em cores diferentes de acordo com a sua categoria sintática, o DrScheme oferece algumas funcionalidades muito interessantes, como a avaliação de expressões passo-a-passo (FINDLER et al, 2002).

Este ambiente de programação também permite o desenvolvimento de aplicações em uma implementação de Scheme que conta com uma extensão para a manipulação de objetos gráficos: a MrEd (FLATT et al, 1999). Para isso, a MrEd provê um conjunto de construções novas para a linguagem, de forma a permitir a definição de formulários, botões e campos de edição. Maiores detalhes sobre esta extensão da linguagem Scheme podem ser encontrados em Flatt et al, (1999).

Dentre os ambientes de programação disponíveis gratuitamente atualmente, o DrScheme apresentou o melhor conjunto de ferramentas para o auxílio ao processo de desenvolvimento e portanto, foi escolhido para ser utilizado na implementação desta proposta.

6.2 CONSTRUÇÃO DA FERRAMENTA

Com o intuito de manter o alto nível de organização e padronização na construção da ferramenta, diversas referências de apoio foram utilizadas. Algumas das referências mais importantes foram Pressman (2005) e Sommerville (2003) na área de engenharia de software e Winston e Horn (1989), Norvig (1992), Dybvig (1987) e Eisenberg e Abelson (1990) com relação a linguagens LISP em geral, e no dialeto Scheme, especificamente.

Nas áreas de projeto de linguagens de programação, as referências de apoio mais utilizadas foram Sebesta (1996), Gordon (1988), Gordon (1979) e Mitchell (1996); com relação à área de projeto de compiladores e interpretadores, as

[12] Permissiva no sentido que suas construções possuem alto grau de ortogonalidade. Desta forma, embora o avaliador e/ou o compilador não acuse nenhum erro de sintaxe, a semântica não é necessariamente a desejada pelo desenvolvedor da aplicação.

referências Neto (1987), Neto (1993) e Aho; Sethi e Ullman (1986) foram utilizadas com mais freqüência, dentre outros títulos de diferentes áreas de conhecimento.

Novamente, como a ferramenta desenvolvida é um protótipo, não foi foco da implementação fazer uso de técnicas ou métodos para o aumento de desempenho que, embora muito vantajosos para projetos de escala industrial, mostram-se inadequados para o projeto e implementação do ambiente proposto.

O desenvolvimento seguiu o padrão apresentado em Abelson; Sussman, G. e Sussman, J. (1996), denominado barreiras de abstração. Este conceito é muito bem conhecido por usuários da tecnologia orientada a objetos, na qual é denominado de encapsulamento (PRESSMAN, 2005).

Este processo consiste em construir barreiras de abstração na maneira de manipular os dados da aplicação, de forma que os detalhes de implementação dos níveis mais baixos não influenciem o desenvolvimento dos níveis mais altos da aplicação. Pode-se observar na figura 14 um esquema que ilustra bem este processo.

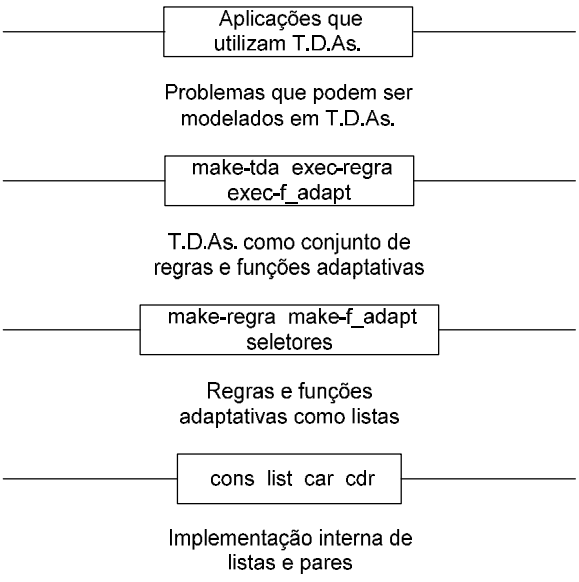

Figura 14 - Barreiras de abstração de dados

O uso deste processo apresenta diversas vantagens. A manutenção dos programas torna-se muito mais fácil, pois com o controle da complexidade na manipulação das estruturas encapsulada em uma barreira de abstração, concentra-

se os esforços na construção de módulos que façam uso destas ‘interfaces’ entre as camadas da aplicação.

6.2.1 Análise do dispositivo

De acordo com as especificações da ferramenta proposta e de seus módulos principais apresentados no quinto capitulo do presente trabalho, executou-se uma análise à luz dos conceitos disponíveis na tecnologia e ambiente escolhidos para a implementação. As principais estruturas necessárias, de acordo com os seus respectivos módulos são os seguintes:

Linguagem de entrada

A especificação completa da linguagem de entrada foi fornecida no quinto capítulo do presente trabalho. Os aspectos de implementação da linguagem de entrada são tratados pelo módulo compilador, descrito a seguir.

Compilador

- Analisador léxico;
- Analisador sintático;
- Gerador de código.

Ambiente de execução

- Interpretador de tabela de decisão adaptativa;
- Ambiente para o armazenamento de variáveis.

6.2.2 Implementação do módulo compilador

Esta seção encarrega-se da ilustração de alguns dos aspectos mais importantes da implementação do módulo compilador. A construção de um compilador completo não se limita ao exposto a seguir de forma alguma. Detalhes da

implementação do analisador léxico, por exemplo, devido à sua relativa simplicidade e padronização de construção, não foram citados.

O processo de construção de um compilador deve ser iniciado somente após o término do projeto formal da linguagem de programação fonte. Deste modo, após a conclusão do processo de formalização da gramática, em qualquer que seja o formato utilizado, há a possibilidade de se obter um reconhecedor para esta linguagem de modo automático de diversas formas (NETO, 1987), (AHO; SETHI; ULLMAN, 1986).

Será utilizada nas seções a seguir uma nomenclatura muito especifica das áreas de linguagens formais, projeto de linguagens de programação e projeto de compiladores. Alguns termos provavelmente não são familiares para o leitor de outras áreas de conhecimento e, portanto, a consulta a referências destas áreas pode se fazer necessária. As indicações, dentre as diversas disponíveis na literatura especializada são Neto (1987), Aho; Sethi e Ullman (1986) e Hopcroft; Motwani e Ullman (2001).

Segundo a técnica descrita em Neto (1987) para o projeto e construção de compiladores, as principais etapas para a obtenção de um compilador completo e funcional são as seguintes:

1. Conversão da especificação da linguagem, de quaisquer outros formatos para a notação de Wirth (NETO, 1987);
2. Agrupamento das regras de produção, de forma a existir apenas uma regra de produção com cada não-terminal gerador;
3. Definição dos não-terminais essenciais, os denominados não-terminais auto-recursivos centrais, através da montagem e análise de uma árvore da gramática;
4. Eliminação de não-terminais que não sejam essenciais, através do método de substituições sucessivas dos mesmos pelas expressões que os definem, de forma a obter uma expressão composta apenas de terminais e não-terminais essenciais;
5. Atribuição de estados do reconhecedor, baseado na expressão obtida na etapa anterior (NETO, 1987);

6. Obtenção do dispositivo reconhecedor necessário. No caso de tratar-se de linguagens livres de contexto, o dispositivo reconhecedor gerado ao final do processo é um autômato de pilha estruturado.
7. Simplificação do autômato de pilha estruturado resultante da etapa anterior.

Ao final deste processo, um reconhecedor para a linguagem estará disponível. Este dispositivo será o reconhecedor sintático da linguagem, e será utilizado como esqueleto do compilador. A arquitetura do compilador construído será a de utilizar o analisador sintático como componente central, conforme pode ser observado na figura 15 (NETO, 1987).

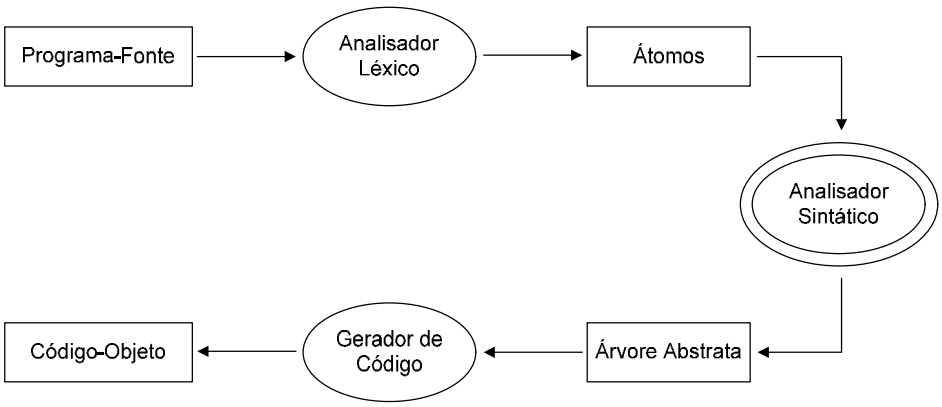

Figura 15 - Arquitetura do compilador

Neste esquema, o analisador sintático é o programa principal e o analisador léxico, bem como o gerador de código, são compostos por funções referenciadas pelo analisador sintático conforme as suas necessidades, conforme ilustrado na figura 15.

A esta arquitetura de compilador, devido ao fato de que a estrutura sintática do programa-fonte determina a seqüência das atividades de compilação, dá-se o nome de compilação dirigida pela sintaxe, e corresponde ao esquema mais popular de organização de compiladores (NETO, 1987), (LOUDEN, 2004).

Para finalizar a construção do compilador, há a necessidade de associar as chamadas às rotinas geradoras de código ao autômato de pilha estruturado gerado.

Devido às características do ambiente de execução, a linguagem de saída do compilador resume-se basicamente a produzir chamadas de rotinas responsáveis pela definição da tabela de decisão adaptativa a ser executada e da manipulação das variáveis de ambiente declaradas.

Maiores detalhes sobre a especificação das funções internas do compilador como um todo e sobre o processo completo de construção de um compilador podem ser encontrados em Neto (1987).

6.2.3 Implementação do ambiente de execução

Conforme a descrição das partes componentes apresentadas anteriormente no presente capítulo, o módulo do ambiente de execução é composto, primordialmente, por três partes: uma estrutura de dados para o armazenamento das variáveis do ambiente de execução, uma estrutura de dados para o armazenamento da tabela de decisão adaptativa e um conjunto de funções para a manipulação destas duas estruturas.

Por fazer uso da linguagem de programação Scheme como meio de implementação da ferramenta proposta, as estruturas de dados responsáveis pela representação interna são todas da forma lista ligada (ABELSON; SUSSMAN, G; SUSSMAN, J, 1996), que é a estrutura de dados básica disponível em Scheme.

A estrutura de dados utilizada para o armazenamento das variáveis do ambiente de execução da ferramenta proposta é representada por uma lista de pares de células. A primeira célula de cada par representa o nome da variável em sua declaração e a segunda célula representa o seu valor atribuído, conforme pode ser observado na figura 16. Em caso da variável haver sido declarada, mas não houver ocorrido o processo de atribuição de valor à variável, a mesma apresentará uma situação inválida, impossibilitando o seu uso em rotinas de manipulação. Na figura 16, a variável QTD_MAT_A apresenta situação inválida. Se esta variável for utilizada na definição de alguma expressão ou comparação, um erro será gerado, indicando que não há nenhum valor atribuído a ela.

Como não há o conceito de múltiplos escopos na ferramenta proposta, não houve a necessidade de definir-se uma estrutura capaz de executar o controle da visibilidade das variáveis, embora não haja muitas complicações em fazê-lo se necessárias, conforme se pode observar em Abelson; Sussman, G. e Sussman, J. (1996).

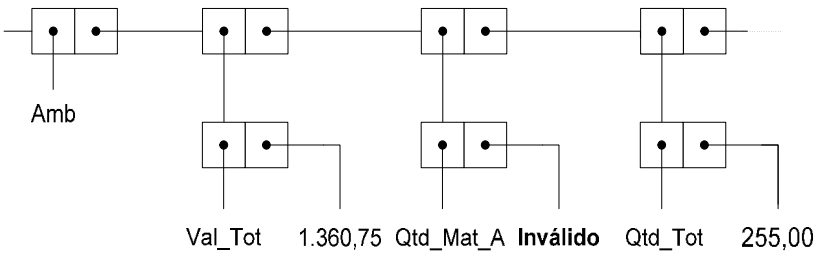

Figura 16 - Estrutura de armazenamento das variáveis

Na definição da representação interna da tabela de decisão adaptativa, observam-se duas partes distintas: o conjunto de regras da tabela de decisão adaptativa, que são responsáveis pelo comportamento do dispositivo, e o conjunto das funções adaptativas referenciadas pelas regras. Desta forma, na visão mais macroscópica, representa-se a tabela de decisão adaptativa como:

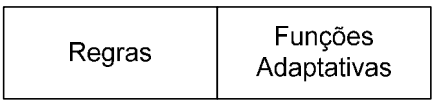

Figura 17 - Visão macro da Tabela de Decisão Adaptativa

A estrutura de dados de cada uma das regras que compõem a primeira parte da tabela de decisão adaptativa é representada por uma lista composta de quatro partes:

- Ação Adaptativa Anterior;
- Conjunto de condições da regra;

- Conjunto de ações da regra;
- Ação Adaptativa Posterior.

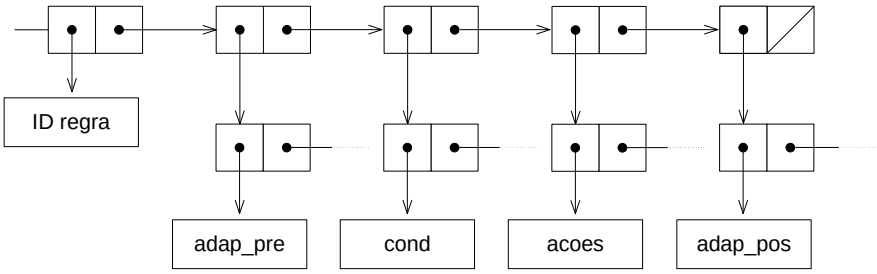

Figura 18 - Estrutura interna das regras de uma Tabela de Decisão Adaptativa

As ações adaptativas anterior e posterior são representadas por uma lista com dois componentes: o nome da função adaptativa a ser acionada e uma lista de pares, com a primeira célula de cada par representando o nome de um parâmetro declarado na função acionada, e a segunda célula representando o valor atribuído àquele parâmetro.

O conjunto de condições da regra é representado por uma lista de pares, com a primeira célula de cada par representando uma variável declarada no ambiente, e a segunda célula representando a condição a ser testada. Os operadores de comparação disponibilizados para a construção das condições nesta versão protótipo da ferramenta são somente os operadores básicos para maior (>), menor (<), igual (=) e diferente (!=).

O conjunto de ações da regra também é representado por uma lista de pares, com a primeira célula de cada par representando uma ação e a segunda célula representando um dado necessário à realização da ação. O conjunto de ações disponibilizados nesta versão protótipo da ferramenta é reduzido, possuindo funções básicas para a manipulação das variáveis declaradas e comandos de entrada e saída de dados, como pode ser observado no capítulo cinco do presente trabalho.

Cada uma das funções adaptativas que compõe a segunda parte de uma tabela de decisão adaptativa é representada por uma lista com cinco componentes principais:

- Nome da função;
- Conjunto de objetos (parâmetros, variáveis e geradores);
- Conjunto de padrões para consulta;
- Conjunto de padrões para exclusão;
- Conjunto de padrões para inclusão.

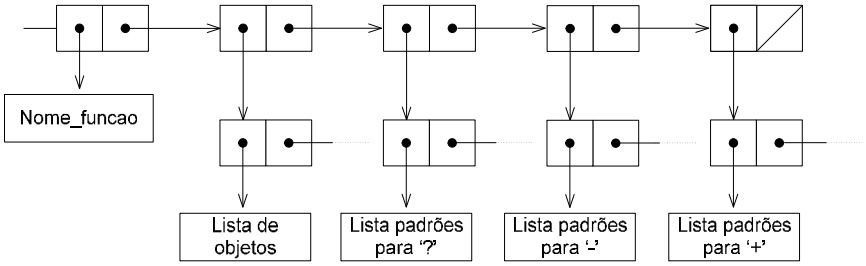

Figura 19 - Estrutura interna das funções adaptativas de uma Tabela de Decisão Adaptativa

A representação do nome da função adaptativa é feita por um átomo, sendo o primeiro componente da estrutura.

O conjunto de elementos da função adaptativa é representado por uma lista de pares. Cada par representa um elemento, com a primeira célula do par representando o nome do elemento e a segunda célula do par representando o tipo do elemento. Os tipos possíveis são o parâmetro, identificado pela letra P; a variável, identificado pela letra V e o gerador, identificado pela letra G. No caso especifico dos geradores, uma terceira célula, responsável pelo armazenamento do valor do gerador em sua ultima instanciação fez-se necessária.

Os conjuntos de padrão de consulta, exclusão e inclusão são representados por listas que seguem a estrutura de uma regra, conforme definição acima. Possuem, desta forma, os mesmos componentes.

O conjunto de funções utilizado para a manipulação das estruturas componentes da tabela de decisão adaptativa e das variáveis do ambiente de execução foi dividido em três partes, de acordo com as suas funcionalidades, conforme capítulo cinco do presente trabalho. São elas: as funções básicas para operação de uma tabela de decisão; as funções responsáveis pela definição,

manipulação e execução das funções e adaptativas; e as funções responsáveis pelas operações de entrada e saída de dados da ferramenta.

As funções responsáveis pela operação da tabela de decisão executam as funcionalidades básicas do dispositivo, como avaliação das regras em busca de alguma que possa ser aplicada no ciclo corrente de execução do dispositivo, execução das ações das regras selecionadas e controles dos ciclos de execução.

O conjunto das funções responsáveis pela camada adaptativa da tabela de decisão adaptativa é responsável pela execução das ações adaptativas sobre o dispositivo e alterações do conjunto de regras da tabela de decisão adaptativa.

As funções responsáveis pelas operações de entrada e saída de dados da ferramenta permitem a entrada e saída de dados da ferramenta proposta, cuidando da interação com os periféricos como, por exemplo, monitor, teclado, mouse e dispositivos de armazenamento secundário.

O funcionamento do programa internamente nas estruturas do ambiente de execução se dá da maneira descrita a seguir:

1. Após o armazenamento das variáveis nas suas respectivas estruturas, executa-se, conforme descrito anteriormente, a busca de regras aderentes às condições descritas pelos valores das variáveis.
2. Não havendo nenhuma regra aderente às condições, a execução do programa é finalizada.
3. Havendo apenas uma regra compatível com as condições, a mesma é selecionada para execução.
4. Havendo mais de uma regra compatível com as condições descritas pelas variáveis, obtém-se uma situação de não-determinismo. Neste caso, todas as regras compatíveis são selecionadas para execução. Essa execução será executada uma a uma, seguindo o processo de *backtracking*.
5. A regra selecionada é executada, primeiramente pela execução da ação adaptativa anterior, se esta existir. Segue-se pela execução das ações codificadas no corpo da regra, e finaliza-se com a execução da ação adaptativa posterior, se esta existir. A execução da regra segue o modelo descrito no terceiro capítulo do presente trabalho.

6. Terminado este ‘ciclo de execução’, inicia-se novamente a busca de regras aderentes às condições do ambiente.

A execução das ações disponibilizadas (atribuição de valor à variável, impressão de dados e leitura de dados) é executada conforme descrito no quinto capítulo do presente trabalho.

A instrução de atribuição atribui o valor passado como parâmetro à variável solicitada. O mesmo ocorre com a instrução de leitura de dados, com a diferença que o valor passado como parâmetro é fornecido pelo usuário do programa. Internamente, a estrutura de armazenamento de variáveis é procurada em busca da variável a ser alterada, e o seu correspondente valor alterado.

A instrução de impressão de dados também é simples: executa a saída de dados através do dispositivo padrão de saída, que pode ser monitor de vídeo ou um arquivo de armazenamento, entre outras opções.

Com a conclusão dos aspectos de implementação do ambiente de execução finaliza-se a seção corrente, responsável pelos aspectos de implementação da ferramenta e seus componentes. A próxima seção do presente capítulo é responsável pela apresentação da parte prática do trabalho, a ser detalhada a seguir.

6.3 PARTE EXPERIMENTAL

Esta seção é responsável pela apresentação da parte experimental (prática) da pesquisa. Com o intuito de mostrar a flexibilidade da ferramenta proposta, três diferentes experimentos foram conduzidos, com níveis de complexidade variados.

O primeiro experimento, considerado o mais simples dos três, representa internamente uma tabela de decisão comum. Com este exemplo, procura-se mostrar a aderência da ferramenta ao dispositivo subjacente clássico, ou seja, não-adaptativo.

O segundo experimento, um pouco mais elaborado, apresenta uma versão estendida do primeiro experimento. Neste exemplo, à lógica da tabela de decisão subjacente foi adicionada uma camada adaptativa, de forma a criar um fluxo de informações e incorporar as alterações ao modelo, com o intuito de estender a utilização do modelo sem a necessidade de interferência por parte do usuário.

No terceiro experimento, será apresentado outro exemplo de uso da tabela de decisão adaptativa aplicada à tomada de decisão, em um exemplo um pouco mais elaborado que o apresentado no exemplo anterior.

Ao final da seção são apresentados breves comentários sobre a parte experimental do trabalho.

6.3.1 Experimento um: tabela de decisão

O problema utilizado no primeiro exemplo é o de um processo de tomada de decisão relativo à aquisição de material necessário a uma obra do ramo de construção civil (PEDRAZZI, TCHEMRA, ROCHA; 2005). As condições ideais para a aquisição são estabelecidas, como o custo do material ser abaixo de um limite pré-estabelecido (custo < limite_custo = S) e o fornecedor se localizar relativamente próximo à obra destino do material específico (proximidade = S). Respectivamente, as condições custo > limite_custo = N, ou proximidade = N indicam condições não-ideais de aquisição do material.

No processo de escolha da regra, uma saída é gerada, de forma que o Código do Material, Obra destino do material e o Fornecedor do material sejam impressos em dispositivo periférico de saída, como o monitor de vídeo ou uma impressora.

Os campos da tabela de decisão utilizada para representar a problemática são os seguintes:

- Material: código do material a ser adquirido, campo de valor numérico;
- Obra: código da obra destino do material, campo de valor numérico;
- Fornecedor: código do fornecedor do material, campo de valor numérico;

- Custo < Limite_custo: condição de aquisição do material, relativa ao custo do mesmo;
- Proximidade: condição de aquisição do material, relativa à proximidade do fornecedor da obra destino;
- Imprimir (regra): Ação responsável pela impressão dos dados do material, obra e fornecedor, em caso de decisão afirmativa.

A figura a seguir (figura 20) ilustra a tabela de decisão que representa as condições de determinada situação do processo de tomada de decisão.

material=	128	128	128	903	903	32
obra=	15	18	18	15	18	33
fornecedor=	589	589	101	589	101	101
custo<limite_custo=	S	S	N	S	S	N
proximidade=	S	N	N	S	S	S
imprimir (regra)	✓			✓	✓	

Figura 20 - Exemplo um: Tabela de Decisão

A seguir o programa escrito na linguagem de entrada da ferramenta proposta é mostrado, de forma a representar as mesmas regras de decisão da tabela de decisão ilustrada, na figura 21. Nota-se que a porção relativa à declaração das funções adaptativas no código fonte é inexistente.

```
Tda
Numero: material, obra, fornecedor;
Caract: preço<limite_preço, proximidade;
Inicio
SE (material=128, obra=15, fornecedor=589,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
SE (material=128, obra=18, fornecedor=589,
preço<limite_preço='S', proximidade='N') ENTAO ();
SE (material=128, obra=18, fornecedor=101,
preço<limite_preço='N', proximidade='N') ENTAO ();
SE (material=903, obra=15, fornecedor=589,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
```

```
SE (material=903, obra=18, fornecedor=101,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
SE (material=32, obra=33, fornecedor=101,
preço<limite_preço='N', proximidade='S') ENTAO ();
Fim;
```

Figura 21 - Exemplo um: Programa escrito na linguagem de entrada

6.3.2 Experimento dois: tabela de decisão adaptativa

O segundo experimento, também baseado no exemplo apresentado em Pedrazzi; Tchemra e Rocha (2005), realiza a extensão da tabela de decisão convencional, utilizada no exemplo apresentado na seção 6.3.1, por uma função adaptativa, responsável pela manutenção das regras da tabela de decisão dita subjacente, de forma a reduzir o número de regras da mesma, melhorando o processo de busca e consequentemente o seu desempenho.

Tratando da mesma função adaptativa, ela é responsável, através de chamadas a funções externas, como a de cotação ou análise de custos, pela inclusão de uma regra que atenda aos requisitos de aquisição de material. Outra função externa, que poderia basear-se na tecnologia GPS[13], seria o de ajuste de proximidade relativa entre o fornecedor do material e a obra destino do material desejado.

A codificação e o domínio dos campos da tabela de decisão subjacente são os mesmos do experimento apresentado na seção 6.3.1. As diferenças ficam por conta da camada adaptativa, representada na figura 22 pela parte esquerda da tabela de decisão adaptativa, e pelas ações adaptativas adicionadas às regras da tabela de decisão convencional, representada pela parte inferior direita na figura 22.

[13] Global Positioning System, ou Sistema de Posicionamento Global é um sistema que funciona por uma rede de satélites. Utilizado originalmente para navegação, possui atualmente diversas aplicações comerciais.

		Tag →	H	-	-	-	+	S	R	R	R	R	R	R	R	E
Tabela de Decisão Subjacente	Condições	material=		p1	p1	p1	p1		"μ"	128	128	128	903	903	32	
		obra=		p2	p2	p2	p2		"o"	15	18	18	15	18	33	
		fornecedor=					g1			589	589	101	589	101	101	
		custo<limite_custo=		N	N	S	S			S	S	N	S	S	N	
		proximidade=		S	N	N	S			S	N	N	S	S	S	
	Ações	leitura (dados)						✓								
		imprimir (regra)					✓			✓			✓	✓		
Funções Adapt.	Fun.	*F*	B						✓							
	Obj.	*p1*	P						"μ"							
		p2	P						"o"							
		g1	G													

Figura 22 - Exemplo dois: Tabela de Decisão Adaptativa

A seguir é mostrado o programa escrito na linguagem de entrada da ferramenta proposta. Neste experimento, diferentemente do apresentado na seção 6.3.1, a porção do código fonte relativa à declaração das funções adaptativas contém a codificação da função F, responsável pelo comportamento descrito no início da presente seção.

```
Tda
Numero: material, obra, fornecedor;
Caract: preço<limite_preço, proximidade;
Funcao F
(Parametros: p1, p2;
 Geradores: g1;)
Inicio-adaptativa
?:
-:
SE (material=p1, obra=p2, preço<limite_preço='N',
proximidade='S') ENTAO ();
SE (material=p1, obra=p2, preço<limite_preço='N',
proximidade='N') ENTAO ();
SE (material=p1, obra=p2, preço<limite_preço='S',
proximidade='N') ENTAO ();
+:
SE (material=p1, obra=p2, fornecedor=g1,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
Fim-adaptativa;
Inicio
SE (material="μ", obra="o") ENTAO F(material "μ", obra
"o")();
```

```
SE (material=128, obra=15, fornecedor=589,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
SE (material=128, obra=18, fornecedor=589,
preço<limite_preço='S', proximidade='N') ENTAO ();
SE (material=128, obra=18, fornecedor=101,
preço<limite_preço='N', proximidade='N') ENTAO ();
SE (material=903, obra=15, fornecedor=589,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
SE (material=903, obra=18, fornecedor=101,
preço<limite_preço='S', proximidade='S') ENTAO (Imprimir
regra);
SE (material=32, obra=33, fornecedor=101,
preço<limite_preço='N', proximidade='S') ENTAO ();
Fim;
```

Figura 23 - Exemplo dois: Programa escrito na linguagem de entrada

6.3.3 Experimento três: tabela de decisão adaptativa

No terceiro experimento, será apresentado um problema de tomada de decisão relativo a programação de utilização de recursos em uma fábrica. Algumas condições iniciais de tomada de decisão foram programadas na tabela de decisão subjacente. A camada adaptativa disponibilizada neste modelo é responsável pela consulta às regras e pela chamada ao processo de inferência em relação aos dados já disponíveis na tabela. Neste exemplo, o processo de inferência é representado por uma chamada a uma função externa, com o intuito de simplificar o mesmo, sem de forma alguma prejudicar a demonstração das capacidades da ferramenta.

Para facilitar a modelagem do problema, foi pressuposto que exista um conjunto de funções capaz de classificar os dados discretos dentre as categorias apresentadas a seguir. Este conjunto de funções é composto por definições, recebidas do usuário na forma de parâmetros de configuração, dos intervalos a serem considerados em cada uma das variáveis. Embora este tipo de função também possa ser codificado em uma tabela de decisão adaptativa, este processo de conversão de dados dificultaria a demonstração do exemplo, e conseqüentemente o seu entendimento.

As variáveis a serem analisadas são as seguintes:

- Demanda: representando a necessidade de produção, podendo ser classificada entre as categorias alta, média e baixa.
- Estoque: representando a quantidade do produto produzido estocado, podendo ser classificado entre as categorias alto, médio e baixo.
- Utilização do parque de máquinas: representando a intensidade de utilização do dos meios de produção disponíveis, podendo ser classificado entre normal e alta.

As ações possíveis, dependendo das condições de decisão, são a recomendação do aumento da capacidade do parque de máquinas e a recomendação do aumento da capacidade de estoque.

Pode ser observado na figura 24 a tabela de decisão representando a modelagem do problema.

		Tag →	H	?	?	?	+	S	R	R	R	R	R	R	R	E
Tabela de Decisão Subiacente	Cond.	Demanda=		p1		p1	p1		"α"	Baixa	Baixa	Média	Média	Média	Alta	
		Estoque=		p2	p2		p2		"β"	Baixo	Alto	Baixo	Médio	-	-	
		Ut. Maquinário=			p3	p3	p3		"χ"	Normal	Normal	Alta	Normal	Alta	Alta	
	Ações	Leitura (dados)						✓								
		Aum. Maquinário		v1	v3	v5	v7*							✓	✓	
		Aum. Estoque		v2	v4	v6	v8*					✓				
Funções Adapt.	Fun.	*F*	B						✓							
	Elementos	*p1*	P						"α"							
		p2	P						"β"							
		p3	P						"χ"							
		v1	V													
		v2	V													
		v3	V													
		v4	V													
		v5	V													
		v6	V													
		*v7**	V													
		*v8**	V													

Figura 24 - Exemplo três: Tabela de Decisão Adaptativa

A seguir é mostrado o programa escrito na linguagem de entrada. Neste experimento, de maneira similar ao exemplo apresentado na seção 6.3.2, a porção do código fonte relativa à declaração das funções adaptativas contém a codificação da função F, responsável pele coleta de informações codificadas na tabela de decisão subjacente.

Como pode ser observado, as variáveis v7 e v8 estão marcadas com um asterisco. Esta diferenciação deve-se a que os valores destas variáveis são obtidos através de uma função de inferência externa, que recebe como parâmetros o conteúdo das variáveis v1, v2, v3, v4, v5 e v6. De posse destes dados, esta função retorna os valores correspondentes aos campos das ações “Aumentar capacidade do maquinário” e “Aumentar capacidade de estoque”, que são inseridas pela ação elementar de inserção da função adaptativa F.

```
Tda
Caract: Demanda, Estoque, Ut.Maquinario;
Funcao F
(Parametros: p1, p2;
 Variaveis: v1, v2, v3, v4, v5, v6, v7, v8;)
Inicio-adaptativa
?:
SE (Demanda=p1, Estoque=p2) ENTAO (Aum.Maquinario=v1,
Aum.Estoque=v2);
SE (Estoque=p2, Ut.Maquinario=p3) ENTAO
(Aum.Maquinario=v3, Aum.Estoque=v4);
SE (Demanda=p1, Ut.Maquinario=p3) ENTAO
(Aum.Maquinario=v5, Aum.Estoque=v6);
+:
SE (Demanda=p1, Estoque=p2, Ut.Maquinario=p3) ENTAO
(Aum.Maquinario=v7, Aum.Estoque=v8);
Fim-adaptativa;
Inicio
SE (Demanda=”α”, Estoque=”β”, Ut.Maquinario=”χ”) ENTAO
F(Demanda ”α”, Estoque ”β”, Ut.Maquinario ”χ”)();
SE (Demanda=’Baixa’, Estoque=’Baixo’,
Ut.Maquinario=’Normal’) ENTAO ();
SE (Demanda=’Baixa’, Estoque=’Alto’,
Ut.Maquinario=’Normal’) ENTAO ();
SE (Demanda=’Media’, Estoque=’Baixo’,
Ut.Maquinario=’Alta’) ENTAO (Aum.Estoque);
SE (Demanda=’Media’, Estoque=’Medio’,
Ut.Maquinario=’Normal’) ENTAO ();
```

```
SE (Demanda=’Media’, Ut.Maquinario=’Alta’) ENTAO
(Aum.Maquinario);
SE (Demanda=’Alta’, Ut.Maquinario=’Alta’) ENTAO
(Aum.Maquinario);
Fim;
```

Figura 25 - Exemplo três: Programa escrito na linguagem de entrada

6.3.4 Comentários sobre a parte prática

Os experimentos realizados e mostrados nas seções anteriores foram simples, porém cumprem o propósito de demonstrar a utilidade da ferramenta em descrever as condições da tomada de decisão coerentemente. A extrapolação do uso da ferramenta torna-se somente uma questão de escala, e não de funcionalidade, visto que o modelo foi exercitado em diversos aspectos.

A simplicidade dos exemplos deve-se também a introdução dos conceitos da tecnologia adaptativa de maneira prática, que é a alteração do conjunto de regras que descrevem o comportamento do dispositivo programado, que foi efetuada de maneira gradual.

Acredita-se que a linguagem de entrada do ambiente tenha cumprido o seu objetivo, que é o de facilitar a definição dos eventos que definem o comportamento do programa. Embora a execução do programa na linguagem de entrada não seja interpretada linearmente, conforme foi observado no quinto capítulo do presente trabalho, o conceito de funcionamento de uma tabela de decisão como um conjunto de eventos é de pleno conhecimento do público-alvo da ferramenta.

A ausência de conceitos muito rebuscados certamente facilita o uso da linguagem de entrada por parte do público-alvo principal da ferramenta, que não necessariamente conhece a fundo conceitos de computação.

As possíveis melhorias, variações e alterações da ferramenta proposta serão apresentadas no próximo capítulo.

7. CONSIDERAÇÕES FINAIS

O principal objetivo deste trabalho foi apresentar uma das possíveis aplicações da tecnologia adaptativa na questão do apoio à tomada de decisão em ambientes empresariais e nos Sistemas de Apoio à Decisão. Este objetivo foi atingido com o projeto e implementação de uma ferramenta de apoio a este processo de tomada de decisão.

O produto final deveria apresentar uma interface simples, de forma a não causar grandes problemas em sua absorção pelo público-alvo principal, que são decisores e administradores em geral. O dispositivo obtido deveria também apresentar comportamento similar aos modelos clássicos da área de inteligência artificial aplicados ao processo de tomada de decisão em organizações.

Desta forma, partindo da necessidade de um processo de tomada de decisão, passando pela tecnologia empregada para modelar esse processo e finalmente o seu método de implementação, obteve-se o levantamento das características da ferramenta proposta, seguida pelo seu projeto e finalmente a sua implementação.

Fazendo uso de diversas áreas da computação, como inteligência artificial, projeto de linguagens de programação e compiladores, para citar as principais, acredita-se que o produto final tenha atendido as expectativas iniciais do estudo.

Dentro dos objetivos traçados, e partindo de três grandes linhas de fundamentos do conhecimento humano: a área de tomada de decisão, a tecnologia adaptativa e os conceitos e aplicações de linguagens funcionais, a pesquisa foi completada apresentando um resultado bastante interessante, mostrando a plena capacidade da tecnologia adaptativa como solução viável para problemas nas mais diferentes áreas do conhecimento humano.

7.1 PRINCIPAIS CONTRIBUIÇÕES

Como contribuições deste trabalho, pode-se citar a definição e construção de tabelas de decisão adaptativas em linguagem funcional, com a estruturação da representação interna dos seus componentes e suas características de funcionamento, e especialmente a implementação de tais estruturas. Pode-se observar que a eficiência do modelo interno utilizado para a representação da ferramenta, a tabela de decisão adaptativa, pode ser vista também como uma contribuição.

Outra contribuição foi a aproximação do conceito de funções como objetos de primeira classe, e conseqüentemente a possibilidade da utilização de programas como dados, ao processo de avaliação de um dispositivo adaptativo, cujo dispositivo subjacente representa o parâmetro recebido por um conjunto de funções adaptativas, de forma que este conjunto retorna como resultado do seu processamento um novo dispositivo subjacente.

A definição de uma linguagem de entrada para a ferramenta, de forma que fosse possível a execução de tabelas de decisão adaptativas, ou seja, não houvesse dependência de uma estrutura de um Sistema de Apoio à Decisão, é outra contribuição deste trabalho.

Este tipo de linguagem, denominada como linguagem de programação de domínio específico, oferece diversas vantagens, como permitir que soluções aos problemas possam ser especificadas em termos de conceitos e no nível de abstração do domínio dos problemas. Conseqüentemente, profissionais que sejam especialistas neste domínio podem compreender, validar, modificar e mesmo desenvolver programas nestas linguagens com relativa facilidade.

Além disso, o uso de linguagens de programação de domínio específico provê ganhos de qualidade, produtividade, confiabilidade e reusabilidade (MERNIK; HEERING; SLOANE, 2005), pois permite que os programas escritos nestas linguagens não façam necessário o uso de muitas estruturas auxiliares para a representação da problemática.

7.2 COMENTÁRIOS CRÍTICOS

Esta seção tem por objetivo levantar os problemas encontrados no processo de desenvolvimento do trabalho, as falhas e as possíveis alterações visando à ampliação do escopo de utilização da ferramenta, e desta forma levantar esses pontos com o intuito de tentar encontrar formas de melhorar o trabalho realizado.

Quando o modelo interno gerado é composto por muitas regras de evento, o desempenho da execução do mesmo pode ser prejudicado, pois a busca de regras compatíveis para execução é linear. Um método de indexação das regras, como por exemplo, uma função *hash*, pode ser adicionada ao procedimento de armazenamento e busca das regras, de forma a reduzir o tempo de busca das mesmas (RAMAKRISHNAN; GEHRKE, 2002).

A implementação de uma interface mais amigável para a ferramenta final, embora o objetivo principal do trabalho fosse somente mostrar a viabilidade do uso da tecnologia adaptativa ao processo de tomada de decisão em ambientes empresariais.

Uma interface gráfica com recursos modernos, de forma a diminuir ou até mesmo suprimir a necessidade da escrita de código por parte do usuário da ferramenta, de modo a reduzir a dependência do conhecimento na própria ferramenta para o desenvolvimento de aplicações, é uma das possíveis melhorias da ferramenta. Pode-se fazer uso da extensão gráfica da linguagem Scheme disponibilizada na ferramenta DrScheme: MrEd (FINDLER et al, 2002). Esta extensão provê a funcionalidade de manipulação de objetos gráficos, como janelas e campos de edição, por exemplo. Ela também fornece um arcabouço de recursos adicionais, como a possibilidade da utilização de *threads* em suas construções.

Outra possível melhoria diz respeito à implementação de um esquema de paralelismo para o processamento de construções não-determinísticas, prevendo uma melhoria de desempenho, especialmente se executado em máquina com diversos processadores.

Ainda a respeito de construções não-determinísticas, uma outra melhoria seria a de possibilitar o usuário da ferramenta a escolher uma das possíveis soluções do modelo, de forma a exercitar até o final das possibilidades apenas a opção que atenda a algum critério do usuário da ferramenta, havendo assim

considerável redução de recursos no processo, seja a utilização de memória, ou mesmo o tempo total de execução.

7.3 TRABALHOS FUTUROS

A ampliação dos recursos da ferramenta, como citado na seção 7.2 do presente capítulo é uma das possíveis continuações deste trabalho. Outro é o desenvolvimento de um conjunto de funções a serem utilizadas como bibliotecas da ferramenta proposta, como a disponibilização de operadores nebulosos, por exemplo.

Este tipo de recurso aumentaria a aplicabilidade da ferramenta na solução de problemas mais complexos, e possibilitaria a comparação direta dos métodos de decisão.

Desenvolver um método para a aplicação de tabelas de decisão adaptativas ao processo de tomada de decisão em organizações, utilizando a ferramenta proposta neste trabalho para desenvolvimento da parte prática, não seria necessariamente uma continuação desta pesquisa, mas um próximo passo para a disseminação do uso da tecnologia adaptativa na área corporativa.

REFERÊNCIAS

ABELSON, H.; SUSSMAN, G. J.; SUSSMAN, J. **Structure and interpretation of computer programs.** 2nd edition. Cambridge; New York: MIT Press, McGraw-Hill, 1996.

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compilers, principles, techniques, and tools.** Reading: Addison-Wesley, 1986.

BARENDREGT, H. P. **The lambda calculus:** its syntax and semantics. Revised edition. Amsterdam, New York: North-Holland, 1984.

CERTO, S. C. **Modern management:** diversity, quality, ethics, and the global environment. 6th ed. Boston: Allyn and Bacon, 1994.

DIJKSTRA, E. W. Guarded commands, non-determinacy and formal derivation of programs. **Communications of the ACM**, v.18, n.8, p.453-457, 1975.

DYBVIG, R. K. **The Scheme programming language.** Englewood Cliffs: Prentice-Hall, 1987.

EISENBERG, M.; ABELSON, H**. Programming in Scheme.** Cambridge: MIT Press, 1990.

FINDLER, R. B. et al. DrScheme: A Programming Environment for Scheme. **Journal of Functional Programming (JFP)**, 12(2), p. 159-182, 2002.

FLATT, M. et al. Programming Languages as Operating Systems (or, Revenge of the Son of the Lisp Machine). **ACM SIGPLAN Notices**, v.34, n.9, p. 138-147, 1999. New York, 1999.

GORDON, M. J. C. **The denotational description of programming languages: an introduction.** New York: Springer-Verlag, 1979.

GORDON, M. J. C. **Programming language theory and its implementation:** applicative and imperative paradigms. New York: Prentice-Hall, 1988.

HAREL, D. et. al. On the formal semantics of statecharts. In: Symposium on Logic in Computer Science, 2, Ithaca, NY, 1987. **Proceedings.** New York, IEEE Press, 1987a, p.54-64.

HAREL, D. Statecharts: a visual formalism for complex systems. **Science of Computer Programming**, v.8, n.3, p.231- 74, Aug. 1987b.

HERNANDEZ, A. **Modelos na análise de decisão:** um estudo comparativo entre a utilização de árvores de decisão e de diagramas de influência. 1992. 146p. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo. São Paulo, 1992.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D**. Introduction to automata theory, languages, and computation.** 2nd edition. Boston: Addison-Wesley, 2001.

HUGHES, J. Why functional programming matters. **The Computer Journal**, v.32, n.2, p.98-107, 1989. Oxford, 1989.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informações Gerenciais.** 5ª edição. São Paulo: Prentice-Hall, 2004.

LEWIS, H.R.; PAPADIMITRIOU, C.H. **Elements of the theory of computation.** Upper Saddle River: Prentice-Hall, 1998.

LOUDEN, K. C. **Compiladores: Princípios e práticas.** São Paulo: Pioneira Thomson Learning, 2004.

McCARTHY, J. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. **Communications of the ACM**, 3, p.184-195, 1960.

MARCOTTY, M.; LEDGARD, H. F. **Programming language landscape:** syntax, semantics, and implementation. 2nd edition. New York: Macmillan, 1986.

MERNIK, M; HEERING, J; SLOANE, A. M. When and how to develop domain-specific languages. **ACM Computing Surveys**, v.37 , n.4, p.316-344, 2005. New York, 2005.

MITCHELL, J. C. **Foundations for programming languages.** Cambridge: MIT Press, 1996.

MITCHELL, T. M. **Machine Learning.** New York: McGraw-Hill, 1997.

MONTALBANO, M. **Decision Tables.** Chicago: Science Research Associates, 1974.

NAGAYAMA, S. **Tabelas de decisão e implementação do gerador i-m-e.** 1990. 237p. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo. São Paulo, 1991.

NETO, J. J.; MAGALHÃES, M. E. Reconhecedores sintáticos: uma alternativa didática para uso em cursos de engenharia. In: XIV Congresso Nacional de Informática - SUCESU. São Paulo, SP. **Anais da XIV Congresso Nacional de Informática - SUCESU.** São Paulo, 1981. p. 171-181.

NETO, J. J. **Introdução à Compilação**. São Paulo: LTC, 1987.

NETO, J. J. **Contribuições à metodologia de construção de compiladores.** 1993. 272p. Tese (Livre Docência) - Escola Politécnica, Universidade de São Paulo. São Paulo, 1993.

NETO, J. J. Adaptive automata for context-dependent languages. **ACM SIGPLAN Notices**, v.29, n.9, p.115-124, 1994. New York, 1994.

NETO, J. J. Adaptive rule-driven devices - general formulation and case study. In: International Conference on Implementation and Application of Automata - CIAA 2001, 6th., 2001, Pretoria, South Africa. **Lectures Notes on Computer Science 2494.** Berlin: Springer-Verlag, 2002, p.234-250.

NORVIG, P**. Paradigms of artificial intelligence programming:** case studies in common LISP. San Mateo: Morgan Kaufman Publishers, 1992.

PEDRAZZI, T. C.; MATSUNO, I. P.; ROCHA, R. L. A. **Uma Proposta de Ensino da Disciplina de Linguagens Formais e Autômatos para um Curso de Engenharia da Computação.** In: *Congresso Brasileiro de Ensino de Engenharia - COBENGE2004*, 2004, Brasília, 2004.

PEDRAZZI, T. C.; TCHEMRA, A. H.; ROCHA, R. L. A. Adaptive Decision Tables - A Case Study of their Application to Decision-Taking Problems. In: 7th International Conference on Adaptive and Natural Computing Algorithms - ICANNGA2005, 2005, Coimbra. **Adaptive and Natural Computing Algorithms.** Viena: Springer-Verlag, 2005. v. 1. p. 341-344.

PIDD, M. **Modelagem empresarial:** ferramentas para tomada de decisão. Imprenta Porto Alegre: Bookman, 2001.

PISTORI, H. **Tecnologia adaptativa em engenharia de computação:** estado da arte e aplicações. 2003. 172p. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo. São Paulo, 2003.

PRESSMAN, R. S. **Software engineering:** a practitioner's approach. 6th edition. Boston: McGraw-Hill, 2005.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems.** 3rd edition. Boston: McGraw-Hill, 2002

ROCHA, R. L. A. **Um método de escolha automática de soluções usando tecnologia adaptativa.** 2000. 221p. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo. São Paulo, 2000.

ROCHA, R. L. A.; NETO, J. J. Autômato adaptativo, limites e complexidade em comparação com máquina de Turing. In: I Congresso de Lógica Aplicada à Tecnologia - LAPTEC´2000, 2000, São Paulo. **Anais do I Congresso de Lógica Aplicada á Tecnologia - LAPTEC´2000.** São Paulo: Faculdade SENAC de Ciências Exatas e Tecnologia, 2000. p. 33-48.

ROCHA, R. L. A.; NETO, J. J. Uma proposta de linguagem de programação funcional com características adaptativas. In: IX Congreso Argentino de Ciencias de la Computación - CACIC 2003, 2003, Buenos Aires. **Anales del IX Congreso Argentino de Ciencias de la Computación - CACIC 2003.** Buenos Aires, 2003. v. CD-ROM. p. 1664-1674.

RUSSEL, S. J.; NORVIG, P. **Artificial intelligence:** a modern approach. Upper Saddle River: Prentice Hall, 1995.

SEBESTA, R. W. **Concepts of programming languages.** 5th edition. Reading: Addison-Wesley, 1996.

SHIMIZU, T. **Decisão nas organizações:** introdução aos problemas de decisão encontrados nas organizações e nos sistemas de apoio à decisão. São Paulo: Atlas, 2001.

SHIMIZU, T; CARVALHO, M. M. de; LAURINDO, F. J. B. **Strategic alignment process and decision support systems:** theory and case studies. Hershey: IRM Press, 2006.

SOMMERVILLE, I. **Engenharia de software.** 6ª edição. São Paulo: Addison Wesley, 2003.

TURBAN, E.; ARONSON, J. E. **Decision support systems and intelligent systems.** 6th ed Upper Saddle River: Prentice Hall, 2001.

WINSTON, P. H.; HORN, B. K. P. **LISP.** 3rd edition. Reading: Addison-Wesley, 1989.

ANEXO I

Gramática da linguagem de entrada da ferramenta proposta em formato BNF.

```
<programa>              ::=  tda
                             <declaracao-variaveis>
                             <funcoes-adaptativas>
                             Inicio
                                    <lista-de-regras>
                             Fim;

<declaracao-variaveis>  ::=  <variavel>
                        |    <declaracao-variaveis>;<variavel>

<variavel>              ::=  <tipo-variavel>:<lista-de-identificadores>;
                        |    ε

<tipo-variavel>         ::=  numero
                        |    moeda
                        |    caract

<funcoes-adaptativas>   ::=  <adaptativa>
                        |    <funcoes-adaptativas>;<adaptativa>

<adaptativa>            ::=  funcao <identificador>
                             (<lista-de-objetos>)
                             Inicio-adaptativa
                             ?: <lista-de-regras>
                             -: <lista-de-regras>
                             +: <lista-de-regras>
                             Fim-adaptativa;
                        |    ε
```

<lista-de-objetos> ::= <objeto>
| <lista-de-objetos>;<objeto>

<objeto> ::= <tipo-objeto>:<lista-de-identificadores>;
| ε

<tipo-objeto> ::= parametros
| variaveis
| geradores

<lista-de-regras> ::= <regra>
| <lista-de-regras>;<regra>

<regra> ::= se (<lista-de-condicoes>) entao
<acao-adaptativa>(<lista-de-acoes>)<acao-adaptativa>;
| ε

<lista-de-condicoes> ::= <condicao>
| <lista-de-condicoes>,<condicao>

<condicao> ::= <expressao><operador-comparacao><expressao>

<expressao> ::= <expressao>+<expressao>
| <expressao>-<expressao>
| <expressao>*<expressao>
| <expressao>/<expressao>
| <identificador>
| <numero>
| ‘<cadeia-de-caracteres>’

<operador-comparacao> ::= >
| <
| =
| !=

<acao-adaptativa> ::= <identificador>(<lista-de-parametros>)
| ε

<lista-de-parametros> ::= <parametro>
| <lista-de-parametros>,<parametro>

<parametro> ::= <identificador> <numero>
| <identificador> ‘<cadeia-de-caracteres>’
| ε

<lista-de-acoes> ::= <acao>
| <lista-de-acoes>,<acao>

<acao> ::= <atribuicao>
| <leitura>
| <impressao>
| ε

<atribuicao> ::= <identificador>:=<expressao>

<leitura> ::= Ler<identificador>

<impressao> ::= Imprimir<identificador>

<lista-de-identificadores> ::= <identificador>
<lista-de-identificadores>,<identificador>

<identificador> ::= <cadeia-de-caracteres>

<numero> ::= <cadeia-de-digitos>
| <cadeia-de-digitos>,<cadeia-de-digitos>

<cadeia-de-digitos> ::= <digito>
| <cadeia-de-digitos><digito>

<cadeia-de-caracteres> ::= <letra>
| <cadeia-de-caracteres ><letra>
| <cadeia-de-caracteres ><digito>

<letra> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | _ | . | -

<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9