Adaptive Dynamic Discovery Language

R. G. Dutra, R. Silva, M. Martucci Jr., W. Vicente Ruggiero

Resumo - O objetivo deste artigo é propor uma meta linguagem de tradução de regras, que permita a descoberta dinâmica de serviços WEB, baseada nos atributos não funcionais que caracterizam estes serviços. Como gerador das regras de produção desta linguagem, será utilizada a ferramenta *Adaptive Fuzzy Neural Tree Network* (*AFNTN*).

Como resultado desta meta linguagem espera-se a criação de documentos no padrão XML, que possam ser utilizados por outras linguagens de definição, gerenciamento e oferta de serviços WEB, complementando estas linguagens com funções adaptativas para a descoberta dinâmica.

Palavras Chave – Linguagens de programação, Sistemas Adaptativos, Árvores de Decisão, Redes Neurais Artificiais, Lógica Difusa.

I. INTRODUÇÃO

Atualmente um grande problema para o gerenciamento de serviços WEB, é o processo de descoberta, procurando o serviço de acordo com as suas funcionalidades. Geralmente os desenvolvedores acessam algum registro em um repositório e a busca é realizada por meio de palavras chave, que possam traduzir a funcionalidade que se almeja. É o mesmo problema enfrentado nos sites de busca na WEB.

A solução para localizar um serviço de acordo com seus atributos não funcionais, tais como qualidade do serviço, performance, aspectos de segurança, entre outros, além dos atributos funcionais, é a combinação de algoritmos de prospecção de dados [21].

Para finalizar o processo de descoberta de serviços, é necessário traduzir as regras que classificam os mesmos em uma linguagem de programação aberta, como XML (Extensible Markup Language), que permita a leitura ou interpretação destes serviços por outros algoritmos para consumo dos mesmos.

Diversas linguagens de programação baseadas no padrão XML foram propostas na literatura [1], destacando-se entre elas:

- WSDL (*WEB Services Definition Language*), criada para descrever a especificação de serviços *WEB*, baseado em atributos funcionais. Atualmente bastante utilizada para a descoberta estática de serviços.
- OWL-S (*Ontology WEB Language for Services*), antigamente conhecida como DAML-S (*Darpa Agent Markup Language for WEB Services*) criada para descrever a especificação de servicos WEB, baseado em ontologias de atributos funcionais e não funcionais.
- WSOL (*WEB Services Offering Language*), criada para oferecer serviços *WEB*, baseado em atributos funcionais e não funcionais, para composição e coreografia de serviços.

Segundo [1], a linguagem WSOL é compatível com WSDL, acrescentando-se atributos não funcionais e permitindo composição dinâmica entre diversos serviços que possuem atributos funcionais similares, necessidade primordial da arquitetura SOA, não coberta pela linguagem OWL-S.

Entretanto, nenhuma das linguagens citadas ou encontradas na literatura até o presente momento, permitem a realização do processo dinâmico de descoberta de serviços *WEB*, baseado em características não funcionais.

Este artigo objetiva a exploração das ferramentas de descoberta dinâmica *(dynamic discovery),* visando a solução deste problema, através da combinação de métodos de Inteligência Artificial, tais como Redes Neurais Artificiais, Árvores de Decisão Adaptativas e Lógica *Fuzzy* em um modelo denominado de *Adaptive Fuzzy Neural Tree Network* (AFNTN) [22].

O problema alvo consiste em classificar serviços WEB, através dos atributos não funcionais que descrevem estes serviços conforme linguagem WSOL, de forma não supervisionada, ou seja, o número de classes e os atributos utilizados para definição das mesmas não é conhecido a priori, cabendo a AFNTN sua determinação e geração de regras de produção de uma linguagem de tradução para XML.

O conteúdo deste artigo está organizado de forma a apresentar inicialmente uma introdução sobre a teoria de linguagens formais de programação, seguida da descrição de mecanismos adaptativos para indução de árvores decisão e a descrição do algoritmo de aprendizado não supervisionado. Em seguida é discutido como tratar incerteza nos dados e a modelagem para implementação da ferramenta proposta neste trabalho. Nas seções finais são apresentados resultados e conclusões obtidas através da aplicação da linguagem gerada pela *AFNTN*.

I. CONCEITOS SOBRE LINGUAGENS DE PROGRAMAÇÃO

Serviços são componentes de software que representam um processo, entidade, atividade ou tarefa. Existem quatro tipos de Serviços:

- Básicos: que representam os elementos básicos de um processo de negócio, como Entidades e Tarefas básicas de negócios;
- Intermediários: são o único tipo de Serviço mais orientados a tecnologia em uma arquitetura orientada a serviços (SOA). Fornecem pontes, conversores ou funcionalidades adicionais aos demais serviços;

- Processos: são os serviços que representam de forma direta um processo ou atividade de negócio, do início ao fim.
- Públicos: extensão aos serviços do tipo Processo que possibilita sua exposição para clientes (usuários) que estejam fora das fronteiras de um determinado domínio.

Todo serviço, independente de seu tipo, é sempre composto por três partes principais. A primeira parte é um *contrato*, um acordo que é fechado entre os consumidores e seus provedores. O contrato explica os propósitos, contexto, regras de utilização, restrições, níveis de serviço esperados, além de apresentar uma definição formal da interface. Tal *interface* é implementada em separado e constitui o sendo segundo elemento de construção de um serviço, representando o único meio de comunicação com o mesmo. A terceira e última parte de um serviço é sua *implementação* propriamente dita, através da realização da lógica do negócio e acesso e manutenção de seus dados, de forma a atender todos os objetivos fixados no contrato.

Um *Repositório* de serviços armazena todos os *Contratos* dos Serviços disponíveis, o que o torna o ponto de partida para utilização destes. Além dos *Contratos*, o *Repositório* pode armazenar informações adicionais e mais específicas acerca dos serviços, como localização física, restrições de uso e segurança, etc. Apesar de ser apresentado por alguns autores como um elemento opcional em uma SOA, o *Repositório* pode ser fator crítico de sucesso em grandes implementações, principalmente naquelas que envolverem a disponibilização de serviços do tipo Público. Neste contexto, os serviços podem ser publicados na *World Wide Web*, sendo denominados de *WEB Services*.

O UDDI (*Universal Description, Discovery, and Integration*) especifica um mecanismo centralizado para os provedores de *WEB Services* anunciarem a existência de seus serviços e para os consumidores poderem localizar os serviços de seu interesse,ie, um *Repositório* de *WEB Services*. Este registro funciona como uma grande lista telefônica cujos serviços são catalogados.

O principal componente do UDDI é um arquivo XML (*Extensive Markup Language*) usado para descrever serviços *WEB*. A informação deste arquivo consiste em três componentes:

- ***Páginas brancas***: informações de contato (nome, endereço, URLs, etc.).
- ***Páginas amarelas***: categorização dos serviços baseados em taxonomias.
- ***Páginas verdes***: informações técnicas sobre o serviço publicado, como especificações do serviço WEB e sua localização.

A especificação do UDDI consiste em um documento no formato XML *Schema* para as mensagens SOAP (*Simple Object Access Protocol*) e a descrição da especificação da *API (Aplication Program Interface)* do UDDI. Juntos eles formam um modelo básico que permite a publicação de informação sobre um grande número de *WEB Services*.

O reconhecimento de padrões que permitam a classificação de atributos não funcionais de serviços WEB, necessita de um método padronizado para expressar instruções dentro de uma arquitetura SOA, ou seja, um conjunto de regras sintáticas e semânticas que fundamentam uma "linguagem formal" de programação.

Entende-se por "linguagens formais", os mecanismos formais para representação e especificação de linguagens, baseados na chamada "Teoria da Computação". As representações podem ser feitas por reconhecedores e geradores. Os reconhecedores são dispositivos formais que servem para verificar se uma sentença pertence ou não à determinada linguagem. São os autômatos finitos, autômatos de pilha e Máquina de Turing. Os sistemas geradores são dispositivos formais que permitem a geração sistemática de todas as sentenças de uma linguagem. A tabela 1 a seguir, ilustra os reconhecedores utilizados em função de cada tipo de gramática segundo a classificação de Chomsky [20].

Teoria de Autômatos: Linguagem formal e gramática formal			
Hierarquia Chomsky	Gramática	Linguagem	Reconhecedor
Tipo-0	Estrutura de frase	Recursivamente enumerável	Máquina de Turing
--	Estrutura de frase	Recursiva	Máquina de Turing
Tipo-1	Sensíveis ao contexto	Sensíveis ao contexto	Máquina de Turing com memória limitada
Tipo-2	Livre de contexto	Livre de contexto	Autômato com pilha
Tipo-3	Regular	Regular	Autômato finito

Tabela 1 – Classificação de Gramáticas segundo a hierarquia de Chomsky.

A classificação das gramáticas começam pelo tipo 0, com maior nível de liberdade em suas regras, e aumentam as restrições até o tipo 3. Cada nível é um super-conjunto do próximo. Logo, uma gramática de tipo n é conseqüentemente uma gramática de tipo n - 1.

Uma gramática G pode ser formalmente definida [2] por uma quádrupla (V, Σ, R, S), onde:

- V é um alfabeto;
- Σ é um alfabeto de símbolos terminais e um subconjunto de V;
- R é um conjunto de regras pertencente ao subconjunto de (V- Σ) x V*;
- S é o símbolo inicial.

Para o reconhecimento de gramáticas não determínicas, requer-se a utilização de autômatos não-determinísticos, uma vez que a trajetória de reconhecimento de uma sentença nem sempre é única, exigindo que sua operação seja, muitas vezes, efetuada por meio de tentativas e erros ("*backtracking*") [18]. Isto acarreta uma substancial perda de eficiência do reconhecedor, tornando anti-econômico o seu uso em aplicações práticas.

Segundo Neto [19], autômatos adaptativos podem ser empregados para a contrução de reconhecedores de gramáticas livres ou dependentes de contexto, determinísticos ou não. O reconhecimento consiste em uma análise sintática (*parsing*), definido como o processo de analisar uma sequência de entrada (lida de um arquivo de computador ou do teclado, por exemplo) para determinar sua estrutura gramatical segundo uma determinada gramática formal. Essa análise faz parte de um compilador, junto com a análise léxica e análise semântica.

A análise sintática transforma um texto na entrada em uma estrutura de dados, em geral uma árvore, o que é conveniente para processamento posterior e captura a hierarquia implícita desta entrada. Através da análise léxica é obtido um grupo de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura, como ilustrado na figura 1.1.

Fig 1.1 – Exemplo de análise sintática (*parsing*)

Em termos práticos, pode também ser usada para decompor um texto em unidades estruturais para serem organizadas dentro de um bloco, por exemplo. A vasta maioria dos analisadores sintáticos implementados em compiladores aceitam alguma linguagem livres de contexto para fazer a análise. Estes analisadores podem ser de vários tipos, como o LL, LR e SLR [17].

II. UTILIZAÇÃO DE DISPOSITIVOS ADAPTATIVOS

O problema de indução incremental de árvores de decisão para atributos discretos pode ser resolvido através da aplicação da tecnologia adaptativa, utilizando um dispositivo adaptativo descrito no algoritmo *AdapTree* [3].

Um dispositivo adaptativo [4] é constituído em duas partes: a primeira é algum dispositivo usual, definido através de regras (em particular, algum autômato, gramática ou qualquer outro dispositivo descrito através de um conjunto finito de regras estáticas), denominado seu dispositivo subjacente (tipicamente não-adaptativo); a segunda é um mecanismo adaptativo, cuja conexão ao formalismo subjacente proporciona-lhe todos os recursos complementares necessários para a realização das propriedades responsáveis pela auto-modificação autônoma que caracteriza os dispositivos adaptativos.

Um autômato adaptativo [5] de estados finitos (AAF) é um dispositivo adaptativo, que estende o poder de expressão do autômato de estados finitos (AF), através da capacidade de modificar a sua própria estrutura com a aplicação de regras adaptativas citadas anteriormente. Formalmente, um AAF pode ser definido conforme Quadro 2.1:

$M = \langle Q, \Sigma, q_0, F, \delta, Q_\infty, \Gamma, \Pi \rangle$

onde:

$Q \subseteq Q_\infty$ é um subconjunto finito de um conjunto, possivelmente infinito, de estados.

Σ é o alfabeto de entrada, finito e não vazio.

$q_0 \in Q$ é o estado inicial do autômato.

$F \subseteq Q$ é o conjunto de Estados Finais.

$\delta \subseteq (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty)$ é a relação de transição.

Os três últimos elementos de M, responsáveis pela adaptabilidade do sistema, são definidos a seguir:

Q_∞ é um conjunto, possivelmente infinito, de estados.

$\Gamma \subseteq (\{+,-\} \times (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty))$ é o conjunto de ações primitivas que podem ser executadas pelo AAF: incluir (+) ou excluir (-) transições [4].

$\Pi : (Q_\infty \times (\Sigma \cup \{\epsilon\}) \times Q_\infty) \rightarrow \Gamma^*$ é uma função que associa a cada transição em δ uma seqüência de ações adaptativas primitivas. Podemos associar uma seqüência vazia a uma determinada transição para indicar que esta é uma transição "normal", não adaptativa.

Quadro 2.1 – Definição formal de um AAF

Neste AAF, para toda a transição:

$$x \in \delta \text{ onde } \Pi(x) = \epsilon \qquad \text{(eq. 2.1)}$$

O autômato adaptativo se reduz, funcionalmente, a um simples autômato de estados finitos. Nos outros casos, antes de executar a transição, o AAF deverá efetuar as alterações adaptativas sugeridas pelas ações adaptativas associadas a mesma. Após executar as ações adaptativas, o AAF passa a operar com sua nova estrutura.

O dispositivo adaptativo no qual o *AdapTree* se baseia, pode ser visto como um AAF classificador, estendido para trabalhar com mais de duas classes de serviços. Neste contexto, a cada estado final é associado um elemento, que corresponderá a uma das classes possíveis. Ao receber um exemplo de treinamento o *AdapTree* cria um caminho ligando o estado inicial do autômato ao estado final correspondente à classificação deste exemplo.

Formalmente, dado n conjuntos disjuntos:

$A_1, A_2, ..., A_n$

O conjunto de treinamento é composto por:

$T = \{w_1, w_2, ..., w_m\}$

Onde:

$|w_i| = n,\ w_i = \alpha_1\alpha_2...\alpha_n \text{ com } \alpha_j \in A_j$

Para:

$1 \leq j \leq n$

$1 \leq i \leq m$

O último símbolo da cadeia é utilizado para representar a calsse do exemplo, sendo que |An|, determina a quantidade de classes do problema e n-1 é o total de atributos (variáveis) a serem considerados no processo de aprendizagem. Os conjuntos:

$A_1, A_2, ..., A_n$

Representam os domínios – valores permitidos – de cada atributo.

Para alternar o classificador do modo de treinamento para o modo de classificação, basta fornecer cadeias de caracteres de tamanho (n-1), excluindo dessa forma as classificações.

Caso não seja possível determinar, sintaticamente, a classe de uma cadeia de entrada, o *AdapTree* utiliza o mecanismo estatístico **ID3** [6], que fornecerá como resposta uma

estimativa baseado no ganho de entropia de informação, definindo a ordem dos atributos utilizada nesta classificação.

Porém, segundo Quinlan [7], um atributo com muitos valores possíveis, teria uma dispersão maior na distribuição de probabilidades desses valores, conseqüentemente maior ganho de entropia de informação. Para evitar esta distorção, que favorece o atributo com maior número de valores, introduziu-se no algoritmo **C4.5** [8], sucessor do ID3, o conceito de Razão do Ganho de Entropia de Informação.

O método **C4.5** tem sido largamente empregado para construir árvores de decisão (*decision tree* ou DT) que implementam classificadores de elevada performance. Contudo, este algoritmo só permite classes previamente definidas para classificação de atributos na fase de treinamento, não tendo a capacidade de interpolar ou deduzir novos padrões por inferência ou tratar dados imprecisos ou incertos. Como mostrado em [9] e [10], o **C4.5** somente pode delimitar hiperplanos paralelos aos eixos coordenados, fato que, em um espaço de atributos contínuos, implicaria em crescimento exponencial da DT resultante.

Diferentemente do **C4.5**, o *AdapTree* não particiona o conjunto de exemplos, ou seja, o princípio de otimização é global e não local. As árvores de decisão geradas pelo *AdapTree* não possuem um número excessivo de ramificações, devido a esta característica.

Adicionalmente, como o *AdapTree* é um algoritmo incremental, onde novos exemplos podem ser incorporados durante o processo de classificação, resolvendo adequadamente o problema de variação temporal de padrões, muito comum em atividades de *Data Mining*. A combinação do *AdapTree* e do mecanismo C4.5, permite maximizar a vantagem de ambos algoritmos, minimizando as desvantagens citadas.

No entanto, ambos *AdapTree* e C4.5 necessitam de um conjunto de treinamento para geração de uma DT cujo número de nós entre a raiz e folhas não seja excessivamente grande. Dessa forma, o *AdapTree* e o C4.5 são algoritmos que trabalham em modo supervisionado durante a fase de treinamento, ou seja, não são capazes de definir o número de classes que particionam o conjunto de treinamento.

Por outro lado, *Redes Neurais Artificiais* (RNA) têm sido empregadas em tarefas de classificação para determinação de padrões em modo supervisionado ou não supervisionado.

Classes complexas podem ser prospectadas através dos dados, a fim de gerar uma DT de forma mais rápida, eficiente e de simples visualização.

Evidentemente que a performance e qualidade dos padrões adquiridos estão diretamente correlacionados com o tipo e a arquitetura escolhida para a modelagem da RNA, como demonstram os estudos realizados comparando-se a arquitetura *Multi-Layered Perceptron* (MLP) com DT [7].

A seguir, será apresentado um algoritmo de RNA adaptativo, porém não baseado no paradigma de Autômatos Adaptativos, cuja vantagem é não necessitar de um conjunto de treinamento classificado a priori, como o *AdapTree*.

III. Redes Neurais Artificiais

O algoritmo *Self-Organizing Map* (SOM), desenvolvido por Teuvo Kohonen [11], é um dos modelos mais populares de RNA. O algoritmo da SOM é baseado em um aprendizado competitivo e não supervisionado, o que implica em um treinamento direcionado exclusivamente pelos dados, sendo que os neurônios que constituem o mapa competem entre si para adquirir padrões dos dados, se aproximando deles. Algoritmos supervisionados, como o *Multi-Layered Perceptron* (MLP), requerem uma classificação pré-definida para cada vetor de treinamento, além de depender fundamentalmente do número de camadas internas (*hidden units*) para um aprendizado com baixo erro de classificação e boa performance, limitações que não ocorrem na arquitetura SOM.

No algoritmo interativo da SOM [15], cada neurônio i é representado por um vetor n-dimensional de pesos conhecido como vetor protótipo ou vetor de referência m_i:

$$m_i = [m_{i1},...,m_{in}]^T \qquad \text{(eq. 3.1)}$$

Onde:

- n é igual ao número de dimensões do vetor de entrada;
- m_{in} é o n-ésimo peso do vetor m_i.

No algoritmo básico, as relações topológicas e o número de neurônios são previamente fixados durante a fase de estruturação da rede. O número de neurônios deve ser usualmente escolhido como o maior possível, assim como o tamanho da vizinhança de forma a buscar um equilíbrio entre performance computacional e um bom grau de generalização e granularidade.

Antes da etapa de treinamento não supervisionado da SOM, valores são escolhidos para os pesos componentes dos vetores de referência.

Em cada passo do treinamento, um vetor de entrada de amostra x dos dados é escolhido aleatoriamente e uma medida de similaridade é calculada entre este e todos os vetores de referência do mapa da SOM. A unidade do mapa que mais se assemelha, também conhecida na literatura como BMU (Best Matching Unit), denominada aqui como unidade vencedora ***c***, é a unidade que possui a menor medida distância, tipicamente distância Euclidiana, segundo a equação 3.2:

$$\|x - m_c\| = \min_i \{\|x - m_i\|\} \qquad \text{(eq. 3.2)}$$

Onde $\|\,.\,\|$ denota a medida de distância Euclidiana.

A regra de atualização para o vetor de referência de uma unidade i da SOM pode ser expressa pela eq. 3.3:

$$m_i(t+1) = m_i(t) + h_{ci}(t)\left[x(t) - m_i(t)\right] \qquad \text{(eq. 3.3)}$$

Onde:

- t denota o instante de tempo ou passo do treinamento;
- x(t) o vetor de entrada selecionado aleatoriamente dos dados em um instante t;

- $h_{ci}(t)$ a relação de vizinhança ao redor de uma BMU ***c*** em um instante t.

Apesar do algoritmo de treinamento ser relativamente simples, o tratamento matemático para obtenção de demonstrações de convergência é bastante complexo. Segundo Vesanto[15], a convergência em um mapa unidimensional ocorre na maioria dos casos. Neste caso, para um número elevado de neurônios cujo raio R final de vizinhança, densidade pontual para os vetores referência da SOM é proporcional à eq. 3.4:

$$p(x)^{\frac{2}{3}-\frac{1}{3R^2+3(R+1)^2}} \qquad \text{(eq. 3.4)}$$

Onde p(x) é a função de densidade de probabilidade das entradas x.

Nesta expressão, quando a dimensão dos vetores de entrada aumenta o expoente acima tende a unidade, assim como a distribuição do vetor de pesos se aproxima aos dados de treinamento e o algoritmo de treinamento converge.

Entretanto, a presença de imprecisão, incerteza ou ruído nos dados pode reduzir drasticamente a performance treinamento não supervisionado do algoritmo SOM, da mesma forma que induz árvores excessivamente grandes utilizando os algoritmos *AdapTree* e C4.5. Para tratar de forma eficaz a incerteza, torna-se necessário a utilização de um mecanismo de inferência baseado em lógica nebulosa.

IV. LÓGICA NEBULOSA

A lógica nebulosa (*fuzzy*) possibilita que seja abordado, de forma mais adequada, o problema referente à representação e manipulação de imprecisão ou incerteza.

Os sistemas baseados em lógica nebulosa foram criados por Zadeh[12], fundamentando-se na representação e manipulação de informações incertas e imprecisas tão comuns no cotidiano humano. Expressões tais como "quase", "muito" e "pouco" representam este tipo de imprecisão, que usualmente não pode ser tratada pelos sistemas da lógica clássica de forma simples.

Os sistemas especialistas *fuzzy* utilizam um conjunto de regras do tipo "*If-Then*", baseadas em variáveis nebulosas. Primeiramente as variáveis de entrada sofrem um processo de "*fuzzificação"*, ou seja, os conjuntos nebulosos das variáveis lingüísticas de entrada são ativados. Terminado este processo, efetua-se a inferência sobre o conjunto de regras nebulosas obtendo os valores das variáveis de saída. Finalmente, as variáveis de saída sofrem um processo de *"defuzzificação*".

Este processo consiste em converter os dados nebulosos para valores numéricos precisos. Para isto são utilizadas várias técnicas, tais como valor máximo, média dos máximos, média local dos máximos, centro de gravidade, ponto central da área, entre outros. Neste artigo, foi utilizado o método de chamado *Takagi-Sugeno* [13], ou simplesmente *Sugeno*, cuja saída representará um valor constante, representando uma classe definida pela RNA do tipo SOM.

V. APLICAÇÃO DA ADAPTIVE FUZZY NEURAL TREE NETWORK

Existem inúmeras formas de se combinar árvores de decisão adaptativas [1], RNA do tipo SOM e Lógica *Fuzzy* para compor a *Adaptive Fuzzy Neural Tree Network*, porém a escolha dos algoritmos, neste artigo, objetiva suportar as atividades de descoberta dinâmica de serviços. O processo de descoberta dinâmica, nesse contexto, consiste basicamente em interceptar as requisões de serviços provenientes de um consumidor e intermediar a procura nos possíveis provedores de serviços, baseado na classificação de atributos não funcionais dos mesmos.

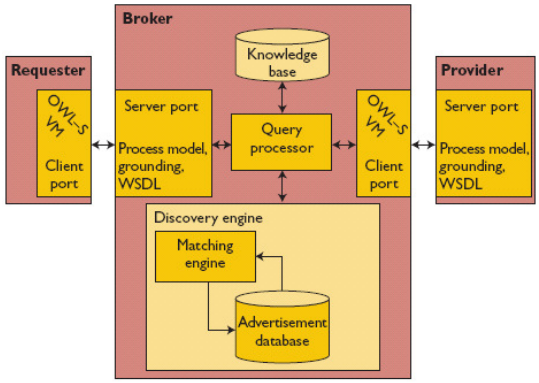

Fig. 5.1 . Intermediador de serviços WEB [14]

A figura 5.1 ilustra o modelo de um intermediador (*broker*), entre o consumidor (*requester*) e provedor (*provider*), utilizando características das linguagens OWL-S e WSDL.

Cabe a *AFNTN* realizar as tarefas de procura (*matching*), realizando uma busca sobre a base de dados (*advertising database*) que contém os atributos não funcionais dos serviços, conforme ilustrado na figura 5.2.

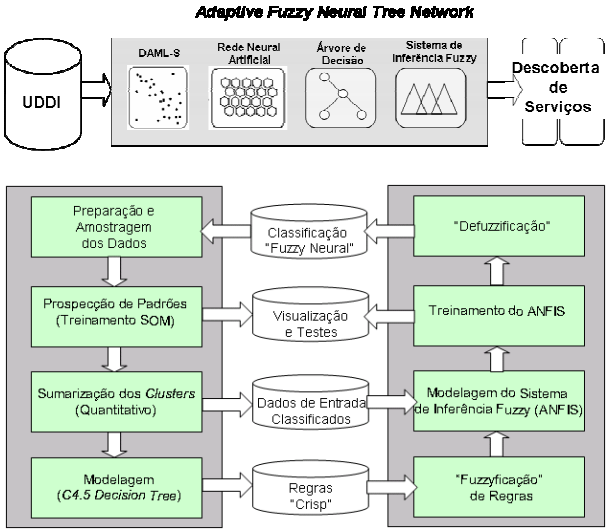

Fig. 5.2. Diagrama de Blocos da *Adaptive Fuzzy Neural Tree Network (AFNTN)*

A base de dados utilizada é composta de características de serviços WEB, geradas a partir dos *services profiles* utilizados pela linguagem OWL-S. A função do service profile é mapear

características de entidades e serviços de negócios em um único repositório. Nesta base, constam aproximadamente 500 *services profiles*, contendo os seguintes atributos não funcionais:

- **QoS (*Quality of Service*)** – índice de qualidade numérico que varia de 0 a 300 (valor adimensional).
- **Preço Médio** – valor cobrado pela utilização do serviço, recebe valores numéricos de 1 a 5000 (valor adimensional).
- **Volume de *Handover* Vertical** – número de vezes na qual um mesmo serviço foi disponibilizado por um provedor diferente, variando entre 1 e 1500 (valor adimensional).
- **Volume de *Handover* Horizontal** - número de vezes na qual diferentes serviços foram disponibilizados por um mesmo provedor, variando entre 1 e 5100 (valor adimensional).

Aplicando-se estes atributos e seus respectivos valores para os services profiles utilizados para classificação da RNA do tipo SOM, resultaram 4 classes que mapeiam os 500 *services profiles*.

Todas as funções utilizadas no treinamento do algoritmo da SOM e os resultados obtidos, basearam-se em funções previamente elaboradas em Matlab® versão 6.5 provenientes da SOMTOOLBOX 2.0 [15].

Estas classes foram utilizadas como entrada da árvore de decisão adaptativa, resultando na seguinte ontologia de classes de serviços, conforme ilustrado na figura 5.3.

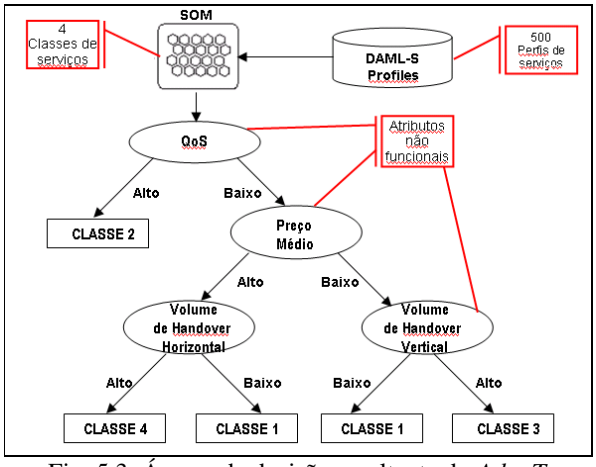

Fig. 5.3. Árvore de decisão resultante do *AdapTree* modificado

É possível inferir as seguintes regras no quadro 5.1, através da árvore de decisão ilustrada na fig. 5.3:

Regra 1. SE QoS:= "Alto" ENTÃO
Serviço PERTENCE Classe2
Regra 2. SE QoS:= "Baixo" E
Preço Médio:= "Baixo" E
Volume de Handover Vertical := "Alto" ENTÃO
Serviço PERTENCE Classe3
Regra 3. SE QoS:= "Baixo" E
Preço Médio:= "Alto" E
Volume de Handover Horizontal := "Baixo" ENTÃO
Serviço PERTENCE Classe3
Regra 4. SE QoS:= "Baixo" E
Preço Médio:= "Alto" E
Volume de Handover Horizontal := "Alto" ENTÃO
Serviço PERTENCE Classe4

Quadro 5.1 Regras inferidas pela árvore de decisão, gerada a partir do *AdapTree*

Os termos *fuzzy* "Alto" ou "Baixo" foram utilizados para "*fuzzificar*" as regras rígidas (*crisp*) geradas pela árvore de decisão.

A definição das funções de pertinência dos termos *fuzzy* será de responsabilidade do sistema adaptativo de inferência *fuzzy* (ANFIS), do tipo *Sugeno* ilustrado na fig. 5.4.

Os dados classificados e as regras *fuzzy* foram utilizadas como entrada do sistema de inferência *fuzzy* do tipo *Sugeno*, fornecido pela FUZZYTOOLBOX [16] do Matlab®.

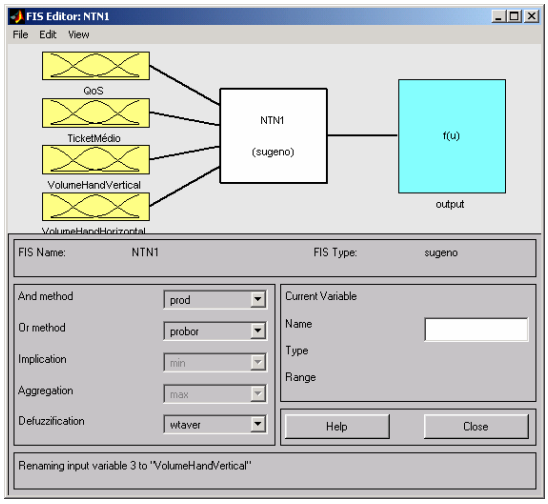

Fig. 5.4. Sistema de inferência *Fuzzy* Adaptativo

Após o treinamento supervisionado do ANFIS do tipo *Sugeno*, obteve-se o ajuste das funções de pertinência de entrada e saída, conforme Figura 5.5:

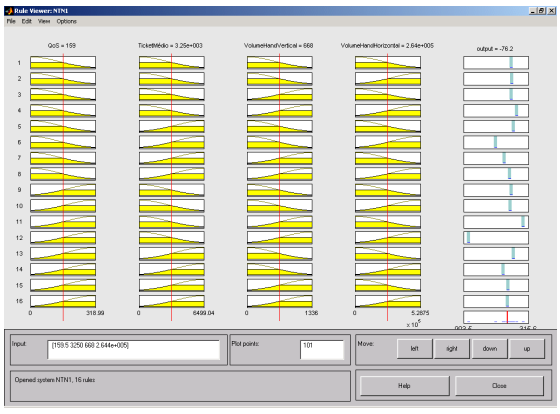

Fig. 5.5. Resultados do treinamento do ANFIS

O sistema de Inferência *Fuzzy* Adaptativo do tipo *Sugeno* (ANFIS) utiliza uma rede neural do tipo *Multi-Layer Perceptron* (MLP) [16] para ajuste das funções de pertinência.

A combinação das variáveis de entrada com as possíveis classes de saída é realizada através da criação de regras *fuzzy* do tipo "*E*", conforme quadro 5.2 a seguir:

```
SE Variável_Entrada_1 PERTENCE A Função_Pertinência_1
  E
    Variável_Entrada_2 PERTENCE A Função_Pertinência_1
  E ...
    Variável_Entrada_n PERTENCE A Função_Pertinência_1
ENTÃO
    Saída PERTENCE A Funcão_Pertinência_Classe_1
```

Quadro 5.2 Fragmento do arquivo de configuração léxica para o FLEX

As funções de pertinência delimitam superfícies de decisão, como ilustrado na Figura 5.6, onde as variáveis de entrada *Devoluções* e *Preço Médio* delimitam uma superfície tridimensional que, por sua vez, delimita a função de pertinência da classificação de saída (*output*), determinada pela RNA do tipo SOM.

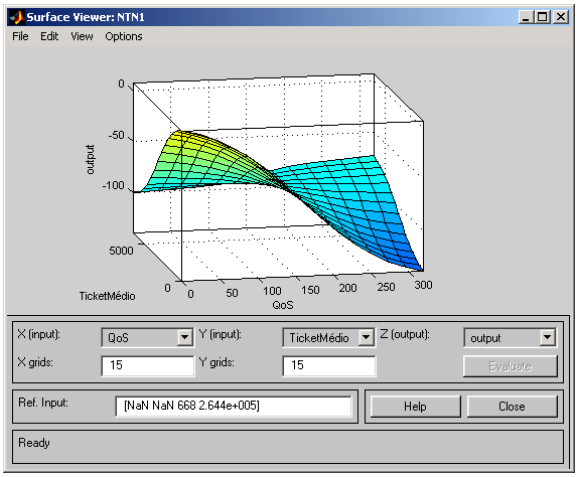

Fig. 5.6. Superfície de decisão formada pelas variáveis *Ticket Médio* e *QoS*

VI. Resultados da Análise Sintática e Léxica da Gramática Gerada pela *AFNTN*

GNU BISON é um *software livre* gerador de parser, escrito para o projeto GNU, e disponível para quase todos os sistemas operacionais. É compatível com o YACC, e oferece muitas melhorias se comparado com este *software*. Ele é utilizado em conjunto com o analisador léxico FLEX (FLEX lexical analyser).

Com a utilização dos programas BISON e FLEX foi implementado um analisador léxico e sintático para documentos que descrevem as regras *fuzzy* do tipo "E".

Como primeiro passo, foi configurado um arquivo com a descrição léxica da linguagem e que foi utilizado como entrada para o analisador léxico FLEX. O quadro 6.1 representa o segmento principal do arquivo de configuração utilizado pelo FLEX para definição dos *tokens* da linguagem.

```
se                      return RESERVED_WORD_IF;
e                       return RESERVED_WORD_AND;
pertence                return RESERVED_WORD_BELONG;
entao                   return RESERVED_WORD_SO;
:=                      return ASSIGNMENT_OPERATOR;
;                       return SEMICOLON;
[0-9]+                  return NUMBER_VAL;
[a-zA-Z]+[a-zA-Z0-9]*   return IDENTIFIER_OR_ALPHANUM_VAL;
\n                      /* ignore end of line */;
[ \t]+                  /* ignore whitespace */;
```

Quadro 6.1 Fragmento do arquivo de configuração léxica para o FLEX

Para a implementação do analisador sintático foi elaborado um arquivo de regras de sintaxe, segundo a formatação definida pelo BISON. Por esta configuração foram estabelecidas as regras sintáticas para a elaboração de programas contemplando as regras *fuzzy* do tipo "E". O quadro 6.2 representa o segmento principal do arquivo de configuração utilizado pelo BISON para a geração do analisador sintático.

```
commands:
        adaptative_service_discovery SEMICOLON
        |commands adaptative_service_discovery SEMICOLON
        ;
adaptative_service_discovery:
        attribute_pertinency_statements RESERVED_WORD_SO service_pertinency_statement
        ;
attribute_pertinency_statements:
        first_attribute_pertinency_statement
        |
        attribute_pertinency_statements RESERVED_WORD_AND attribute_pertinency_statement
        ;
first_attribute_pertinency_statement:
        RESERVED_WORD_IF attribute_pertinency_statement
        ;
attribute_pertinency_statement:
        IDENTIFIER_OR_ALPHANUM_VAL ASSIGNMENT_OPERATOR attribute_value
        RESERVED_WORD_BELONG IDENTIFIER_OR_ALPHANUM_VAL
        ;
attribute_value:
        NUMBER_VAL
        |
        IDENTIFIER_OR_ALPHANUM_VAL
        ;
service_pertinency_statement:
        IDENTIFIER_OR_ALPHANUM_VAL RESERVED_WORD_BELONG IDENTIFIER_OR_ALPHANUM_VAL
        ;
```

Quadro 6.2. Fragmento do arquivo de configuração sintática da linguagem utilizado pelo gerador BISON

Adicionalmente, foi utilizada uma extensão ao BISON, denominada YAXX (YAcc eXtension to XML). Com a ferramenta a YAXX foi possível a implementação de um analisador léxico e sintático que produz como saída uma representação XML do arquivo de regras utilizado como parâmetro de entrada. O quadro 6.3 apresenta um fragmento do arquivo XML gerado pelo YAXX.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="yaxx.xsl"?><!DOCTYPE commands SYSTEM "yyyaxx.dtd">
<yaxx:commands xmlns:yaxx="urn:YAcc-Xml-eXtension">
<yaxx:commands>
<yaxx:adaptative_service_discovery>
<yaxx:attribute_pertinency_statements>
<yaxx:attribute_pertinency_statements>
<yaxx:attribute_pertinency_statements>
<yaxx:first_attribute_pertinency_statement>
<yaxx:RESERVED_WORD_IF>se</yaxx:RESERVED_WORD_IF>
<yaxx:attribute_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>QoS</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:ASSIGNMENT_OPERATOR>:=</yaxx:ASSIGNMENT_OPERATOR>
<yaxx:attribute_value>
<yaxx:NUMBER_VAL>150</yaxx:NUMBER_VAL>
</yaxx:attribute_value>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Funcao_Pertinencia_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
</yaxx:attribute_pertinency_statement>
</yaxx:first_attribute_pertinency_statement>
</yaxx:attribute_pertinency_statements>
<yaxx:RESERVED_WORD_AND>e</yaxx:RESERVED_WORD_AND>
<yaxx:attribute_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>PrecoMedio</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:ASSIGNMENT_OPERATOR>:=</yaxx:ASSIGNMENT_OPERATOR>
<yaxx:attribute_value>
<yaxx:NUMBER_VAL>100</yaxx:NUMBER_VAL>
</yaxx:attribute_value>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Funcao_Pertinencia_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
</yaxx:attribute_pertinency_statement>
</yaxx:attribute_pertinency_statements>
<yaxx:RESERVED_WORD_AND>e</yaxx:RESERVED_WORD_AND>
<yaxx:attribute_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>VolHandoverVertical</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:ASSIGNMENT_OPERATOR>:=</yaxx:ASSIGNMENT_OPERATOR>
<yaxx:attribute_value>
<yaxx:NUMBER_VAL>200</yaxx:NUMBER_VAL>
</yaxx:attribute_value>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Funcao_Pertinencia_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
</yaxx:attribute_pertinency_statement>
</yaxx:attribute_pertinency_statements>
<yaxx:RESERVED_WORD_SO>entao</yaxx:RESERVED_WORD_SO>
<yaxx:service_pertinency_statement>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Servico_S</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
<yaxx:RESERVED_WORD_BELONG>pertence</yaxx:RESERVED_WORD_BELONG>
<yaxx:IDENTIFIER_OR_ALPHANUM_VAL>Classe_1</yaxx:IDENTIFIER_OR_ALPHANUM_VAL>
```

Quadro 6.3. Fragmento do arquivo XML produzido pelo analisador léxico e sintático gerado via ferramentas FLEX, BISON e YAXX

VII. CONCLUSÃO

A linguagem denominada de *Adaptive Dynamic Discovery Language*, gerada a partir da combinação de regras do tipo *"E"*, tem como princípio construtivo a utilização da recursividade, como ilustrado na figura 7.1, a seguir:

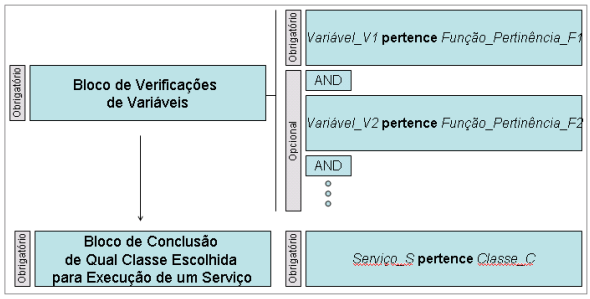

Fig 7.1 Estrutura da linguagem XML adaptativa

A recursão, nesta linguagem, foi necessária para definir qual a classe pertence um determinado serviço baseado nos parâmetros de entrada avaliados pela *AFNTN*, realizando as seguintes tarefas:

- Definição dos agrupamentos (*clusters*) de serviços através da utilização da Rede Neural Artificial do tipo SOM.
- Estruturação dos atributos não funcionais em um formato de árvore através da utilização da Árvore de Decisão Adaptativa (*AdapTree* modificado).
- Eliminação de incertezas nos dados através da utilização de um sistema de inferência *fuzzy* do tipo *Sugeno* (*ANFIS*), com a geração das regras de produção da linguagem proposta.

As regras de produção, geradas a partir do *ANFIS*, foram validadas de forma léxica e sintática gerando uma arquivo XML, conforme figura 7.2 a seguir

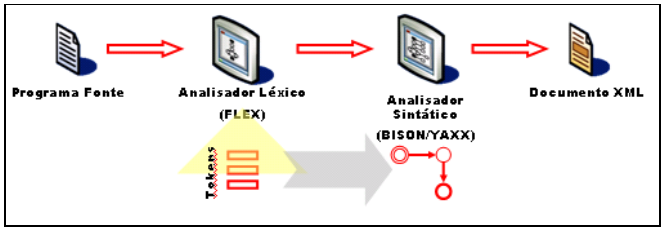

Fig. 7.2. Seqüência de passos para a validação da regras de produção da linguagem

Tal representação é bastante adequada para a implementação de serviços de busca (*discovery*) em um repositório que considere também atributos não funcionais, tornando-se um diferencial aos mecanismos atuais baseados em UDDI e WSDL, e permitirá a composição de serviços WEB através de linguagens como WSOL e OWL-S.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] PATEL, K. Improvements on WSOL Grammar and Premier WSOL Parser. Research Report. SCE-03-25 October 2003

[2] PAPADIMITRIOU, C. H.; LEWIS, H. R.; Elementos da teoria da computação. Bookmann, 2a. edição, 2004.

[3] PISTORI, H. e NETO, J.J. AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas. Anais Conferência Latino Americana de Informática - CLEI 2002. Montevideo, Uruguai, novembro, 2002

[4] PISTORI, H.; NETO, J. J.; PEREIRA, M.C.;Tecnologia Adaptativa em Engenharia da Computação. Estado da Arte e aplicações. Edição Revisada, 174p. São Paulo, 2003.

[5] PISTORI, H.; NETO, J. J.; PEREIRA, M. C. Adaptive Non-Deterministic Decision Trees: General Formulation and Case Study. INFOCOMP Journal of Computer Science, Lavras, MG, 2006 (accepted).

[6] CHENG, J.; FAYYAD, U. M.; IRANI, K. B.; QIAN, Z.; Improved decision trees: A generalized version of ID3. Proceedings of the Fifth International Conference on Machine Learning (pp. 100-106). Ann Arbor, MI: Morgan Kaufman, 1988.

[7] QUINLAN, J. R.; Comparing Connectionist and Symbolic Learning Methods. Basser Department of Computer Science; University of Sydney; Sydney NSW 2006;Australia. 1990.

[8] QUINLAN, J. R. C4.5 Programs for Machine Learning. Morgan Kaufmann. 1992.

[9] BRODLEY, C. E.; UTGOFF, P. E. Multivariate versus univariate decision trees. Technical report, Department of Computer Sciences University of Massachussetts. 1992.

[10] MITCHELL, T. Machine Learning. McGraw Hill. 1997.

[11] KOHONEN, T. Self-Organizing Maps. Springer-Verlag. 1995.

[12] ZADEH, L.A., "Fuzzy sets," Information and Control, Vol. 8, pp. 338-353, 1965.

[12] SUGENO, M., Industrial applications of fuzzy control, Elsevier Science Pub. Co., 1985.

[14] HUNHS; M. N.; Dynamic Discovery and Coordination of Agent Based Semantic WEB Services. IEEE internet computing . pp. 66-73, May-June, 2004.

[15] VESANTO, J.; ALHONIEMI, E.; HIMBERG, J.; PARHANKANGAS, J. Som Toolbox 2.0 BETA online documentation. Internet address http:// www.cis.hut.fi/projects/ somtoolbox. 1999.
[16] JANG, J.-S. R., ANFIS: Adaptive-Network-based Fuzzy Inference Systems, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 3, pp. 665-685, May 1993.
[17] MITCHELL P. M. Theory of Syntactic Recognition for Natural Languages. Cambridge: MIT Press, 1980. 335 p. ISBN 0262131498
[18] GILLES B., BRATLEY P.(1995). Fundamentals of Algorithmics. Prentice-Hall.
[19] NETO, J. J. Contribuições à metodologia de construções de compiladores. Tese de livre docência. EPUSP, 2003.
[20] CHOMSKY, N. "On certain formal properties of grammars". Information and Control (2): 137-167. 1959
[21] DUTRA, R. G.; CABRAL, E.; Aplicação de métodos de inteligência artificial em inteligência de negócios. Dissertação de Mestrado, Poli-USP, 2001.
[22] DUTRA, R. G.; MARTUCCI, M.; Adaptive Fuzzy Neural Tree Network. IEEE TLA Volume: 6, Issue: 5. 2008

Dutra, R., Engenharia Eletrônica pelo Instituto Tecnológico de Aeronáutica (1995), mestrado em Engenharia Elétrica pela Universidade de São Paulo (2001) e atualmente cursando doutorado na Politécnica da Universidade de São Paulo. Atualmente é gerente de projetos SAP, com ênfase em projetos de implementação de soluções de sistemas integrados e Inteligência de Negócios. Atuando principalmente nos seguintes temas: Sistemas de Automação, Sistemas distribuídos, Sistemas Abertos, Arquitetura Orientada a Serviços, Arquitetura Distribuída.

Silva, R., Bacharelado em Ciências da Computação pela Universidade de São Paulo (1998), pós-graduação lato senso em Administração de Empresas pela Fundação Getúlio Vargas (2005) e atualmente cursando mestrado na Politécnica da Universidade de São Paulo. Atualmente é gerente de projetos de consultoria em TI, com ênfase em projetos de implementação de soluções de sistemas na indústria de bancos. Atuando principalmente nos seguintes temas: Sistemas em Plataforma Web, Desenvolvimento Orientado a Objetos, Soluções com Tecnologia Java.

Martucci Jr., M., Engenharia Elétrica pela Universidade de São Paulo (1973), graduação em Bacharedado Em Física pela Universidade de São Paulo (1975), mestrado em Engenharia Elétrica pela Universidade de São Paulo (1977) e doutorado em Engenharia Elétrica pela Universidade de São Paulo (1982) . Atualmente é professor titular da Universidade de São Paulo. Tem experiência na área de Engenharia Elétrica , com ênfase em Eletrônica Industrial, Sistemas e Controles Eletrônicos. Atuando principalmente nos seguintes temas: Sistemas de Automação, Sistemas distribuídos, Sistemas Abertos, Arquitetura Hierarquizada, Arquitetura Distribuída.

Vicente Ruggiero, W., Diretor Presidente da Scopus Tecnologia Ltda, Engenheiro Eletricista pela Escola Politécnica da Universidade de São Paulo em 1971, Mestre em Engenharia Elétrica pela EPUSP-1975, PhD em Ciência da Computação pela UCLA-1978, Prof. Titular do Departamento de Computação e Sistemas Digitais da Escola Politécnica da USP, Diretor Técnico do Laboratório de Arquitetura e Redes de Computadores (LARC) da Escola Politécnica da USP e Investigador Principal do Projeto TIDIA – Aprendizado Eletrônico. Na Scopus é responsável pelo desenvolvimento de sistemas transacionais seguros na Internet e coordena o processo de inovação tecnológica. Autor de mais de 100 artigos técnicos publicados em revistas e anais de congressos nacionais e internacionais.