Agentes Replicantes e Algoritmos de Eco

Marcos Euzébio

Relatório Técnico DCC–95-16

Outubro de 1995

Agentes Replicantes e Algoritmos de Eco

Marcos Euzébio *

Sumário

Os algoritmos de eco formam uma classe de algoritmos distribuídos simples, versáteis e eficientes utilizados para a obtenção de informações globais de uma forma descentralizada. Normalmente estes algoritmos são implementados utilizando-se o paradigma de troca de mensagens, ainda que sua descrição siga a metáfora de agentes replicantes. Neste artigo será apresentado o ambiente de execução, que oferece a infra-estrutura para a execução de agentes replicantes, assim como a descrição de algumas experiências iniciais obtidas na programação de agentes replicantes, incluindo a implementação de uma instância dos algoritmos de eco.

1 Introdução

Os **algoritmos de eco** [Chang 82] formam uma classe de algoritmos distribuídos utilizados para a obtenção de informações globais de uma forma descentralizada. Normalmente estes algoritmos são implementados utilizando-se o paradigma de troca de mensagens [Tanenbaum 92], ainda que sua descrição siga a metáfora de **agentes replicantes** [Manthy 87, Shoch e Hupp 82].

Neste artigo será apresentado o ambiente de execução, criado com o objetivo de oferecer um infra-estrutura simples, adequada e minimal para permitir a execução e experimentação com protótipos de agentes replicantes. Além disto será discutido a implementação de uma instância de algoritmo de eco utilizando o modelo de agentes replicantes.

No modelo de agentes replicantes as mensagens carregam não apenas dados mas também informação de controle. Assim ele oferece um ponto de vista dual ao que se costuma utilizar em sistema distribuído baseado em troca de mensagens. Aqui a abstração utilizada confere as mensagens um papel secundário, passivo mesmo. O controle do processo é feito em pontos localizados do sistema, nós, e o conteúdo computacional das mensagens não é tão poderoso. Uma ilustração destes conceitos seria o processo de colheita de néctar pelas abelhas: o modelo abstrato que estamos chamando de convencional veria o processo do ponto de vista das flores; o modelo de agentes replicantes veria o processo do ponto de vista das abelhas!

*Departamento de Ciência da Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

2 O Modelo: Agentes Replicantes

2.1 O que são Agentes?

A palavra **agente** vem sendo utilizada nos mais diferentes contextos, as vezes não muito distantes, com significados quase sempre diferentes [Wooldridge e Jennings 95]. O uso do termo começou na comunidade de Inteligência Artificial e já até existem taxonomias relativas ao termo: agentes de aconselhamento, agentes assistentes, agentes Internet, agentes de comunicação, etc [Etzioni e Weld 94, Indermauer 95, Wayner 95].

Fora da comunidade de IA o termo é utilizado com um sentido mais específico, mais fraco, sem carregar toda a semântica inerente à IA.

Entretanto, mesmo sem considerar a taxonomia de IA, espera-se que um agente cumpra certos requisitos mínimos de [Shoham 93]:

autonomia operação sem a intervenção direta de humanos;

sociabilidade capacidade de interação com outros agentes e/ou humanos;

reatividade denota uma capacidade de percepção do ambiente e habilidade de reagir as mudanças ocorrendo no mesmo e

pro-ativismo o agente além de responder ao ambiente deve possuir capacidade de iniciativa e possuir um comportamento dirigido a objetivos.

2.2 Agentes replicantes

Um **agente replicante** (em inglês: {*mobile,itinerant, migrating, etc.,*} todos os termos são usados) deve possuir as características acima e também ser capaz de percorrer uma rede (tendo um ambiente de execução apropriado) além de poder se auto-replicar quando necessário. Este modelo de agente replicante é popularmente conhecido como vírus ou *worms* e tem causado um série de transtornos aos usuários de computadores pessoais. (Para uma segunda opinião um tanto quanto complacente e benevolente a respeito dos vírus veja [Dibbel 95].)

A idéia de se usar agentes replicantes é bastante antiga e pode ter surgido assim que apareceram as primeiras redes de computadores. [Shoch e Hupp 82] descreve suas experiências no Centro de Pesquisas da Xerox em Palo Alto. Seus agentes replicantes eram escritos em *BCPL* e chegavam a interagir a nível de dispositivos de comunicação (placas de rede). Além de conseguir deixar a rede de uma centena de máquinas fora do ar de vez em quando eles desenvolveram uma série de aplicações interessantes e úteis como um sistema para teste de placas de redes.

[Borenstein 1994] trata do uso de tais agentes dentro do ambiente de correio eletrônico. Além de receber textos, audio, imagens, etc. o usuário poderá também ter acesso a mensagens interativas. Em cima deste serviço seria implementado aplicações mais sofisticadas voltadas para o uso em memória organizacional [Cesar e Wainer 94], por exemplo. Nesta mesma linha atuam outros pesquisadores, como [Majewski 95]. Alguns pontos considerados aqui são:

- autenticação/assinatura digital;
- controle de recursos, cobrança e monitoramento;
- controle de acesso e segurança;
- negociação e protocolos para a interação de agentes.

Do ponto de vista de desempenho, a implantação de certos serviços utilizando o modelo de agentes replicantes é, muitas vezes, a mais adequada. Na verdade a colocação de código de controle por si só já apresenta vantagens, daí o surgimento da linguagem/processadores PostScript e SQL.

No caso em que o serviço exige a consulta a diversos servidores, além da vantagem potencial de se reduzir o volume de dados a ser transferido pode acontecer também um redução no tempo de resposta devido a uma melhor utilização da rede, evitando-se a duplicação do tráfego de mensagens em uma mesma rota [Finney 95]. Com a capacidade de auto-replicação dos agentes replicantes aparece a possibilidade do paralalelismo e com ele novos ganhos em tempo de resposta. Estes ganhos são muito benvindos para a computação móvel [Forman e Zahorjan 94].

A introdução de novos serviços ou novas versões de serviços também é facilitada com o uso de replicantes[Reinhardt 94].

Entretanto a utilização de agentes replicantes introduz uma séria ameaça a segurança dos sistema ligados à rede. As precauções citadas acima no caso de correio eletrônico continuam válidas.

Com o objetivo de tornar viável a utilização de agentes replicantes várias pesquisas tem sido feitas e podemos citar: o desenvolvimento do ambiente/linguagem TeleScript pela General Magic [Loewenstern 94, Merel 94, Reinhardt 94, Wayner 94, White 94], que pretende ser no mundo de redes o que PostScript é no mundo das impressoras; a linguagem Obliq na DEC [Cardelli 95]; a linguagem de scripts da Apple (OSA) [Cook e Harris 95]; um projeto de adequar Safe-Tcl para a programação de agentes (Agent-Tcl) [Gallo 94]; a linguagem Java sendo desenvolvida na Sun [Sun 95]; etc. [Wayner 95] faz uma comparação entre Smalltalk, TeleScript e Safe-Tcl.

3 O Paradigma: Algoritmos de Eco

Esta seção é fortemente baseada em [Chang 82], que é um artigo seminal em computação distribuída. Os objetivos do Chang; construção de algoritmos distribuídos que obtém propriedades gerais de grafos; são mais abstratos que os nossos, mas muitas das suas idéias são bastante relevantes. Aliás, alguns de seus algoritmos para a detecção de propriedades de grafos tem uma tradução prática bastante importante. A detecção de componentes biconexas e vértices de articulação poderia ser utilizada para determinação de roteadores em uma rede de computação móvel [Parekh 94] .

O objetivo é tentar ser fiel ao espírito das idéias do autor para que o leitor perceba como é natural a adequação do paradigma de algoritmos de eco ao modelo de agentes replicantes. Entretanto em [Chang 82] existe uma preocupação em dar uma descrição baseada no modelo convencional de sistemas distribuídos.

3.1 O modelo de computação

[Chang 82] vê um sistema de computação distribuído como um grafo conexo em que cada nó é um processador e cada aresta um enlace bidirecional de comunicação. Cada processador possui memória local própria e é capaz de hospedar vários processos; assim enquanto uma aplicação esta rodando, o processador é capaz de receber, enviar mensagens e iniciar processos de controle.

Outros detalhes, talvez menos importantes dentro do escopo de nosso artigo, são: capacidade de enviar em paralelo a todos os vizinhos e memória generosa o suficiente para armazenar todas as mensagens que chegarem; não se presume nada a respeito das diferenças de velocidades de transmissão entre os processadores mas exige-se que a ordem de recepção das mensagens seja igual a ordem de envio para um dado enlace; não há perda de mensagens; não existe memória compartilhada, relógio global ou controle central; nenhum nó tem *a priori* o conhecimento do configuração total do sistema.

3.2 Esquema geral de um algoritmo de eco

As idéias básicas de um algoritmo de eco são simples:

1. A operação fundamental é a **troca de mensagens**. O **percurso** de uma grafo significa que mensagens serão passadas de um para o outro.

2. Os algoritmos de eco implementam um percurso no grafo/sistema de computação distribuído. O percurso pode ser dividido em duas fases: **avanço** e **recuo**. As mensagens utilizadas na fase de **avanço** são chamadas de **pioneiras** e as utilizadas na fase de **recuo** de **eco.**

3. Ao receber uma **pioneira** pela primeira vez um nó deverá reenvia-la para todos os seus vizinhos menos o remetente da pioneira. As pioneiras que chegarem a um nó já visitado se transformam em ecos. Ecos viajam na direção oposta às pioneiras.

4. Deve existir um mecanismo de **arbitração.** No caso de duas pioneiras chegarem ao um mesmo nó ao mesmo tempo apenas uma delas será escolhida como tal.

5. **Sincronização** é fundamental. Ao receber ecos para todas todas as pioneiras enviadas o nó deverá enviar um eco para remetente da sua pioneira. Este eco será obtido através de um mecanismo de composição dos vários ecos recebidos.

6. Pioneiras e ecos poderão conter informações obtidas a partir das partes do grafo visitado.

3.3 Outros aspectos do artigo de Chang

Após a apresentação do modelo computacional e do algoritmo genérico descrito informalmente acima Chang introduz outras definições necessárias para as análises de eficiência e provas de terminação que ele vai fazer dos seus algoritmos. A seguir ele especializa o seu algoritmo de eco para resolver os seguintes problemas de ordenação, eleição e obtenção das

componentes biconexas de um grafo. A descrição pormenorizada de tais algoritmos fogem do escopo deste artigo.

A tradução do algoritmo genérico para o modelo de agentes replicantes não é muito complicado. Na verdade ocorre até uma simplificação, uma vez que só vai existir um tipo de mensagem. A mensagem, que contém um programa/agente poderá determinar em que fase ela está, avanço ou recuo, baseado na situação do nó que ela está visitando.

4 A Plataforma de Execução

A plataforma de execução é implementada utilizando uma série de servidores. O sistema operacional utilizado foi o Unix e a linguagem Perl [Ryan 93, Schwartz 93], [Wall e Schwartz 93] que possui uma interface para *sockets*. Qualquer outra linguagem com tal interface poderia ser utilizada.

Os servidores isolados não seriam de muita valia e por isto associado a cada um deles está uma lista de vizinhos. No nosso caso particular optou por uma topologia regular; cada servidor conhece outros quatro. Para obter um grafo de diâmetro gerou-se dois circuito hamiltonianos quase-aleatórios.

Para simplificar a implementação não nos preocupamos nesta etapa com a instalação de mecanismos de segurança, controle de acesso, etc. Tais adições não funcionais podem ser feitas mais tarde se necessárias ou se o enfoque do trabalho for desviado para tais aspectos. Nesta etapa estavamos mais interessados em ter uma plataforma de execução o mais simples possível.

O servidor, após a definição de algumas variavéis de ambiente e criação de um diretório próprio fica aguardando conexões. Ao receber um agente replicante, é criado um diretório para o mesmo e repassado a um processo filho a tarefa da execução do agente replicante. Após o encerramento do agente, a área em que ele residira é removida. Agentes replicantes podem no entanto criar sub-diretórios na área do servidor; sem este recurso não seria possível a implementação dos algoritmos de eco. Perl foi escolhida para a programação dos agentes replicantes também.

4.1 Aplicação: Lista de Processos na Rede Local

A obtenção da lista global de processos que estão rodando em uma rede local pode servir muitas finalidades; é uma tarefa rotineira de administradores de rede, que fazem muito bom uso desta informação. [Maltzahn e Vollmar 95] descreve um sistema que tenta catalisar a cooperação de uma comunidade de usuários e para isto verificar quais são as ferramentas sendo utilizados por seus membros. A seguir os membros são consultados e solicitados a compartilhar seus conhecimentos com os outros colegas.

A aplicação escolhida foi esta. Entretanto a especialização do algoritmo de eco implementado como agente replicante para outras aplicações que colhem dados globais é trivial. A parte específica da aplicação muitas vezes representa o aspecto mais trivial da implementação.

Informalmente, seguindo o molde utilizado por Chang [Chang 82] a implementação de um algoritmo genérico é feita assim:

1. A **migração** dos agentes replicantes é um princípio fundamental aqui. O percurso do grafo é feito através da migração dos agentes replicantes. Isto significa apenas que cópias do código do agente em execução (a plataforma oferece isto de graça em um arquivo) são enviadas para os servidores vizinhos. Todos os agentes começam sua execução pelo começo. É possível modificar o código do agente em execução para que alguma informações de estados possam ser transportadas.

2. Existe apenas um tipo de agente replicante. Ao chegar em um servidor ele verifica se é o primeiro (independente de ser pioneiro ou eco). Se não for ele passa a se comportar como se fosse um eco.

3. Um agente replicante **pioneiro** envia cópias de seu código para todos os vizinhos. Para que o vizinho saiba quem foi o remetente ele altera estas informações do códigos que estão em linhas bem específcas do programa. Funcionam como o cabeçalho de uma mensagem de correio eletrônico.

4. O mecanismo de **arbitração** é através da verificação da existência do diretório relacionada aquele percurso (todo percurso deve ter uma identificação própria). O teste da existência de um diretório não é uma operação atômica, mas funciona.

5. A **Sincronização** é feita da seguinte forma: a execução dos ecos é serializada (*locks:* verificação da existência de arquivo). Para cada procedência foi criado um arquivo na hora do envio da pioneira. Ao final do processamento o eco verifica se todos já chegaram e neste faz a composição dos resultados, depois de efetuar o processamento local relativo a aplicação e cria um novo agente replicante para conter esta informação, além das outras mudanças que também são feitas na fase de avanço.

Além do servidor e do agente replicante é necessário um pseudo-servidor para catapultear o agente replicante na plataforma e também aguardar o resultado. O pseudo-servidor é parecido com o servidor, apenas mais simples.

5 Conclusão e Planos para o Futuro

A implementação de instâncias dos algoritmos de eco seguindo a elegante metáfora dos agentes replicantes foi interessante por vários motivos. Foi gratificante encontrar uma classe de algoritmos que casava bem com esta metáfora e que também explorava suas principais características: migração e auto-replicagem. Além disso o paradigma de algoritmos de eco serve a uma série de aplicações práticas.

A descrição de Chang mostra seu *bias* com relação ao ponto de vista convencional. Como estamos acostumados a programar segundo um modelo sequencial, ao passarmos para um sistema distribuído, é natural manter um enfoque centrado nos locais de processamento. Um diagrama que indicasse a ordem parcial entre os vários eventos colocaria em uma mesma reta os eventos que ocorreriam em cada processo. A dependência entre eventos ocorrendo em processos diferentes seria indicada pela presença de mensagens entre os mesmos. Para algumas aplicações este diagrama seria útil, para outros não.

No caso de classes de algoritmos tais como os algoritmos de eco o melhor seria traçar a ordem entre os eventos a partir da perspectiva das mensagens (agentes replicantes). A sincronização/comunicação entre os eventos ocorrendo em agentes replicantes diferentes seria através de acesso a uma memória compartilhada.

É possível que uma melhor compreensão desta dualidade dos sistemas distribuídos contribua de alguma maneira para para um melhor entendimento da arte de programá-lo.

A partir deste trabalho, dentro do contexto de programação baseada em agentes replicantes as perspectivas são bastante amplas. Vários aspectos ainda precisam ser explorados, o que explica o grande interesse na área; acadêmico e comercial. Assim temos os aspectos relacionados ao desenvolvimento de ambientes de execução como segurança, cobrança, etc.; aspectos relacionados à programação dos agentes como a questão do desenvolvimento de linguagens adequadas; aspectos relacionados ao desenvolvimento de algoritmos, inclusive tolerância a falhas; aspectos relacionados à criação de interfaces "inteligentes" adequadas aos usuários leigos, etc.

Dentro de nossa área de pesquisa relacionada a aspectos teóricos e práticos de memória organizacional parece que o uso de agentes replicantes será muito importante. A verificação da realidade desta hipótese, assim como a determinação do escopo e abrangência da mesma é uma das nossas preocupações futuras.

Referências

[Borenstein 1994]
N. S. Borenstein. The Safe-Tcl Language for Enabled Mail. *ULPAA '94,* Barcelona, Espanha, 1994,
http://minsky.med.Virginia.Edu/sdm7g/Projects/Python/safe-tcl.

[Cardelli 95]
L. Cardelli. *Obliq,* 1995,
http://www.research.digital.com/SRC/Obliq/Obliq.html.

[Cesar e Wainer 94]
F. L. Cesar e J. Wainer. vIBIS: A Discussion and Voting System. *Journal of the Brazilian Computer Society,* pp. 36–43, novembro de 1994.

[Chang 82]
E. J. H. Chang. Echo Algorithms: Depth Parallel Operations on General Graphs. *IEEE Transactions on Software Engineering,* Vol. SE-8, No. 4, julho de 1982.

[Christiansen 95]
T. Christiansen. TCP::Server Module. *Newsgroup comp.lang.perl,* Março de 1995, news:comp.lang.perl.

[Cook e Harris 95]
W. R. Cook e W. H. Harris. *The Open Scripting Architecture: Automating, Integrating and Customizing Applications,*
http://minsky.med.Virginia.Edu/sdm7g/Projects/Python/OSA, 1995.

[Dibbel 95]
J. Dibbel. Viruses are good for you. *Wired Magazine,* Fevereiro 1995.

[Etzioni e Weld 94]
O. Etzioni e D. Weld. A Softbot-Based Interface to the Internet. *Communications of the ACM,* julho de 1994,
http://www.cs.washington.edu/research/projects/softbots/www/softbots.html.

[Finney 95]
H. Finney et all. Why Should Agents Roam? *Software Agents Mailing List*, março de 1995,
mailto:agents-requests@SunLabs.Eng.Sun.Com.

[Forman e Zahorjan 94]
G. H. Forman e J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer,* pp. 38–47, abril de 1994.

[Gallo 94]
F. Gallo. Agent-Tcl: A White Paper. Safe-Tcl Mailing List, dezembro de 1994,
http://minsky.med.Virginia.Edu/sdm7g/Projects/Python/safe-tcl/agent-tcl.txt.

[Indermauer 95]
K. Indermauer. Baby Steps. *Byte,* pp. 97–104, março de 1995.

[Loewenstern 94]
A. Loewenstern. Info on Telescript. *Software Agents Mailing List*, dezembro de 1994,
mailto:agents-requests@SunLabs.Eng.Sun.Com.

[Majewski 95]
The Programming Language Safe-Phyton, 1995,
http://minsky.med.Virginia.Edu/sdm7g/Projects/Python/SafePython.html.

[Maltzahn e Vollmar 95]
C. Maltzahn e D. Vollmar. Toolbox: A Living Directory For Unix Tools Owned by the Community. submetido à *HCISS 95.*

[Manthy 87]
M. J. Manthy. Hierarchy in Sequential and Concurrent Systems *or* What's in a Reply? In *The Characteristics of Parallel Algorithms,* editado por L. H. Jamieson et all, The MIT Press, 1987.

[Merel 94]
P. Merel. Telescript == MOO? *Magic Cap Discussion List,* dezembro 1994,
mailto:MagicCap@BrownVm.Bitnet.

[Parekh 94]
A. K. Parekh. Selecting Routers in Ad-hoc Wireless Networks. *Journal of the Brazilian Computer Society,* pp. 13–21, novembro de 1994.

[Reinhardt 94]
A. Reinhardt. The Network with Smarts. *Byte,* pp. 51–64, outubro 1994.

[Ryan 93]
P. M. Ryan. *Introduction to Perl,* 1993,
ftp://ftp.khoros.unm.edu/pub/perl/doc/introduction.ps.gz.

[Schwartz 93]
R. L. Schwartz. *Learning Perl,* O'Reilly & Associates, Inc., 1993.

[Shoch e Hupp 82]
J. F. Shoch e J. H. Hupp The "Worm" Programs – Early Experience with a Distributed Computation. Communications of the ACM, pp. 172–180, março 1982.

[Shoham 93]
Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence,* vol. 60, pp. 51–92, 1993.

[Sun 95]
Sun Microsystems. *The Java Language White Paper,*
http://java.sun.com/1.0alpha2/doc/overview/java/index.html

[Tanenbaum 92]
Modern Operating Systems, Prentice-Hall, 1992.

[Wall e Schwartz 93]
H. Wall e R. L. Schwartz. *Programming in Perl,* O'Reilly & Associates, Inc., 1993.

[Wayner 94]
P. Wayner. Agents Away. *Byte,* pp. 113–118, maio de 1994.

[Wayner 95]
P. Wayner. Free Agents. *Byte,* pp. 105–114, março de 1995.

[White 94]
J. E. White. *Telescript Technology: The Foundation for the Electronic Marketplace,* General Magic White Paper, 1994.

[Wooldridge e Jennings 95]
M. Wooldridge e N. R. Jennings (editores). *Intelligent Agents: Theories, Architectures and Languages.* Lectures Notes in AI, Vol. 890, janeiro de 1995, Springer-Verlag.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

95-13 **Modelos de Computação Paralela e Projeto de Algoritmos,** *Ronaldo Parente de Menezes e João Carlos Setubal*

95-14 **Vertex Splitting and Tension-Free Layout,** *P. Eades, C. F. X. de Mendonça N.*

95-15 **NP-Hardness Results for Tension-Free Layout,** *C. F. X. de Mendonça N., P. Eades, C. L. Lucchesi, J. Meidanis*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br